JSExplain: A Double Debugger for JavaScript

Celtique Team

Alan Schmitt

Arthur Charguéraud

Thomas Wood

The Language of the Web

JavaScript is THE language of the web. It is used in all browsers to interpret dynamic website pages.

Outside its original ecosystem, some operating system native software is written with JavaScript (like with Electron framework).

But it is also a language with a complex syntax and many implicit type castings. These issues make some programs difficult to understand. As a consequence, evaluating JavaScript software coherence, safety or security could be a hard task. It creates a drag for language evolution because evaluating new feature proposal impact is difficult.

A Tool to Understand JavaScript

JSExplain is a JavaScript program debugger that can be used in a browser. You can write or load the program you want to verify, and then interpret and run it step by step (❶).

We visualize at the same time:

The program execution context: memory state, expressions, variables, types and values (❷).

The lexical context (❸).

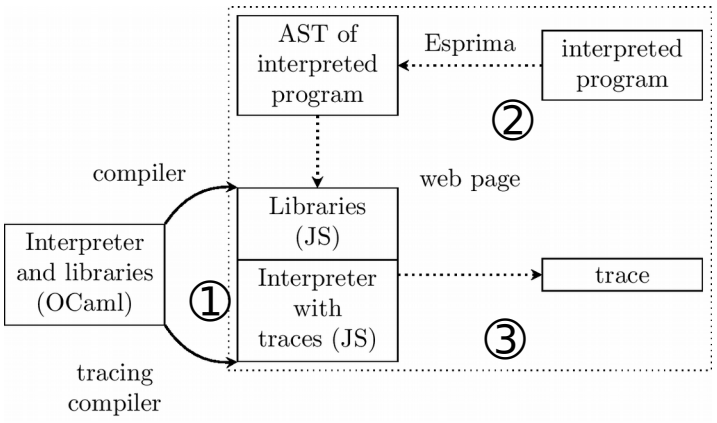
At each program step we see in parallel:

The JavaScript interpreter code and its localization regarding the precise execution point (❹).

What part of the ECMAScript specification it corresponds to (❺).

Buttons allow the user to go forward and backward in the program execution and to skip functions that he does not want to inspect (❻).

Architecture



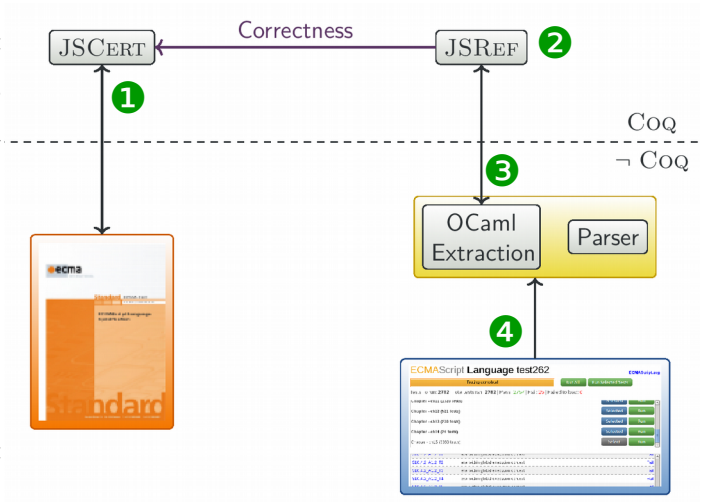
The JavaScript interpreter is written in Ocaml. It is derived from the interpreter extracted from the JSCert project that was written in Coq. It is compiled into JavaScript so that it can be embeded in the web interface (①). The source code is converted in a specific data structure called Abstract Syntax Tree (②). The interpreter takes it as input and produces the program execution trace (③).

A Certified JavaScript Interpreter

Web browser interpreters implement ECMAScript, the official language specification in different ways, mainly because it is written in pseudo natural language.

During the JSCert project, it has been formalized and verified with the Coq proof assistant (❶).

