The development of high-quality software is a tough task: ECLAIR has been designed to assist development and quality assurance teams to realize that quality, as well as helping quality control teams assess its achievement.

ECLAIR is a powerful platform for the automatic analysis, verification, testing and transformation of C and C++ programs (with an ongoing extension to Java and a planned extension to other languages).

ECLAIR is very flexible and highly configurable. It can support your software development workflow and environment, whatever they are. You can ask us to bend it to your precise needs or do that yourself.

ECLAIR is suitable for light verification tasks that can be run right on the developer’s desktop, as well as for tough semantics-based analyses to be run in batch mode.

ECLAIR is fit for use in mission- and safety-critical software projects: it has been designed from the outset so as to exclude false negatives unless the user’s configuration explicitly asks for them.

ECLAIR is developed in a rigorous way and carefully checked also with extensive internal test suites (tens of thousands of test cases) and industry-standard validation suites, such as ACE SuperTest and the Plum Hall Validation Suite for C.

ECLAIR is developed by a passionate team of programming language and software verification experts.\(^1\)

Do not hesitate to let us have your feedback: you may be surprised to discover just how much your suggestions matter to us.

1 Supported Applications

Here are some applications that run on top of ECLAIR platform.

Automatic Checking of Coding Standards

ECLAIR provides support for automatically checking conformance with respect to a number of widely used coding standards, among which:

---
\(^1\) Among them is the Italian representative within ISO JTC1/SC22/WG14, the international standardization working group for the programming language C.
In addition to these, BUGSENG has developed checkers to verify conformance with respect to several other coding standards, including proprietary company standards and project-specific standards. Coding rules are enforced using very general and accurate checkers, which operate on the precise sequences of tokens and abstract syntax trees that are manipulated by the compiler.\(^2\) Coupled with the fact that ECLAIR always checks each rule in the appropriate context (at the token, declaration, translation unit, whole program or whole project levels), this makes sure that the checkers for decidable rules are exact (neither false positives nor false negatives). For undecidable rules, ECLAIR provides different checkers characterized by different solutions to the tradeoff among computational complexity, number of false positives and number of false negatives. In any case, when false negatives are possible, they are always clearly and unambiguously delimited.

ECLAIR can be configured to produce all sorts of violation reports:

- for immediate or later browsing using popular IDEs like Eclipse, Microsoft Visual Studio®, IAR Embedded Workbench®, Texas Instruments Code Composer Studio™, Keil µVision®, or any suitable editor (a very powerful, web-based browser is currently in beta-test);
- for the automatic insertion into issue-tracking systems or any other database;
- for the automatic production of compliance matrices required to meet industrial standards and guidelines such as IEC 61508, ISO 26262 (automotive), CENELEC EN 50128 (railways), DO-178B/C (aerospace), IEC 60880 (nuclear power), IEC 62304 and FDA norms (medical).

**Metrics**

For assessing the complexity, readability and maintainability of software, ECLAIR can provide comprehensive metrics about the code including the fundamental set defined by the HIS Source Code Metrics document and many of the requirements of the SQALE Method quality model. These measures may be incrementally reported, showing exactly where in the code the value was computed, or aggregated (e.g., maximized, summed, averaged) over a single function, translation unit, program or the whole project.

**Semantic Analysis**

We are equipping ECLAIR with powerful constraint propagation, symbolic model checking, and abstract interpretation engines. Their combined use allows the realization of different compromises between computational complexity and precision, so that each user can select an optimal cost-benefit tradeoff with respect to her objectives. On the developer’s desktop, quick analyses are the most appropriate; but when the alternative is between, e.g., proving absence of run-time errors by hand or by machine, 12 hours of computation are preferable to a man-month of work: ECLAIR has been designed to support the entire spectrum between these two extremes.

\(^2\)This is in sharp contrast with checkers that are based on pattern matching and imprecise parsing. If, on the one hand, they can deal with programs that do not compile, on the other hand they are plagued by a high number of false positives and, most importantly, false negatives, something that makes them unsuitable to safety-critical and mission-critical contexts. Generally speaking, beware of tools based on obsolete technology: directives such as 90/385/EEC and 93/42/EEC in the medical sector assert that compliance can only be claimed if the generally acknowledged state-of-the-art is followed.
Automatic Unit Test Case Generation

ECLAIR can automatically synthesize minimal sets of unit test inputs that reach a specified coverage criterion (or prove that such coverage cannot be reached due to infeasible conditions in the program). If a human or mechanized “oracle” is available to predict the expected output, then complete unit tests can be synthesized in a fraction of the time required for manual generation.

