Automatic JavaScript Bug-Detecting Framework and Different Approaches to False-Positive Minimization

2013.12.19.
Master Workshop
Session 2. Program Analysis, Debugging

Junho Jin

Advisor	Ryu, Sukyoung
Dept.	Department of Computer Science
Lab.	Programming Language Research Group
Contact	junho.jin@kaist.ac.kr

What's JavaScript?

- **Simple scripting language** designed by Brendan Eich in 1995
- One of the **most popular languages** in modern industries

Problem Statement

Dynamic, yet often unintentional behaviors

- **Prototype-based inheritance** allows to dynamically change inheritance chains.
- **Null and Undefined** are different types, yet same in equality comparison.
- **Flexible array length** lets no index out of bounds exceptions occur.
- **The number of function arguments** is not restricted to its definition.
- etc.

✓ **Difficult to *debug* in JavaScript programs**

✓ **Difficult to *even define***

what JavaScript bugs are !

My Solution

Our Definition of JavaScript Bugs

Error

Any JavaScript semantics that causes ***critical exceptions***

Error	Definition
AbsentReadVariable	Program is trying to read a non-exisitent variable x.
BinaryOpSecondType	Right-hand side operand e of binary operator op is non-object (it must be an object).
CallNonConstructor	Program is calling non-constructor as if it's a constructor.
CallNonFunction	Program is calling non-function as if it's a constructor.
ObjectNullOrUndefined	Program is trying to access a property p of `null` or `undefined` value.
WrongThisType	Value of `this` is not of the expected type in built-in function f.

Warning

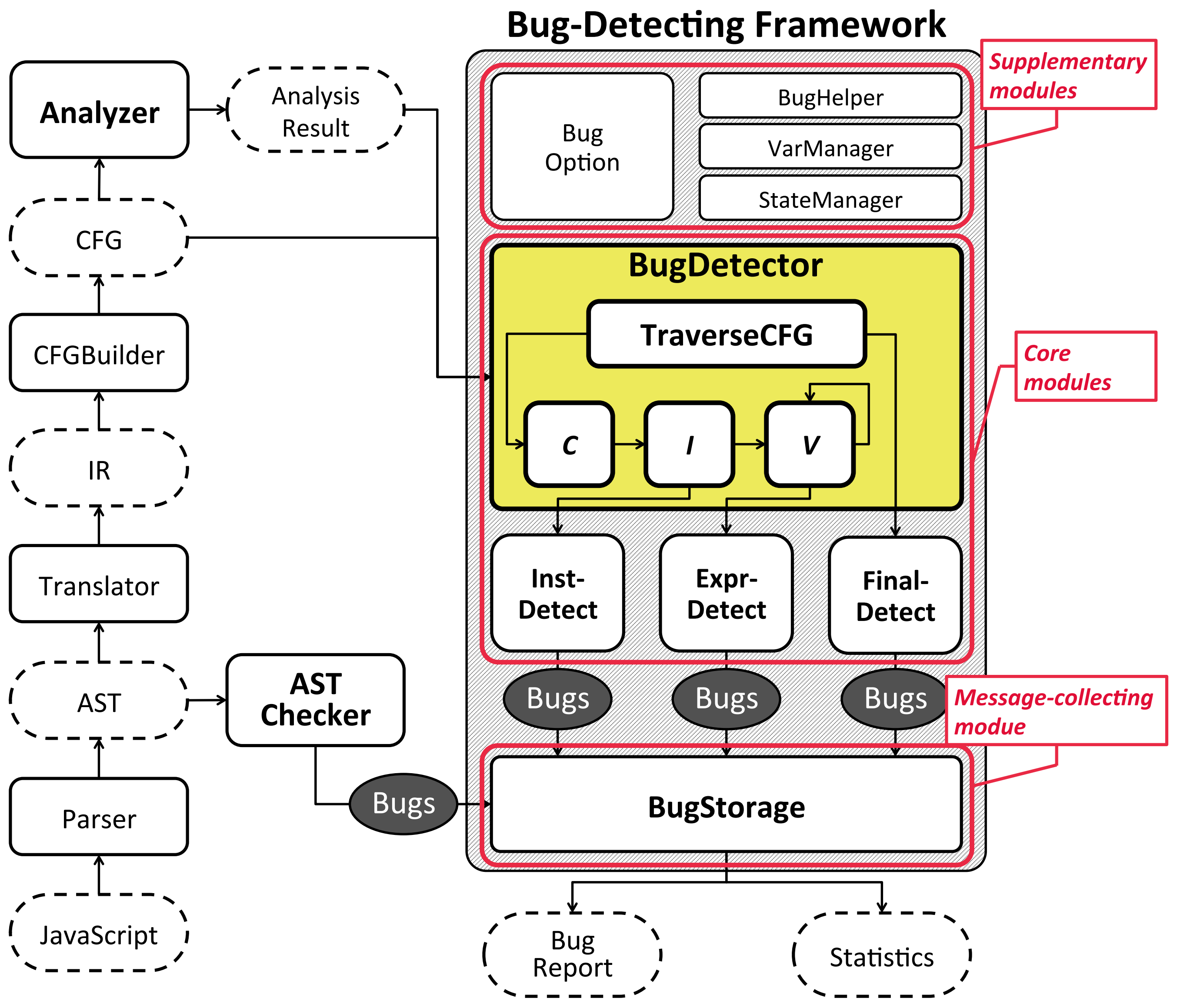
Any JavaScript semantics that does ***not*** causes critical exceptions,
yet causes ***unexpected behaviors***,
threatens the security of programs,
or ***hampers the optimization*** of programs

