An Infrastructure for Tractable Verification of JavaScript

Program Specification and Verification Group

Imperial College London


Philippa Gardner

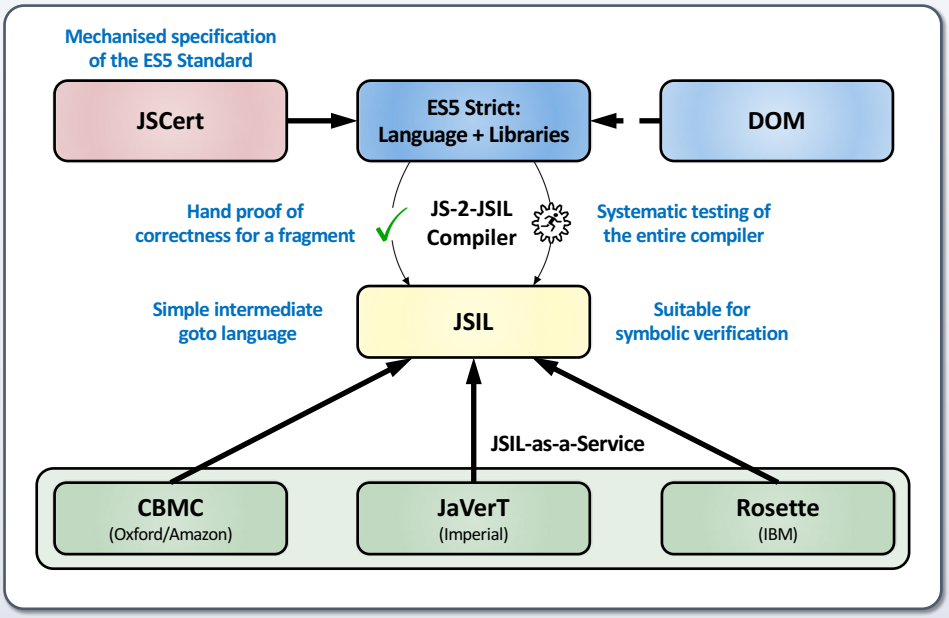

José Fragoso Santos


Petar Maksimović


Daiva Naudžiūnienė


Azalea Raad


Thomas Wood



The dynamic nature of JavaScript and its complex semantics make it a difficult target for verification. To address this issue, we develop JS-2-JSIL, a compiler from JavaScript (ECMAScript 5 strict) to a simple intermediate goto language, JSIL. We design JS-2-JSIL to be step-by-step faithful to the ECMAScript standard, systematically testing it against the official ECMAScript conformance test suite, and proving it correct with respect to a fragment of the ES5 Strict operational semantics. We develop JSIL logic, a separation logic for JSIL that we prove sound with respect to its operational semantics, and a semi-automatic verification tool based on this logic. Together, these results allow us to verify Hoare triples for JavaScript using JSIL logic and JS-2-JSIL.

We establish the infrastructure required for verifying JavaScript code. We focus on the internal functions underlying JavaScript, which provide essential functionalities and are used in the semantics of all JavaScript commands. We implement all of these functions in JSIL, give functionally correct specifications for most of them, and verify, using our tool, that their JSIL implementations are correct with respect to these specifications. We are able to translate JavaScript triples to JSIL triples and verify the JSIL triples using our tool.

JSIL

Goto language: extensible objects, dynamic field access and function calls

$m \in Str \quad n \in \mathcal{N}um \quad b \in Bool \quad l \in \mathcal{L}$

$\mathtt{x} \in \mathcal{X}_{\text{JSIL}} \quad \lambda \in \mathcal{L}it \ ::= \ n \mid b \mid m \mid \text{undefined} \mid \text{null}$

$\mathtt{t} \in \text{Types} ::= \mathtt{Num} \mid \mathtt{Bool} \mid \mathtt{Str} \mid \mathtt{Undef} \mid \mathtt{Null} \mid \mathtt{Empty} \mid \mathtt{Obj} \mid \mathtt{List} \mid \mathtt{Type}$

$\mathtt{v} \in \mathcal{V}_{\text{JSIL}} \ ::= \lambda \mid l \mid \text{empty} \mid \text{error} \mid \mathtt{t} \mid \overline{\mathtt{v}}$