Semantic Matcher and Patcher

ECLAIR can perform very sophisticated program transformations by automatically finding program regions in the source identified by syntactic and semantics-based criteria, optionally replacing the matching program fragment with a parametrized substitution.

Other Applications

Simplifiers, obfuscators, translators, … these are just a few examples of the plethora of applications that, because of ECLAIR’s highly modular design, can be developed relatively easily at the right level of abstraction. For instance, ECLAIR includes a C simplifier: it reads any valid C translation unit and (taking into account the implementation-defined aspect of the considered C implementation) it writes equivalent but much simplified C code (e.g., where side effects have been removed from expressions).

2 Proper Integration with the Toolchain

ECLAIR intercepts every invocation of the toolchain components (compilers, linker, assembler, archive manager) and it automatically extracts and interprets the options that the build system has passed to them. This allows for the seamless integration with any build system. Moreover, the user does not need to engage in error-prone activities such as:

- specifying which files compose the application and where the right header files are located;
- configuring the static analyzer so that the analysis parameters match the options given to the compilers (several options do affect the program semantics).

All this is automatic and supports build processes that involve the automatic generation of source files that depend on the configuration, without requiring the development and maintenance of a separated analysis procedure: with ECLAIR the existing build procedure can be used verbatim.

One of the key properties of ECLAIR is that it understands all the analysis-relevant options of the supported compilers. The language used to abstractly model such options is so powerful that adding support for a new compiler is no longer a problem.

3 Web-Based Configuration Interface

All the verification tasks supported by ECLAIR can be specified and refined incrementally by means of a very convenient web-based configuration interface. From any web browser this allows, for instance: finding coding rules using a powerful tag-based selection logic; activating and customizing coding rules, possibly restricting their use to only part of the project; selecting and customizing the kind of reports and compliance matrices to be generated; fine-tuning the semantic treatment of each elementary operation (particularly for undefined- and implementation-defined behaviors) in model-checking and automatic test data generation; choosing to run the verification task immediately or save the task for later.
4 Precise Parsing of Source Files

ECLAIR includes a state-of-the-art parser for C and C++ languages. In particular, for:

- the pre-standardization K&R C dialect;
- the standardized C languages (C90, C99 and C11);
- the standardized C++ languages (C++98, C++03 and C++11);
- the language extensions of the GNU C/C++ dialects;
- the language extensions of the Microsoft C/C++ dialects.

Work is in progress to support other extensions (e.g., OpenCL and CUDA) as well as the Java language.

The parser produces an accurate abstract syntax tree (AST) representing all of the information available in the analyzed code. Accuracy means, among other things, that:

- the AST represents not only the explicit, but also all of the implicit language constructs in the source code: implicit type conversions, compiler-generated functions (e.g., not explicitly defined constructors), instantiations of function/class templates, etc.;
- all language constructs are provided with precise location information, enabling the generation of reports pointing to the exact source of a violation.

Source location information is not only precise, but also thorough:

- complete information is available for the chain of file inclusions and, orthogonally, for the chain of macro expansions that brought a lexical token to become part of the analyzed code;
- the well-known difficulty of tracking problems in implicit template instantiations is solved presenting to the user complete yet easily readable information about the full instantiation chain.

5 Supported Platforms and Development Environments

ECLAIR is available on most modern flavors of UNIX®, Linux, OS X® and Windows®, including Cygwin and MinGW, and can be used with just about any development environment. Thanks to its ability to intercept the toolchain components, it supports all sorts of makefile-based, script-based or hybrid build systems. ECLAIR can leverage the availability of computing resources by supporting parallel and distributed program analyses. Most popular C/C++ compilers and cross compilers are supported, including CodeWarrior™, GCC, Green Hills®, IAR™, Intel®, Keil Software®, Microsoft®, MPLAB®, Renesas Electronics, Texas Instruments™, clang/LLVM.

For More Information

BUGSENG srl
Parco Area delle Scienze 53/A
I-43124 Parma, Italy
Tel: +39 0521 906 906
Fax: +39 0521 906 950
Email: info@bugseng.com
Web: http://bugseng.com

no shortcuts, no compromises, no excuses: software verification done right