JavaScript Analysis Framework

ERC Workshop (25 July to 28 July 2012)

Sukyoung Ryu Junhee Cho Junho Jin Soocheol Won Hongki Lee

JavaScript

Client-side script language on web browsers
Dynamic reactions to user inputs
Used not only on web browsers but also in smart appliances

Motivation

Existing Research
Based on **ECMAScript 3** (old version)
Missing constructs in different frameworks
Undocumented, **unstructured** frameworks

Any Solution? → JSAF!

Based on **ECMAScript 5** (the **LATEST** version)
Formal description of the **entire** ECMAScript 5
Pluggable framework (to enable various future work)

JSAF

ecma INTERNATIONAL

12.6.2 The `while` Statement

The production *IterationStatement* : **`while`** **`(`** *Expression* **`)`** *Statement* is evaluated as follows:

1. Let V = empty.
2. Repeat
   a. Let *exprRef* be the result of evaluating *Expression*.
   b. If ToBoolean(GetValue(*exprRef*)) is **false**, return (normal, V, empty).
   c. Let *stmt* be the result of evaluating *Statement*.
   d. If *stmt*.value is not empty, let V = *stmt*.value.
   e. If *stmt*.type is not continue || *stmt*.target is not in the current label set, then
      i. If *stmt*.type is break and *stmt*.target is in the current label set, then
         1. Return (normal, V, empty).
      ii. If *stmt* is an abrupt completion, return *stmt*.

Hoister — Lift the declarations of variables and functions up to the beginning of a program or a function

Disambiguator — Check some static restrictions and rename identifiers to unique names

withRewriter — Rename 'with' statements into other constructs with the same semantics

Interpreter — Evaluate IR according to its formal specification

12.6.2 The `while` Statement

$$\frac{(H, A, tb), \underline{x} \rightarrow_e err}{(H, A, tb), \texttt{while}\ (\underline{x})\ \underline{s} \rightarrow_s (H, A), \texttt{Throw}(err)}$$

$$\frac{(H, A, tb), \underline{x} \rightarrow_e v \qquad ToBoolean(v) = \texttt{false}}{(H, A, tb), \texttt{while}\ (\underline{x})\ \underline{s} \rightarrow_s (H, A), \texttt{Normal(empty)}}$$

Result

JavaScript

```
var x;
while(true) {
    x = 1;
}
x = 2;
```

Parser

Generated by Rats!

```
/ while w openparen w a1:Expression w closeparen
  w a2:Statement
  { Span span = createSpan(yyStart,yyCount);
    yyValue = NodeFactory.makeWhile(span, a1, a2);
  }
```

AST (Abstract Syntax Tree)

Using ASTGen

```
...
   (While
    _body=(Block
     _info=(SpanInfo)
     _stmts=[
      (ExprStmt
       _expr=(AssignOpApp
        _info=(SpanInfo)
...
```

AST2IR

Translate AST to IR according to its formal specification

$$= \langle \underline{\diamond break} : \{ \text{IRSeq}(\langle$$
$$ast2ir_e[\![e]\!](\Sigma, \Gamma)(\underline{\diamond new_1});$$
$$\underline{\diamond new_2} = \diamond\text{toBoolean}(\underline{\diamond new_1});$$
$$\text{while}\ (\underline{\diamond new_2})\ \text{IRSeq}(\langle$$
$$\underline{\diamond continue} : \{$$
$$\text{IRSeq}(\langle ast2ir_s[\![s]\!](\Sigma; \underline{\diamond break}; \underline{\diamond continue}, \Gamma)$$
$$\rangle) \}$$
$$ast2ir_e[\![e]\!](\Sigma, \Gamma)(\underline{\diamond new_1});$$
$$\underline{\diamond new_2} = \diamond\text{toBoolean}(\underline{\diamond new_1})$$
$$\rangle)$$
$$\rangle) \} \rangle$$

IR (Intermediate Representation)

IR Semantics

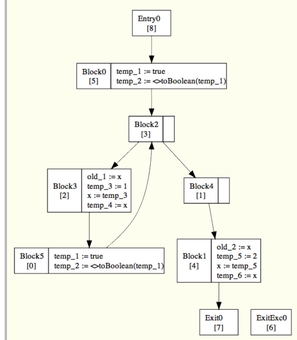
```
...
  while(<>_temp_3)
    {
      <>_temp_4 : {
        {
          <>_old_5 = x
          <>_temp_6 = 1
          x = <>_temp_6
          <>_temp_7 = x
        }
      }
...
```

IR2CFG

Build CFG from IR according to its formal specification

$$n_1 \stackrel{let}{=} GetTail(G, N)(fid)$$
$$n_{head} \stackrel{let}{=} G.NewBlock(fid)$$
$$n_2 \stackrel{let}{=} G.NewBlock(fid)$$
$$n_3 \stackrel{let}{=} G.NewBlock(fid)$$
$$G.AddEdge(n_1, n_{head})$$
$$G.AddEdge(n_{head}, n_2)$$
$$G.AddEdge(n_{head}, n_3)$$
$$(N_1, L_1) \stackrel{let}{=} [\![s]\!]_{stmt}(G, [n_2], L)(fid)$$
$$G.AddEdge(N_1, n_{head})$$
$$([n_3], L_1)$$

CFG (Control Flow Graph)



Future Work

Detect code clones in multiple JavaScript programs via AST comparisons → CloneDetector

Calculate code coverage of JavaScript programs via IR evaluation → CodeCoverage

Estimate static properties of JavaScript program via CFG traversal → Analyzer