Building Dependency Graph for Slicing Erlang Programs

Melinda Tóth and István Bozó

Program slicing [1] is the most well-known method used under impact analysis. Different methods are available to perform program slicing (e.g. dataflow equations, information flow relations, dependency graphs), but the most popular from them are based on the dependency graphs of the program to be sliced [2]. These graphs include data and control dependencies of the program.

There are many forms of using program slicing during the software life-cycle. It can be used in debugging, optimization, program analysis, testing or other software maintenance tasks. For example, using program slicing to detect the impact of a change on a certain point on the program, could help to the programmer to find those test cases which could be affected by a program code change.

Our goal is to adopt the existing methods and to develop new algorithms for program slicing of programs written in a dynamically typed functional programming language, Erlang [3]. We describe the algorithms to define Data Dependency Graphs and Control Dependency Graphs for Erlang, and how to build Program/System Dependency Graphs from them. The dependency graphs are usable to reach the mentioned goal and transform the program slicing to a graph reachability problem. We want to calculate the forward slices of the program, especially for those program parts which are changed after a refactoring [4]. Calculating the forward slices could help the programmers to reduce the number of test cases to be rerun after the transformation.

Acknowledgements

This work was supported by TECH_08_A2-SZOMIN08.

References

[1] M. Weiser: Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method. *PhD thesis, University of Michigan, Ann Arbor, MI, 1979.*

[2] S. Horwitz, T. Reps, and D. Binkley: Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems, 12(1):3546, January 1990.*

[3] Erlang Homepage, *https://www.erlang.org*

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts: *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.