$\mathtt{e} \in \mathcal{E}_{\text{JSIL}} \ ::= \mathtt{v} \mid \mathtt{x} \mid \ominus\, \mathtt{e} \mid \mathtt{e} \oplus \mathtt{e} \mid \text{typeof}\,(\mathtt{e}) \mid \overline{\mathtt{e}} \mid \text{nth}\,(\mathtt{e}, \mathtt{e})$

$\mathtt{bc} \in \text{BCmd} ::= \text{skip} \mid \mathtt{x} := \mathtt{e} \mid \mathtt{x} := \text{new}\,() \mid \mathtt{x} := [\mathtt{e}, \mathtt{e}] \mid [\mathtt{e}_1, \mathtt{e}_2] := \mathtt{e}_3 \mid \text{delete}\,(\mathtt{e}, \mathtt{e}) \mid \mathtt{x} := \text{hasField}\,(\mathtt{e}, \mathtt{e}) \mid \mathtt{x} := \text{getFields}\,(\mathtt{e})$

$\mathtt{c} \in \text{Cmd} ::= \mathtt{bc} \mid \text{goto}\ i \mid \text{goto}\ [\mathtt{e}]\ i,\ j \mid \mathtt{x} := \mathtt{e}(\overline{\mathtt{e}})\ \text{with}\ j \mid \mathtt{x} := \phi(\overline{\mathtt{x}})$

Built-in ϕ-nodes provide support for SSA

We have implemented all JavaScript internal functions and most of the built-in libraries directly in JSIL (~3Kloc)

Testing

Systematic testing of the full JS-2-JSIL compiler against the official ECMAScript Test262 test suite

ES6 version of the test suite due to major deficiencies in the ES5 version

Rigorous filtering and analysis

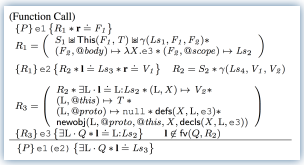
Continuous-integration infrastructure, tests running en masse in parallel

ES5 Strict Tests	10469
Tests for non-impl. features	1297
Compiler Coverage	**9160**
ES5-ES6 differences	345
Tests using non-impl. features	30
Applicable Tests	**8797**
Passed tests	**8797**
Failed tests	0

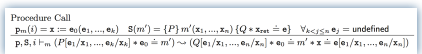
Separation-logic-based reasoning

JS Logic (POPL '12)
- ✓ Soundness proof
- ! Complex proof rules
- ! Difficult to automate

JSIL Logic (now)
- ✓ Soundness proof
- ✓ Simple proof rules
- ✓ Can be automated

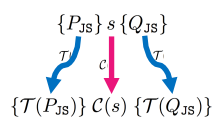


The complexity of JavaScript has moved to the translation!



Translating assertions from JS logic to JSIL logic

Non-trivial cases:
- Variable resolution
- Property descriptors
- Function objects
- The this object



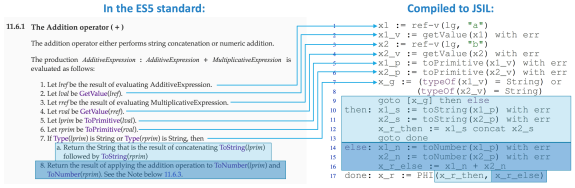
Correctness of the translation for assertions

$$H_{\text{JS}}, L, \epsilon_{\text{JS}} \models P_{\text{JS}} \ \Leftrightarrow\ H_{\text{JSIL}}, \rho, \mathcal{T}(\epsilon_{\text{JS}}) \models \mathcal{T}(P_{\text{JS}}) \ \text{when}\ H_{\text{JS}}, L \sim_{\beta} H_{\text{JSIL}}, \rho$$

JS-2-JSIL

Compilation follows English standard line-by-line; correctness shown via hand proof and testing

Correctness allows JSIL analysis to lift to JavaScript analysis



JaVerT

Semi-automatic verification tool for JSIL

Based on symbolic execution and JSIL Logic

User specifies pre- and post-conditions, loop invariants, and predicate manipulations in JS Logic; all automatically translated to JSIL Logic

Current state: JaVerT can verify a substantial fragment of JavaScript's internal functions and very simple JavaScript programs

JSIL-as-a-Service

JSIL front-end to enable verification tools to analyse JavaScript

CBMC, with Kroening and Tautschnig at Amazon

Rosette, with Julian Dolby at IBM