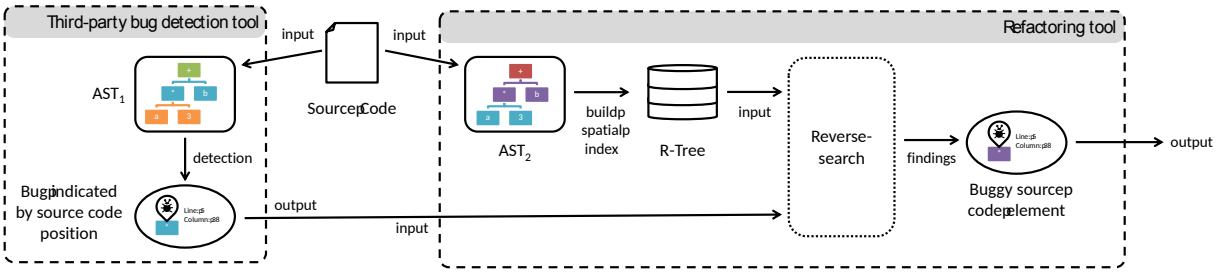
Finding Matching Source Code Elements in an AST Using Position Information

Gábor Szőke

To decrease software maintenance cost software development companies use static source code analysis techniques. Static analysis tools are capable of finding potential bugs, anti-patterns, coding rule violations and they can enforce coding style standards. To achieve this the tools create a hierarchical representation of the source code called *Abstract Syntax Tree* (AST). In this structure each node of the tree denotes a construct occurring in the source code. The AST can be represented in various ways and most static analysis tools build their own AST representation, typically which fits them the best for their given purpose.

Sometimes it is necessary to create a mapping between two different ASTs, e.g. finding matching source code elements in an AST using just position information. We met this problem in a project where we developed a refactoring tool which was meant to be able to refactor coding issues based on source code position. In this study, we present an approach which addresses this problem. Our solution takes source code position and type information of a node from one AST and uses this information to *reverse search* the matching node on the other AST. To make reverse searching possible, a *spatial database* is used, which is created by transforming the source code into *geometric space*. Line numbers and column positions from the AST are used to define areas. These areas are used to create *R-trees*, where area based reverse searching is possible.



To evaluate our approach we use the output of the *PMD* [1] source code analyzer tool and use its bug report's position information as the source data and then we pick the *Columbus* [2] AST as target AST to find the matching source code element in the syntax tree. We used this technique in a project, where the found element was used as input in an automated refactoring transformation on the target AST. The transformation modified the syntax tree in a way which fixed the bug reported by the PMD static analyzer tool. As the final step, the refactored version of the source code got generated from the target AST and developers could review the changes.

The evaluation showed that our approach can be adapted to a real-life scenario, and it can provide viable results. Our tool was used in a project where it assisted in more than 4,000 automated refactorings covering systems ranging from 200 to 2,500 kLOC.

References

[1] PMD website: https://pmd.github.io/

[2] Ferenc, Rudolf and Beszédes, Árpád and Tarkiainen, Mikko and Gyimóthy, Tibor, *Columbus – Reverse Engineering Tool and Schema for C++*, Proceedings of the 18th International Conference on Software Maintenance (ICSM'02), pp. 172-181, 2002.