An Approach to Detect Portability Issues of the Include Dependencies in C++

Bence Babati, Norbert Pataki, Zoltán Porkoláb

Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers

{babati, patakino, gsd}@caesar.elte.hu

The C++ Standard Template Library is the flagship example for libraries based on the generic programming paradigm. The usage of this library is intended to minimize classical C/C++ errors, but does not warrant bug-free programs. Furthermore, many new kinds of errors may arise from the inaccurate use of the generic programming paradigm, like dereferencing invalid iterators or misunderstanding remove-like algorithms [**?**].

Unfortunately, the C++ Standard does not define which standard header includes another standard headers. It is easy to write code that works perfectly on an implementation but fails to compile with another implementation of STL. These unportable codes should be result in compilation error with every STL implementation [**?**]. However, in this case the compiler does not warn us that this code is erronous.

We present our approach how to detect the portability issues. Our approach uses dependency graphs. We present how to build the dependency graphs to discover include problems. We have developed a tool that is able to detect these portability issues based on the source code with this approach. The tool takes advantage of the Clang compiler infrastructure [**?**]. Clang is the most appropriate approach to develop tools that analyze the software code based on its abstract syntax tree (AST) [**?**]. Our software builds dependency graph from the included header files. The tool detects which has not been included intentionally or which header has been included needlessly. The needless includes may increase the compilation time [**?**].

References

[1] Horváth, G., Pataki, N.: *Clang matchers for verified usage of the C++ Standard Template Library*, Annales Mathematicae et Informaticae, Vol. **44** (2015), pp. 99–109.

[2] Lattner, C.: *LLVM and Clang: Next Generation Compiler Technology*, The BSD Conference, 2008.

[3] Meyers, S.: "Effective STL - 50 Specific Ways to Improve Your Use of the Standard Template Library", Addison-Wesley (2001)

[4] Mihalicza J.: *How #includes Affect Build Time in Large Systems*, in Proc. of 8th International Conference on Applied Informatics (ICAI 2010), Volume II., pp. 343–350.

[5] Pataki, N.: *C++ Standard Template Library by Safe Functors*, in Proc. of 8th Joint Conference on Mathematics and Computer Science, MaCS 2010, Selected Papers, pp. 363–374.