Warning	Definition
AbsentReadProperty	Program is trying to read a non-existent property p of an object.
BuiltinWrongArgType	Parameter x to a built-in function f is not of the expected type.
CallConstFunc	Function f is called as both a function and a constructor.
ConditionalBranch	Conditional expression e is always false (or always true).
ConvertUndefToNum	Program is trying to convert undefined to a number.
DefaultValue	Assigning a non-function value to toString or valueOf property may cause a TypeError.
FunctionArgSize	The number of parameters to a function f does not match to its declaration.
GlobalThis	`this` refers to the global object.
ImplicitTypeConversion	Implicit type conversion occurs in equality comparison.
PrimitiveToObject	Program is trying to convert primitive value to an object.
Shadowing	Function, parameter or variable x is shadowed by a function, parameter or variable.
UnreachableCode	Following codes will never be executed.
UncalledFunction	Function f will never be called.
UnusedVarProp	Value assigned to a variable or an object property x will never be used.
VaryingTypeArguments	Type of parameter x to a function f is varying.

Table : Definition of 15 instances of warning

Design and Implementation of *New* Bug-Detecting Framework

- Based on the **analysis result** of SAFE Analyzer
- **Detect all** of the bugs we defined
- **Modularly designed** structure (easy to modify, prove, customize, …)



Core Modules (ExprDetect, InstDetect, FinalDetect)

ExprDetect	Detect Expression-level bugs ConvertUndefToNum, ImplicitTypeConversion, …
InstDetect	Detect Instruction-level bugs ObjectNullOrUndefined, CallNonFunction, …
ExprDetect	Detect yet uncaught bugs UncalledFunctions, VaryingTypeArguments, …

Core Modules (BugDetector)

```
P : Node*;     T : inTable;     B : BugStorage;
begin
    T := Analyzer (P, {});     B := {};
    repeat
        curr := P.head;
        P := P.tail;
        B := $C_{Bug}$ (cmdMap[curr.Label], T[curr], B);     /* detect bugs */
    until P = {};                                         /* end of program */
    B = FinalDetect (B);                                  /* detect uncaught bugs */
    return B;                                             /* report detected bugs */
end
```

```
N : Node;     S : State;     B : BugStorage;     /* arguments */
begin
    IF  N == Block (insts)  THEN
        repeat
            inst := insts.head;
            insts := insts.tail;
            B := $I_{Bug}$ (inst, S, B);
        until insts = {};                /* no more instructions */
        return B;
    ELSE
        return B;
end
```

```
I : Inst;     S : State;     B : BugStorage;          /* arguments */
begin
    IF  I == CFGAlloc (expr?)  THEN
        B = $V_{Bug}$ (expr?, S, B);
    ELIF  I == CFGCall (expr1, expr2, expr3)  THEN
        B = $V_{Bug}$ (expr1, S, B);   B = $V_{Bug}$ (expr2, S, B);   B = $V_{Bug}$ (expr3, S, B);
    …
    ELSE
        /* do nothing */
    return InstDetect (I, S, B);
end
```

```
E : Expr;     S : State;     B : BugStorage;       /* arguments */
begin
    IF  E == CFGBin (expr1, op, expr2)  THEN
        B = $V_{Bug}$ (expr1, S, B);   B = $V_{Bug}$ (expr2, S, B);
    …
    ELSE
        /* do nothing */
    return ExprDetect (E, S, B);
end
```