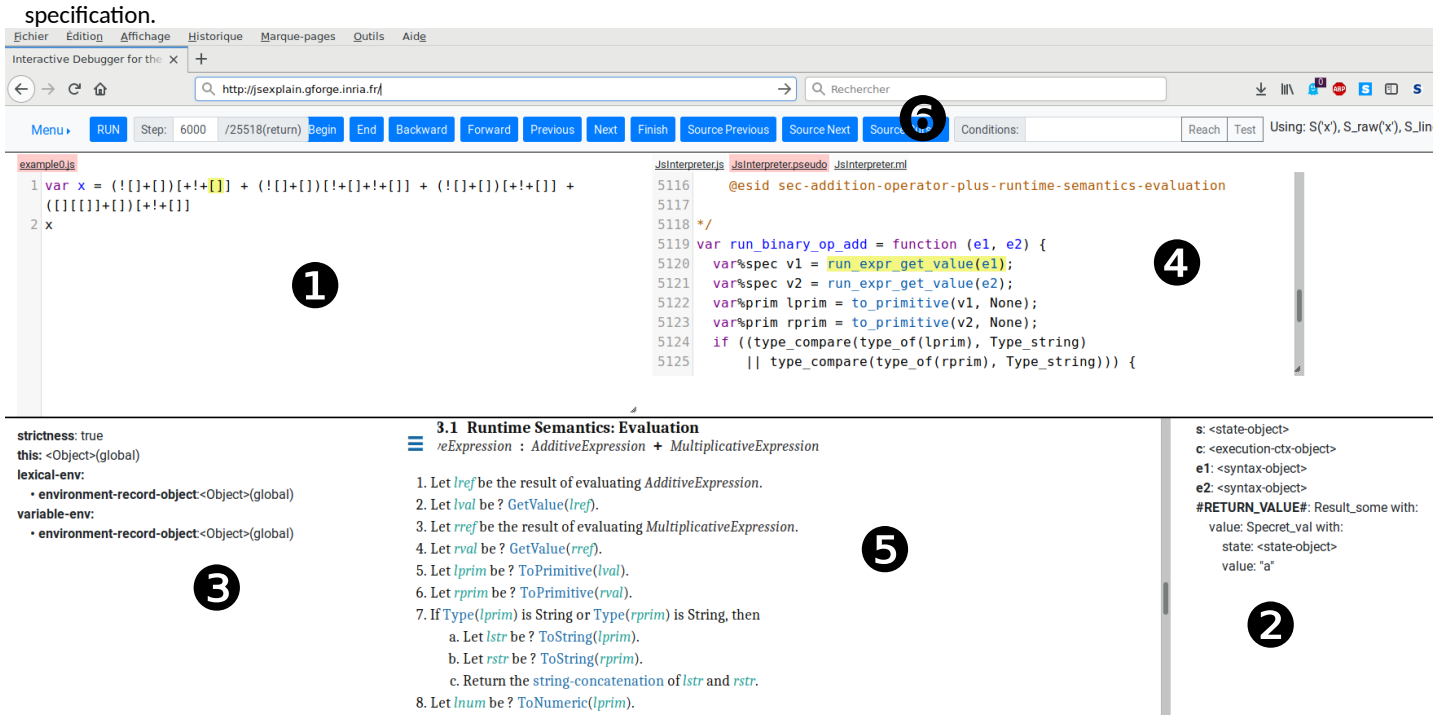
The JSRef interpreter is correct regarding this specification (❷).



Its Ocaml extraction maintains this correctness too (❸). The semantics of the language is preserved from end to end. We have the guarantee that the executable interpreter is an exact mechanizable version of the semantics expressed in the specification.

To verify this assertion and maintain the interpreter, it is tested with the official ECMA test set (❹).

Finally, from the study of a langage and the formalization of its specification we got a certified JavaScript reference interpreter. People can use it to verify their own developments.



Normalization

ECMA, the European association for standardizing information and communication systems was created in 1959. When JavaScript is born in 1995 ECMA has started to publish the language specification since 1997. So beside its issues, at least the language has been formalized since its early beginning.

The process to suggest a new feature, to study and validate it is strongly normalized. Once it is done it can be integrated in the specification.

Help JavaScript Evolution

If the JSRef JavaScript Interpreter is a naive one, it is correct and give an exact idea of what should be the behavior of a real world interpreter. So beyond JavaScript developpers it is a precious tool for normalization people.

Upstream and during the integration of a new feature in the language one would like to know the impact of the evolution. Does it break or contradict any existing feature? A version of the feature can be implemented in the JSRef interpreter and checked with the official test set.