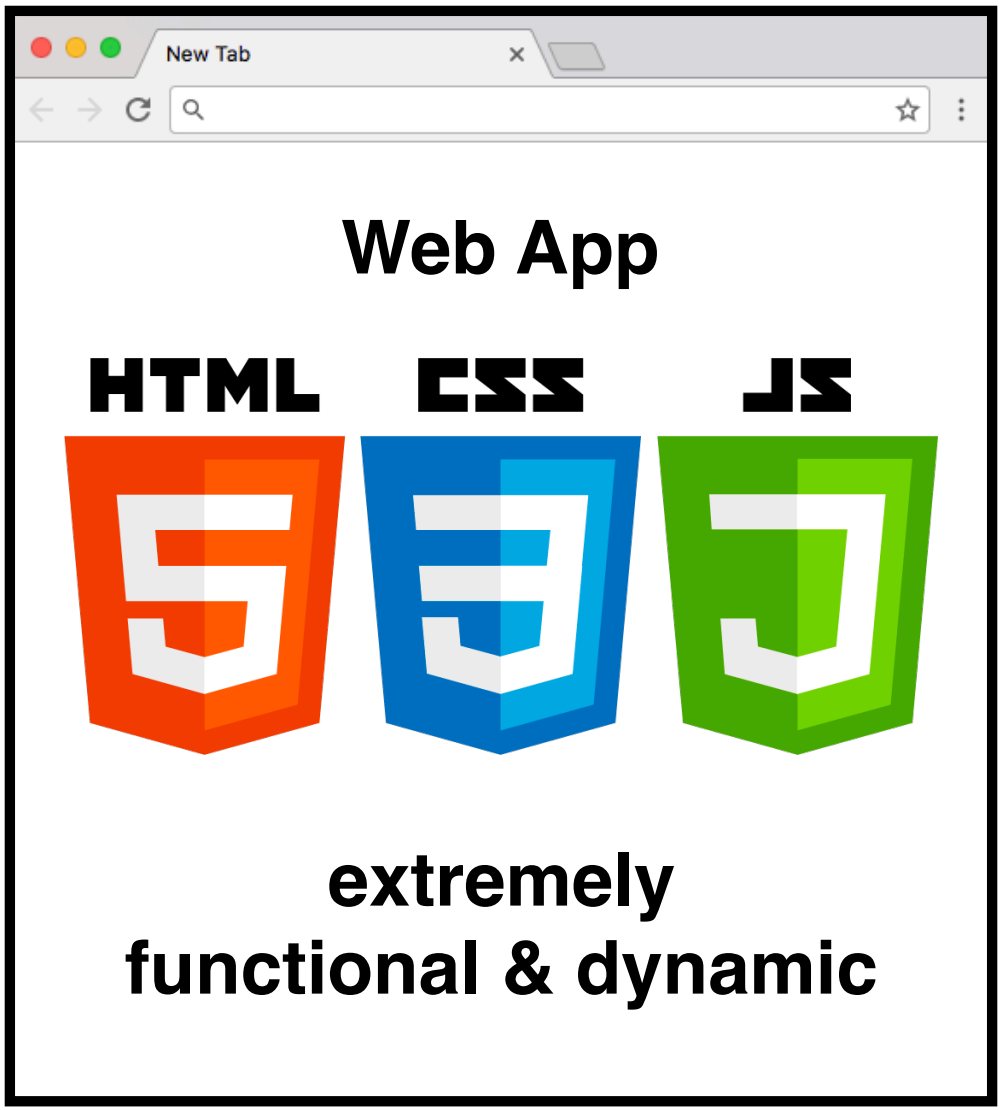
Analysis of JavaScript Web Applications Using SAFE 2.0

ICSE Demo 2017

Jihyeok Park, Yeonhee Ryou, Joonyoung Park, Sukyoung Ryu

PLRG @ KAIST

JavaScript Web Application



JavaScript Web Application

Error

Analyzer

JavaScript Web Application

Error

Analyzer

JavaScript Web Application

Error

Analyzer

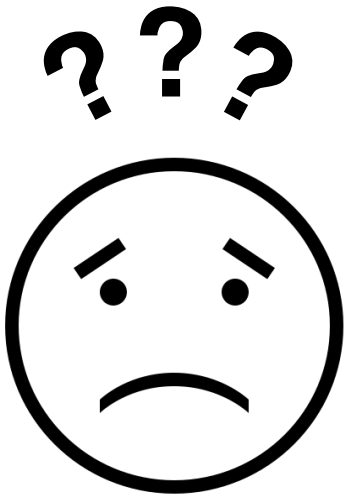
JavaScript Web Application

Error

Analyzer

Use by Analysis Developers

- Poor **usability** for **analysis developers**

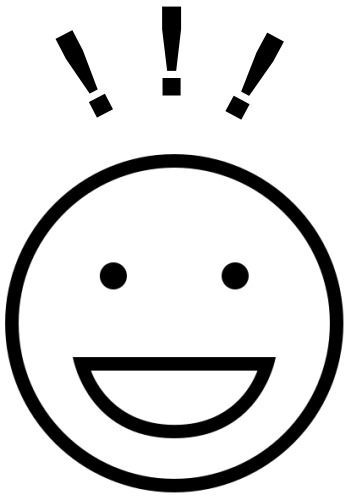


Analysis Developers

- **configure** analysis techniques
- **add** new techniques
- **debug** their implementation

