P3893R0 2025-10-24 Audience: EWG, LEWG
Reply-To: michaelrfairhurst@github.com

The CppCon 2025 Talk on Contracts and CodeQL in Context

Mike Fairhurst
Microsoft

CppCon 2025 featured a talk on CodeQL and contracts[Fairhurst25], and it is important to note that the portions of this talk presented by GitHub are not an endorsement of P2900 and does not address all of Microsoft's concerns. Notably, the prototype static analysis tool demonstrated in the talk is currently designed to analyze traditional assertions, not contract specifiers as proposed by P2900. While P2900's contract specifier syntax may remove one barrier this tool faces related to declarations vs definitions, it notably may raise other issues with static analysis tooling. This talk is therefore not an endorsement of P2900R14 or future revisions of P2900, and prior concerns raised by Microsoft remain unaddressed.

1. Background on CodeQL, Contract Analysis

CodeQL is a static analysis engine built by GitHub that supports analysis of many different programming languages, such as Java, Javascript, Ruby, and C/C++. CodeQL allows users to write static analyses in the form over queries and treats code as data. For C/C++ projects, this means observing a build of a target program to produce a WORM (write-once-read-many) database. In addition to supporting custom queries, CodeQL ships with a default set of security queries for C++.

One suite of custom queries was recently presented at CppCon[Fairhurst25]. This experimental project[CodeQL-Z3] uses CodeQL to discover function preconditions expressed as code in the form of assert() and BSLS_ASSERT from BDE, not contract specifiers. Other queries in this project discover call sites to functions with such preconditions and use SSA (static single assignment)-based range analysis to build SMT (Satisfiability-Modulo-Theories) proofs that these preconditions will or will not hold. Microsoft's Z3 constraint solver is used to verify these proofs or discover counterexamples to them.

These preconditions do not need to be declared in the form of contracts specifiers as proposed in P2900R14 for the tool to perform its analysis.

Reply-To: michaelrfairhurst@github.com

2. CodeQL and Contract Specifiers

P2900R14 would have several impacts on how to best utilize CodeQL. Some of these impacts may be positive while others are negative.

For CodeQL to produce high quality static analysis results, the database and the queries of that data must together meet certain criteria. Firstly, these components must accurately model the runtime behavior of the program under analysis (program modeling). Secondly, they must accurately reflect the runtime behavior of dependencies (dependency modeling). For example, the behavior of dynamically linked libraries must be modeled without knowing an exact implementation. Thirdly, analysis should produce results that project owners find valuable (useful reporting).

Contract specifiers as proposed in P2900R14 allows library authors to declare preconditions in their APIs in header files, which can simplify dependency modeling for CodeQL in this case[Fairhurst25]. Source-based analyzers such as CodeQL can only discover preconditions inside of source code which is available to them, and dependency header files are more likely to be available. Our tool which we presented at Cppcon requires users to build “Spec dbs” of separately build dependencies in order to discover contracts within dependency cpp files. Contract specifiers may not be sufficient to remove this requirement. For example, in our talk we presented an exploration of directly translating SSA to SMT, and inlining calls to perform better analyses than contract specifiers offer alone. Future versions of this tool may analyze source code to detect codepaths that violate postconditions, or contract_assert statements, or paths that throw exceptions.

P2900R14 raises questions related to program modeling and useful reporting, since contracts can be disabled at runtime. Analysis tools must choose whether to treat contract specifiers as enabled or disabled. Both options have tradeoffs, and both options may be surprising to users. It may be dangerous to suppress an overflow error because of a disabled assertion, and it may interfere with the goal of useful reporting to show such overflow errors to users. This ambiguity does not exist for c-style assertions, which are definitively enabled or disabled at compilation time, and alter program control flow graph in a manner that is directly handled by typical range analysis techniques.

P2900R14 allows for nearly arbitrary expressions inside of a precondition body, including expressions with side-effects. This is also the case for c-style assertions. Fully verifying these types of programs is a tremendous challenge. The purpose of our tool is to find **some** bugs in **some** programs. Our tool will not cover all types of contracts that project authors declare.

Reply-To: michaelrfairhurst@github.com

3. Conclusion

The new tooling presented by Mike Fairhurst and Peter Martin [Fairhurst25] is a small part of what CodeQL offers in the realm of C/C++ static analysis. While certain parts of this tool may be easier to operate under the proposed P2900R14 contract specifier syntax, other more fundamental aspects of CodeQL may face new questions and difficulties around program modeling, dependency modeling, and useful reporting. This new tool is still a prototype and a proof of concept and should not be construed as a basis for evaluating the overall feasibility of static analysis related to P2900R14.

[Fairhurst25] “Catching Bugs Early: Validating C++ Contracts With Static Analysis” https://www.youtube.com/watch?v=3DDqDKaKmio, Mike Fairhurst and Peter Martin

[CodeQL-Z3] https://github.com/advanced-security/codeql-contracts-smt-z3

[P2900R14] “Contracts for C++” https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2025/p2900r14.pdf, Joshua Berne et al.