JSFox: Integrating Static and Dynamic Type Analysis of JavaScript Programs

Tian Huat Tan[1], Yinxing Xue[2], Manman Chen[3], Shuang Liu[4], Yi Yu[5], Jun Sun[3]
[1]Acronis Singapore, [2]NTU Singapore, [3]SUTD Singapore, [4]SIT Singapore, [5]NII Japan

Motivation

- JavaScript is one of the **most used** programming languages.
- JavaScript is a **dynamic**, **weakly typed** language with many **flexible** features.

Type analysis is crucial for capturing representation errors

- Type information is the basis for program analysis methods
- Type information can serve as an abstraction for analysis methods

We proposed JSFox to infer types for JavaScript programs

Motivation Example

```
1  var y=0;
2  var x="a";
3  var i=1;
4  function f(a){
5    return a;
6  }
7  function chkFlag(flag){
8    var code=arguments[i];
9    if(flag==0){
10     return x*1;
11   }else if(flag==1){
12     return 1/y;
13   }else if(flag==2){
14     return -1/y;
15   }else{
16     return eval(code);
17   }
18 }
19 var res=chkFlag(3,"f('res')");
```

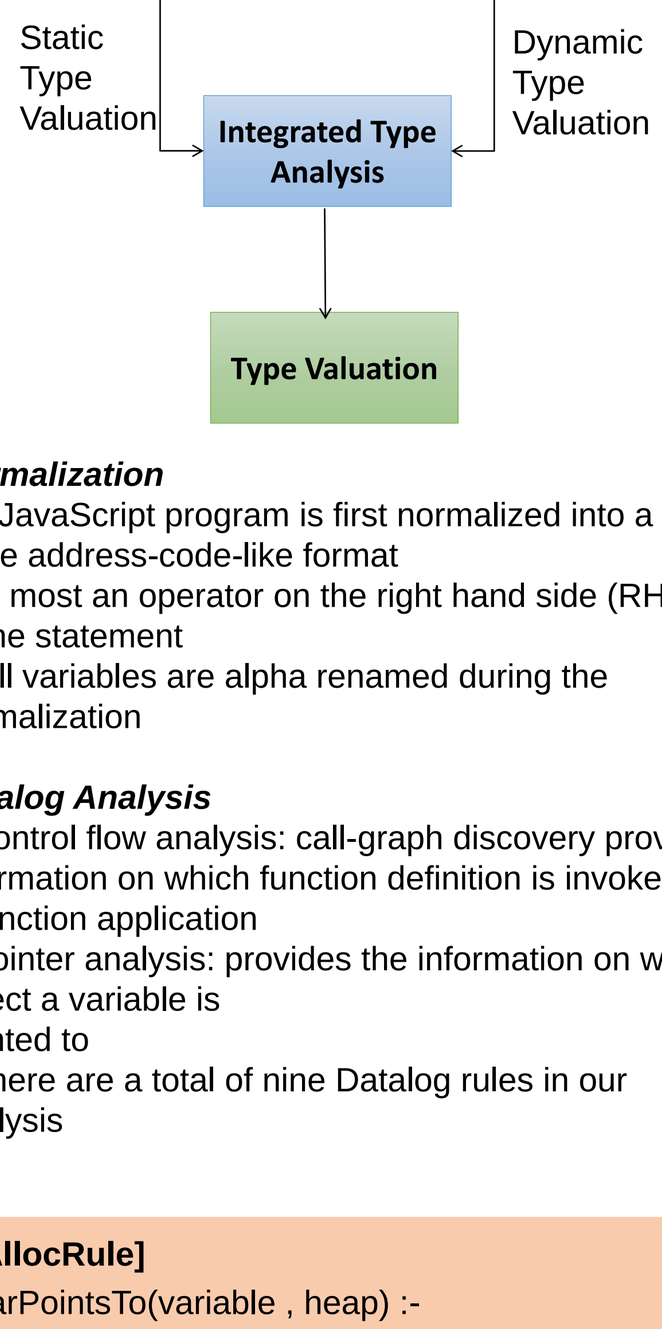
The program contains a **variadic** function, and the **eval** construct for evaluating dynamically loaded code.

These dynamic features contribute to the dynamism of JavaScript, and cannot be precisely modeled in static type analysis!

Dynamic type analysis is mostly incomplete, cannot cover all paths. Therefore, static type analysis is incorporated.

JSFox: make use of integrated static and dynamic type analysis

JSFox Architecture



Normalization

- A JavaScript program is first normalized into a three address-code-like format
- At most an operator on the right hand side (RHS) of the statement
- All variables are alpha renamed during the normalization

Datalog Analysis

- Control flow analysis: call-graph discovery provides information on which function definition is invoked by a function application
- Pointer analysis: provides the information on which object a variable is pointed to
- There are a total of nine Datalog rules in our analysis

[AllocRule]
VarPointsTo(variable , heap) :-
Reachable(methHeap), Alloc(variable , heap , methHeap)

***Instrumented Execution*:** Obtain the variable values and dynamic call graph edges

- For collection of variable values, we record the values of variables that have been assigned at a particular line
- For collection of dynamic call graph edges, we record the function definition that is invoked by a function application

Static Type Analysis

- field-sensitive, flow-sensitive, context-insensitive, and path-insensitive

Dynamic Type Analysis

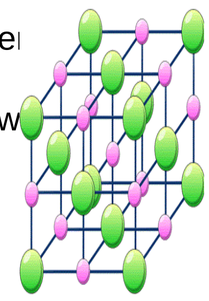
- The instrumented JavaScript program is executed during dynamic analysis.

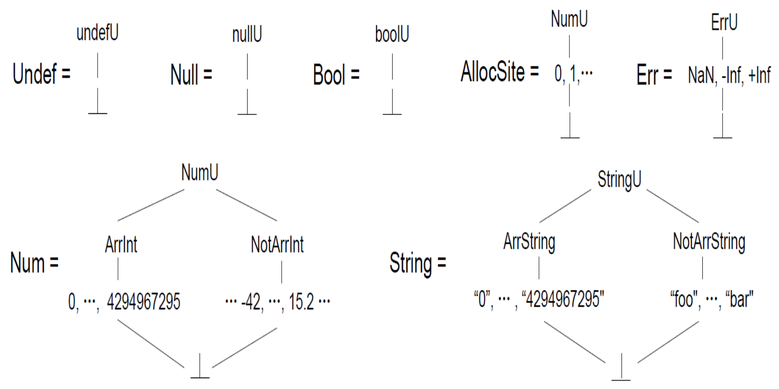
Integrated Type Analysis: use dynamic type analysis in refining static type analysis to make the analysis more complete and precise

- Dynamic call graph edges: capture the call graph edges that cannot be easily discovered by static analysis
- Dynamic type valuation: discover variable values that are obtained through dynamic features of JavaScript, e.g., runtime code evaluation.

Types in JavaScript

- We divide JavaScript types into several domains.
- A domain is defined using a lattice, which Represented using a Hasse diagram





Hasse Diagrams

Experimental Evaluation

We evaluate our tool on a popular JavaScript **benchmarks** collection
and real-world Web applications:

- JetStream: benchmarks from the SunSpider 1.0.2 and Octane 2
- Web Applications: Five real-world Web applications

Experimental Results

- We have discovered ***23*** type issues, and out of them ***8*** cases can ***only*** be detected by integrated type analysis
- The approach in [1] has identified ***12*** of them, which are all included in our *pure dynamic* type analysis

[1] M. Pradel, P. Schuh, and K. Sen. Typedevil: Dynamic type inconsistency analysis for javascript. In 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1, pages 314–324, 2015.