Title: Automatic non-functional testing and tuning of configurable generators

Abstract:
Generative software development has paved the way for the creation of multiple generators (code generators and compilers) that serve as a basis for automatically producing code to a broad range of software and hardware platforms. With full automatic code generation, users are able to rapidly synthesize software artifacts for various software platforms. In addition, they can easily customize the generated code for the target hardware platform since modern generators (i.e., C compilers) become highly configurable, offering numerous configuration options that the user can apply. Consequently, the quality of generated software becomes highly correlated to the configuration settings as well as to the generator itself.

In this context, it is crucial to verify the correct behavior of generators. Numerous approaches have been proposed to verify the functional outcome of generated code but few of them evaluate the non-functional properties of automatically generated code, namely the performance and resource usage properties.

This thesis addresses three problems:

(1) Non-functional testing of generators: We benefit from the existence of multiple code generators with comparable functionality (i.e., code generator families) to automatically test the generated code. We leverage the metamorphic testing approach to detect non-functional inconsistencies in code generator families by defining metamorphic relations as test oracles. We define the metamorphic relation as a comparison between the variations of performance and resource usage of code, generated from the same code generator family. We evaluate our approach by analyzing the performance of HAXE, a popular code generator family. Experimental results show that our approach is able to automatically detect several inconsistencies that reveal real issues in this family of code generators.

(2) Generators auto-tuning: We exploit the recent advances in search-based software engineering in order to provide an effective approach to tune generators (i.e., through optimizations) according to user's non-functional requirements (i.e., performance and resource usage). We also demonstrate that our approach can be used to automatically construct optimization levels that represent optimal trade-offs between multiple non-functional properties such as execution time and resource usage requirements. We evaluate our approach by verifying the optimizations performed by the GCC compiler. Our experimental results show that our approach is able to auto-tune compilers and construct optimizations that yield to better performance results than standard optimization levels.

(3) Handling the diversity of software and hardware platforms in software testing: Running tests and evaluating the resource usage in heterogeneous environments is tedious. To handle this problem, we benefit from the recent advances in lightweight system virtualization, in particular container-based virtualization, in order to offer effective support for automatically deploying, executing, and monitoring code in heterogeneous environment, and collect non-functional metrics (e.g., memory and CPU consumptions). This testing infrastructure serves as a basis for evaluating the experiments conducted in the two first contributions.