Cominlabs Report

Samuel Risbourg and Alan Schmitt

March 10, 2020

1 Introduction

JavaScript is the language of the web. It is used in every browser to interpret dynamic pages.

While the first implementation of JavaScript interpreter has been available in 1996, **Netscape** started to work with the **European association for standardizing information and communication systems** (ECMA) to standardize the language. The first version of the **ECMAScript** standard, **ECMA-262**, was adopted in 1997. Despite its early specification and its mainstream adoption by web developers JavaScript has a complex syntax and semantics. Those can lead to ambiguity or misunderstanding in source code and finally in security or safety issues in produced software. Targeting these JavaScript engineering concerns is the purpose of the **JSExplain** tool.

The story of JSExplain starts with an earlier tool. As JavaScript is ubiquitous and very precisely specified, it is amenable to mechanized formalization. In the **JSRef** project, the ECMAScript specification was translated into **Gallina**, the **Coq** proof assistant language, to formally specify the semantics of JavaScript

From there it was possible to extract an OCaml JavaScript interpreter. Assuming that no error were done during the translation, we can affirm that this interpreter is correct regarding the official specification of the language. It is not optimized as the ones in usual web browsers, but it is very close to the specification. It can thus be used as a reference interpreter to check the expected behavior of the language.

Such a naive interpreter can be useful for developers implementing real world interpreters. Even if these developers follow rigorously the specification, there is still room for interpretation. This can result in different interpreters having different behaviors, as ambiguities may remain in the specification, due to the way it is written in natural language. This is a

common and traditional issue in software engineering. Comparing the results of interpreters is not sufficient, however. One needs to understand the reasons behind the correct result. **JSExplain** is such a tool: it helps to explain how the specification defines the evaluation of JavaScript to developers of optimized JavaScript interpreters.

Of course anyone who have any interest in the deep understanding of JavaScript semantics and behavior, such as web application developers, can also use this tool. To help spread its usage by the JavaScript community, the tool is actually embedded in a web page.

2 JSExplain

Inria Celtique's team main research domains are programming language from a theoretical point of view: semantics, compilation, safety and security of various languages are studied in the perspective of producing proof of soundness and certification guarantee on studied language.

The Coq Proof Assistant and OCaml programming language are used to target these objectives. During the development of JSExplain the decision to have a convivial web interface developed in JavaScript was made to target the JavaScript developer community [1]

Indeed, further than a simple research prototype, the **JSExplain** team targets an adoption of the tool by the **ECMAScript** consortium so that it help for in debugging and evolution of **JavaScript** language.

2.1 Software Architecture and build

Starting from the OCaml source code of the **JSRef** interpreter the tool is build as follows (1):

- First the **JSRef** is converted in JavaScript so that it can be embedded in the web interface
- The User Interface parses the JavaScript source code submitted. This code is parsed using the **Esprima** JavaScript library. The parser returns an Abstract Syntax Tree, a common data structure used to represent the parsed program.
- Then the core of the tool generates a trace of the symbolic execution of the program. The interface is used to browse this execution step by

[1] Arthur Charguéraud, Alan Schmitt, and Thomas Wood. **JSExplain: A Double Debugger for JavaScript**. In *The Web Conference*, 2018

step visualizing in parallel the code and state of the program, as well as the code and state of the interpreter.

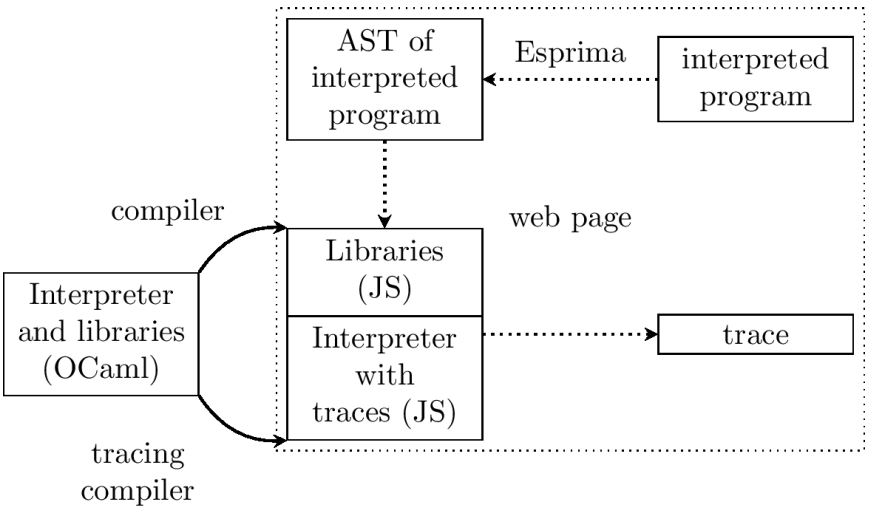

Figure 1: JSExplain Architecture

3 Evolution

JSExplain is a join work of Martin Bodin, Alan Schmitt, Arthur Charguéraud, and Thomas Wood. During the last year some development was done by Samuel Risbourg to improve the user interface and add continuous integration.

3.1 Migration

JSExplain repository was hosted on Github and some of the continuous integration servers were at Imperial College. One of the task was to migrate the repository and continuous integration to Inria Gitlab. The project is now hosted on `https://gitlab.inria.fr/star-explain/` and the Continuous Integration takes place on `https://ci.inria.fr/`.