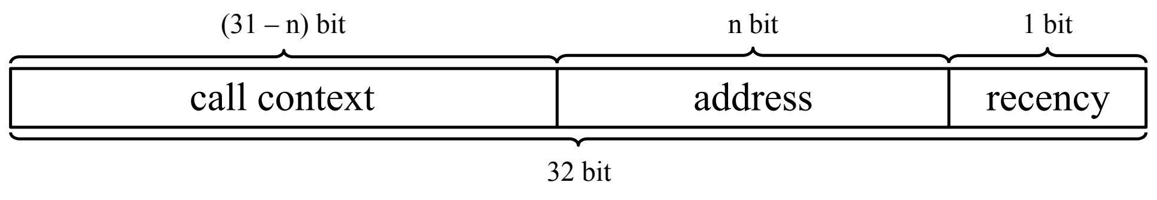
False-Positive Minimization

Object-Sensitive Analysis using Location Cloning

Refined Object Location

$$\hat{l} \in \widehat{\mathsf{Loc}}_{old} = \widehat{\mathsf{Address}} \times \widehat{\mathsf{RecencyTag}}$$
$$\hat{l} \in \widehat{\mathsf{Loc}}_{new} = \widehat{\mathsf{CallContext}} \times \widehat{\mathsf{Address}} \times \widehat{\mathsf{RecencyTag}}$$

In practice,
at most few thousands of objects and dozens of call contexts



	number of error	number of warning
no option	25	97
1-context*	28	95
loc**	0	66
1-context & loc	0	21
5-context & loc	0	18

100%

81%

all true alarms!

User-Configurable Bug Options

More ***general*** approach to reduce false positives

➢ **Filtering Options**

Filter out the bugs that can be found only in some of all possible states, object locations, abstract values, and types

➢ **Restricting Options**

Force to detect bugs only when the bugs meet the conditions that user provided

Experiment results ☞

Benchmark	#Line	No Options			Bug Options		
		time(s)	#err	#war	time(s)	#err	#war
bitops-bitwise-and	42	0.02	0	0	0.02	0	0
bitops-3bit-bits-in-byte	46	0.06	0	0	0.06	0	0
bitops-bits-in-byte	35	0.08	0	0	0.09	0	0
3d-morph	68	0.16	0	2	0.16	0	0
access-nsieve	52	0.13	0	1	0.13	0	0
bitops-nsieve-bits	46	0.15	0	4	0.14	0	0
math-cordic	109	0.35	0	4	0.45	0	0
math-partial-sums	47	0.14	0	0	0.15	0	0
access-fannkuch	80	0.38	0	19	0.55	0	1
crypto-sha1	238	0.44	0	26	0.44	0	1
access-nbody	183	0.47	**43**	6	0.54	**0**	0
string-base64	149	0.59	**1**	34	0.57	**1**	0
math-spectral-norm	65	0.40	0	7	0.42	0	0
controlflow-recursive	39	0.44	0	2	0.43	0	0
string-fasta	99	0.57	0	11	0.53	0	0
access-binary-trees	64	0.90	**1**	7	0.93	**0**	0
splay	401	1.62	**51**	5	1.51	**0**	0
richards	544	55.40	**28**	95	5.33	**0**	0
3d-raytrace	456	3.66	**23**	59	3.59	**0**	8
crypto-md5	300	51.56	0	80	1.48	0	1
3d-cube	351	3.82	**31**	122	3.87	**0**	1
deltablue	885	52.71	**125**	85	54.29	**7**	3
crypto	1704	60.56	**155**	598	60.98	**1**	15

94% ~ 100%

ignorable alarms

Contributions

My work …

- is the very first attempt to **provide definitions** of JavaScript bugs and **formal representation** of their semantics.
- provides **design and implementation** of scalable bug-detecting **framework** in detail.
- provides different approaches to **minimize false positives** among bug reports.
- makes the source code of the framework **open to the public** for the JavaScript community.

Future Work

- **Provide more elaborated bug options**
  - **Bug categorization**: automatic bug-option configuration
  - **Bug hierarchy**: selective bug reports according to priority orders
- **Loop Sensitive Analysis**
  - One of the main causes of imprecise analysis results
  - More precise analysis in for-loops with clear condition