



## Current Features

The main features of the version of **eclair** exhibited at FM'2009 are summarized in the following paragraphs.

### Toolchain Emulation

A powerful and completely generic option-processing mechanism allows **eclair** to masquerade as any toolchain component, such as `gcc`, with its 1000+ options, and any other compiler, assembler and linker.

### Unchanged Build System

Thanks to this kind of mimicry, **eclair** can access all the code comprising an application or library without requiring *any* change in the build system. In particular, if the application/library can be built in parallel (e.g., via `make -j`), then it can be analyzed with **eclair** in parallel. Here, for instance, is how `httpd` version 2.2.11 can be analyzed once the verification tasks to be performed have been specified in the file `my_setup`:

```
$ tar jxf httpd-2.2.11.tar.bz2
$ cd httpd-2.2.11
$ ./configure
$ eclair_env \
  -eval-file=my_setup \
  -- make -j 6
```

### Double AST Interface

The **eclair** parser builds an abstract syntax tree (AST) that is available to the other components of the system via two different, though completely interoperable, interfaces:

- As a Prolog term that directly encodes all and only the syntactic aspects of the program; but incorporating “handles” as subterms that give easy access to all non-syntactic information.
- As a number of inter-related C++ classes: a more complex representation that, however, gives access to all the information.

The Prolog view makes it easy to code syntax-based algorithms, such as coding rule checkers, program analyzers and transformers.

### Precise and Fast Parsing

The AST built by the parser encodes precise information about the original source code so that we can recover, almost for each token, complete information on the chain of file inclusions and, orthogonally, the chain of macro expansions that brought it to the preprocessed source. This means that we can build tools able to identify the precise points in the source that are responsible for

a given state of affairs. The parser, which supports ISO C90 and C99 along with several GNU and Microsoft extensions, is pretty fast: today running at more than 100,000 lines of code per second, with room for further optimizations.

### Simplified Intermediate Form

**eclair** includes a semantics-preserving program transformer that can simplify a program into a subset of C, the main simplification consisting of the removal of side-effects from subexpressions. The simplified form, which is of course compilable and semantically equivalent, makes it much easier to develop semantics-based program manipulations.

## Features Under Development



### Coding Rule Checkers and Bug Finders

The coding rule checkers will support the following rule sets: MISRA-C:2004, CERT C Secure Coding Standard, JSF C++, High-Integrity C++.

The simplicity and elegance with which checkers can be defined is such that we plan to have full support for such rule sets (and another one we are developing ourselves) in a short time. The same simplicity is available to users that wish to develop their own checkers.

### Precise and Scalable Program Analysis

We are porting the semantic analysis engine of a previous prototype to **eclair**. This includes a flow-, context- and field-sensitive pointer analysis along with numerical value analyses based on the Parma Polyhedra Library, a state-of-the-art library of numerical abstractions. A key feature of **eclair** will be the reduction of false positives thanks to sophisticated program analyses and *not* via the introduction of false negatives. Nonetheless, we are investing a lot of effort into the development of a fully configurable system: no users will have to pay for costly analyses that they do not want; while users such as those working in safety-critical domains will be able to *automatically prove*

that their software is completely immune from certain defects, such as unexpected run-time exceptions.

### Extension to C++

Work on extending **eclair** to C++ has already started. We will first complete support for the fragment of C++ that is more relevant for the development of safety-critical systems (no templates, no STL, no exceptions, limited inheritance, pre-allocation of memory and RTOS resources and so on). Support for the full C++ language will be incorporated at a later stage.

### Browser and Cross Reference Tool

An xhtml-based source code browser and cross reference tool will graphically present the analysis and verification results marking, on the source code, culprits and evidence along with supplementary information needed by programmers and security engineers when resolving the problems. In addition, the browser will allow the user to quickly visit:

- the declaration of an entity starting from any reference to that entity;
- all uses of a certain entity, for a number of different classes of use (assignment, dereference, ...), starting from the declaration of that entity;

- a macro definition starting from any of its instantiations;
- all the instantiations of a macro starting from its definition;

The browser will present tooltips concerning the type of expressions and other information coming from semantic analysis. For the semantic checkers, it will show the execution path that leads to the violations.

### Proving Complex Program Properties

We will port to **eclair** a termination analyzer we developed in a previous prototype. In addition, we will implement a cost analyzer that will be able to automatically obtain upper bounds on the computational complexity of a program, its heap and stack consumption, as well as other cost measures definable by the user. We will also include a module for automatically proving the absence of buffer overflows in programs with that property. The system will also provide support for the automatic proof of user definable assertions. When these proofs cannot be completed, the system will point out the source of the difficulty so that programmers can either fix the program or add sufficient assertions to allow the completion of the proof.

\* With ongoing extension to C++.

its development status has now reached the stage where validation in industrial contexts is needed.

its development joins scrupulous research on the theoretical foundations of program analysis and verification to a total adherence to the best available practices in software development.

is a new, professional platform for the verification of the C family of languages.

Applied Formal Methods Laboratory  
University of Parma, Italy

An Industrial-Strength Platform  
for the Verification of C Programs\*



# eclair

# eclair

## For More Information

For all inquiries, please write to  
**[bagnara@cs.unipr.it](mailto:bagnara@cs.unipr.it)**

At this stage of the development of **eclair** we are more than willing to take up challenges coming from both the industrial and the academic worlds.

We are willing to sign all the required NDAs and can even provide a name obfuscator to further protect your code.

We seek partnerships (including, but not limited to, the participation in European projects) that will enable us to finalize the design and implementation of **eclair** so as to make it as powerful as possible.