3.2 Integration

Another focus was to give a better accessibility to the tool so that it can be easily tested by the JavaScript developer community. To target this goal

some cleaning has been done in the installation procedure (Makefile) and new functionality were added for a better deployment mechanism and system compatibility. For example, instead of a static installation, the original Makefile was migrated to a GNU Autotools generic one. This way some installation prerequisites could be checked on the targeted operating system at installation time.

3.2.1 OPAM

In the same perspective of easy delivery, the tool is now integrated as a package in OPAM, the OCaml Package Manager. The reason why the project is present in the OCaml ecosystem and not the JavaScript one is that the build toolchain is an OCaml one, the JavaScript core of the web site comes from a compilation from a JavaScript interpreter written in OCaml.

This OPAM package installation procedure now give an synthetic way to get the tool. It is available in the README of the repository. The main gain of this achievement is that the final user does not have to take care of many software dependencies. In the future this could become even easier. Each official software release could be integrated in OPAM. Thus, if the user has already OPAM installed, the installation could be done in one instruction.

3.2.2 Site hosting

Another facility is given to the tool developer to deploy an online version through Makefile command line. The site is hosted there `http://jsexplain.gforge.inria.fr/`. This could be useful during presentation to show new functionality or to fix a bug fast.

3.2.3 The Inria Forge

Both the OPAM packages and the web site are hosted on the Inria Forge. At this point the generation and the deployment of both services on the platform is managed through the Makefile. This could be an entry point in further continuous integration testing. We also plan to migrate hosting to Inria Gitlab, as the Inria Forge will be deprecated.

3.3 Development

The figure 2 is a screenshot of the final Web User Interface.

JSExplain is a JavaScript program debugger that can be used in a browser. The interface is divided in a classical way for such a tool.

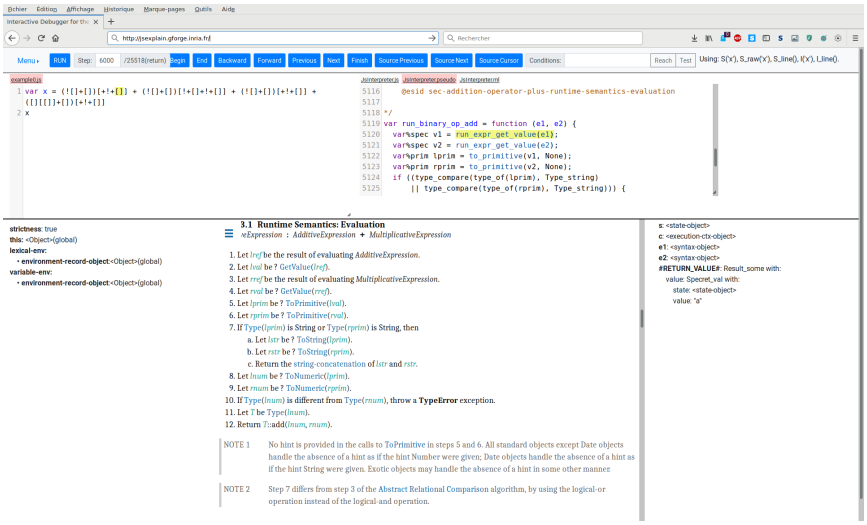

Figure 2: JSExplain User Interface

- You can write or load the program you want to verify, and then interpret and run it step by step
- We visualize at the same time:
  - The program execution context: memory state, expressions, variables, types and values
  - The lexical context
- At each program step we see in parallel:
  - The JavaScript interpreter code and its localization regarding the precise execution point .
  - What part of the ECMAScript specification it corresponds to.
- Buttons in the menu bar allow the user to go forward and backward in the program execution and to skip functions that he does not want to inspect

The main evolution during the last year in software architecture concern both the core of the tool as well as its front-end.

3.3.1 Core

- Targeting future development and maintenance of the tool, an intermediate Abstract Syntax Tree has been added been added in the OCaml to JavaScript Compiler (`fjs_of_fml`). Previously the compiler was just a JavaScript printer based on the OCaml AST. Now there is a JavaScript AST and a translation from the OCaml AST to the JavaScript one. This could help to manage new language feature and to understand some compilation error.

- Another task was to simplify the interpreter to get code closer to the ECMA specification. As a consequence, the parallel between pseudo formal ECMA specification and the JavaScript implementation of the embedded interpreter becomes more apparent.

3.3.2 Front-end

- A first task was to add a frame to the ECMA website `https://tc39.es/` JavaScript Specification in the User Interface and to synchronize it with the code of the interpreter. Thus, while the studied source code is interpreted through the embedded interpreter, the user can see the step by step execution of the source code, the step by step execution of the interpreter code and the corresponding ECMAScript specification. Thus, the user can understand a JavaScript program or the right behavior expected by the ECMA consortium.

- Some cosmetic and ergonomic change has been done to the user interface to improve user experience. A modern web framework has been added to manage the different sub-windows in the interface.

- Some bug fixes have been done in the trace navigation functionality.

3.4 Tests

Another effort was made on benchmarking the tool to avoid any regression in it while adding any new feature in it. Since the language semantics changes often, the tool has to be updated frequently to stay synchronous with it. As a test suite is available and maintained by the ECMA Script team (`https://github.com/tc39/test262`) the tool is tested with it. Thus, we can compare two different states (git commit) of the tool. A script test both of them an check that the interpreter stays stable.

4 Conclusion

During this year the development effort focused on usability. Integration and deployment has been improved, as well as the User Experience trough a new Interface.

On a more technical side, the tool is now tested using the official ECMAScript test suite. We automated the usage of this test suite to avoid any regressions during the software evolution.