13th International Conference on Applied Informatics
Eszterházy Károly Catholic University
Eger, Hungary, February 19–21, 2026

# Static analysis possibilities for regular expressions in C++

**Tibor Teodor Fürész, Kristóf Umann, Zoltán Porkoláb**

ELTE Eötvös Lóránd University, Budapest, Hungary. Faculty of Informatics,
Department of Programming Languages and Compilers
jaoycb@inf.elte.hu, szelethus@inf.elte.hu, gsd@inf.elte.hu

## 1. Introduction

Regular expressions are an everyday part of software programming. Common scenarios like input parsing, file lookup, and search and replace are just a few examples, and in truth, it is hard to fathom a program with any meaningful complexity that can omit them. Most tools, like grep, that build regular expressions also come with robust techniques to check for their correctness.

Most of the current libraries works on the way that regular expressions implemented using state machines. The state machine is usually built up in the constructor, based on a regular expression pattern received as the construction parameter. While this is a time consuming process, later repeated use of the state machine is relatively cheap. However, the constructor parameter should be matched for the correct syntax of the regular expression. This error checking happens dynamically inside the constructor when the state machine is constructed.

## 2. Problem statement

During development, regular expressions are notoriously hard to get right. Their syntax is unintuitive, hard to read, and more importantly, to write [8]. Testing the expressions is also a challenge, as test effectiveness heavily depends on the test coverage. It is easy to see that test frameworks usually miss the same corner cases, which programmers also likely forget to implement correctly.

The trade-offs between dynamic and static analysis have been widely studied and publicized [2, 5–7, 9]. Briefly, most forms of dynamic analysis require the

**Table 1.** Number of regex constructors found in open-source projects.

| | |
|---|---|
| Constructors | 296 |
| Constructors with hard-coded strings | 234 |

execution of the program and also precise input parameters. On the other hand, static analyzers usually do not require the concrete execution of the program or any input parameters, offering a unique advantage for detecting programming errors in the early stages of development [1]. As bugs that are caught early are faster, easier, and cheaper to fix [3], there are clear advantages for pursuing static analysis whenever possible.

# 3. Results

This paper discusses the possibilities of static analysis of regular expressions with a particular focus on C/C++. Despite the age and the wealth of knowledge accumulated in the field of regular expressions as of writing, we are not aware of any works that statically check the correctness of types like `std::regex` or `boost::regex`.

However, static analysis could be applied for regular expressions only if the expression is available at compile-time. Before any effort is made in this direction we have to understand how programmers use regular expressions, especially in which rate they apply patterns known at compilation time.

We applied clang-tidy [4] static analysis tool to detect regular expression objects in large open-source C++ projects and to determine in which ratio those objects initialized with compile-time constant strings. We found that in typical C++ projects most of the regular expression objects are initialized based on hard-coded string patterns, therefore static analysis of them is viable.

The paper describes the method, the algorithm, a prototype implementation, and the measurement results on large open-source projects. Our results proved that 60-80% of the regular expression objects are initialized with hard-coded string patterns, as seen in Table 1, therefore, a static analysis-based regular expression checker would be highly valuable.

# Acknowledgment

Project no. C2314106 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the KDP-2023 funding scheme.

Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal

# References


[1] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, J. Penix: *Using Static Analysis to Find Bugs*, IEEE Software 25.5 (Sept. 2008), pp. 22–29, doi: 10.1109/ms.2008.130, url: https://doi.org/10.1109%2Fms.2008.130.

[2] Á. Balogh, R. Szalay: *On the Applicability of Static Analysis for System Software using CodeChecker*, in: 2024 7th International Conference on Software and System Engineering (ICoSSE), 2024, pp. 15–22, doi: 10.1109/ICoSSE62619.2024.00011.

[3] B. Boehm, V. R. Basili: *Software Defect Reduction Top 10 List*, Computer 34.1 (Jan. 2001), pp. 135–137, issn: 0018-9162, doi: 10.1109/2.962984, url: http://dx.doi.org/10.1109/2.962984.

[4] Clang Tidy: *Clang-Tidy*, https://clang.llvm.org/extra/clang-tidy/ (last accessed: 28-02-2019)., 2019.

[5] M. D. Ernst: *Static and dynamic analysis: Synergy and duality*, in: WODA 2003: ICSE Workshop on Dynamic Analysis, 2003, pp. 24–27, doi: 10.1109/icse.2003.1201290.

[6] S. Hallem, B. Chelf, Y. Xie, D. Engler: *A System and Language for Building System-specific, Static Analyses*, PLDI '02 (2002), pp. 69–82, doi: 10.1145/512529.512539, url: http://doi.acm.org/10.1145/512529.512539.

[7] S. Heckman, L. Williams: *A systematic literature review of actionable alert identification techniques for automated static code analysis*, Information and Software Technology 53.4 (2011), Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing, pp. 363–387, issn: 0950-5849, doi: https://doi.org/10.1016/j.infsof.2010.12.007, url: https://www.sciencedirect.com/science/article/pii/S0950584910002235.

[8] L. G. Michael, J. Donohue, J. C. Davis, D. Lee, F. Servant: *Regexes are Hard: Decision-Making, Difficulties, and Risks in Programming Regular Expressions*, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 415–426, doi: 10.1109/ASE.2019.00047.

[9] N. Nagappan, T. Ball: *Static Analysis Tools As Early Indicators of Pre-release Defect Density*, ICSE '05 (2005), pp. 580–586, doi: 10.1145/1062455.1062558, url: http://doi.acm.org/10.1145/1062455.1062558.