Deliverable WP1: Final report on the mechanization of the full JavaScript language

Project Ajacs

June 2016

The technical content of this deliverable is the JSExplain platform, found at `https://github.com/jscert/jsexplain`. What follows is a motivation for this approach and an overview of the platform.

1 Motivation for JSExplain

Our previous work on formalizing JavaScript, called JSCert [1] consisted of two formalization in Coq [2] of the semantics. First, we provided a descriptive formalization, close to the specification and useful to prove programs. Second, we also provided an executable specification, to run the JavaScript test suite. Finally, we proved that these two formalizations captured the same semantics.

This principled approach is very robust, but requires a lot of investment since adding any feature amounts to formalizing it twice and proving both formalizations match. To cover the full semantics of recent JavaScript, we need to extend from ES5 (whose specification is a 252 pages document) to ES2016 (586 pages). The amount of work is staggering. In addition, only people fluent in Coq can participate to this formalization. These are the reasons why we switched to a different approach, where the amount of work required is lessened, and where it is no necessary to know Coq to contribute.

2 Description of JSExplain

JSExplain is an implementation of a JavaScript interpreter in a subset of OCaml. We started from the extraction to OCaml of our interpreter written in Coq, which we then manually edited to make it easier to read. For instance we rely on monadic operators to simplify error handling.

The code `let%run (s,r) = e in cont` is evaluated as follows:

1. execute `e`;
2. make sure there is no internal problem (is so, propagate it);
3. check the result type, if it's not `normal`, then propagate it;
4. bind the state and value to `s` and `r` and evaluate `cont`.

The subset of OCaml we rely on is pure, there are no side effects, with the following features:

- types: bool, string, int, float;
- algebraic datatypes, tuples, records;
- simple pattern matching;
- no partial application;
- custom notation for monadic operators (ppx).

We can then leverage this implementation to obtain several tools:

- a compilation in a subset of JavaScript, allowing standardization people to easily experiment with the language (done);
- an instrumentation of the implementation, to execute JavaScript programs step by step showing both the state of the program and the state of the interpreter, allowing an inspection of how the spec executes (done, a demonstration is available[1]);
- a compilation to Coq to recover a descriptive formal semantics (to be done);
- an extraction of a textual description of the semantics from the OCaml implementation (to be done).

This new approach is still close to the specification and can be executed to run the test suite, both the OCaml version and the JavaScript version. We are currently extending it to cover recent features of JavaScript, in particular proxies. Our long-term goal is to make the code easy to write for the participants of the specification committee, so that they directly provide the formalization.

[1] `https://jscert.github.io/jsexplain/branch/master/driver.html`

References

[1] Martin Bodin et al. "A Trusted Mechanised JavaScript Specification". In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014)*. San Diego, CA, USA, Jan. 2014, pp. 87–100.

[2] The Coq development team. *The Coq proof assistant reference manual*. Version 8.4. 2011. URL: `http://coq.inria.fr`.