SAFE 2.0

- Advanced version of **SAFE**
- **SAFE 2.0** with **three main features**



Analysis Developers

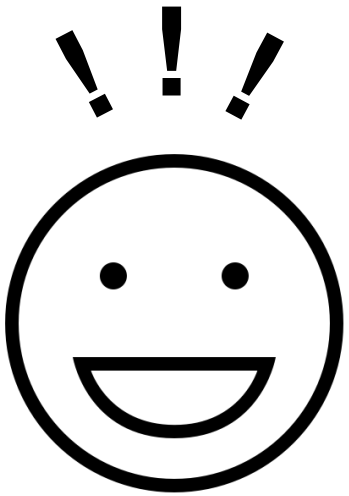
Pluggability

Extensibility

Debuggability

SAFE 2.0

- Advanced version of **SAFE**
- **SAFE 2.0** with **three main features**



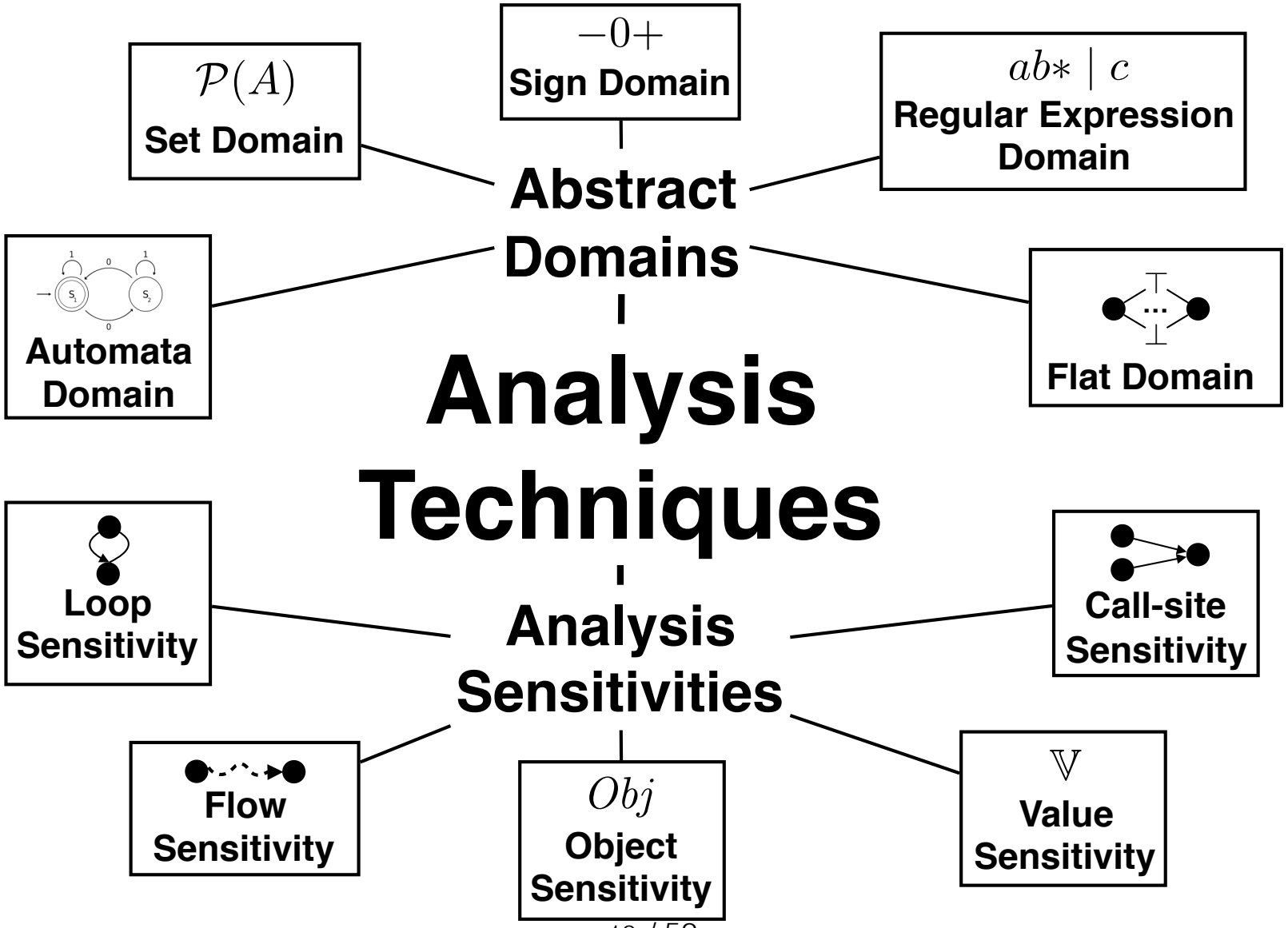
Analysis Developers

Pluggability

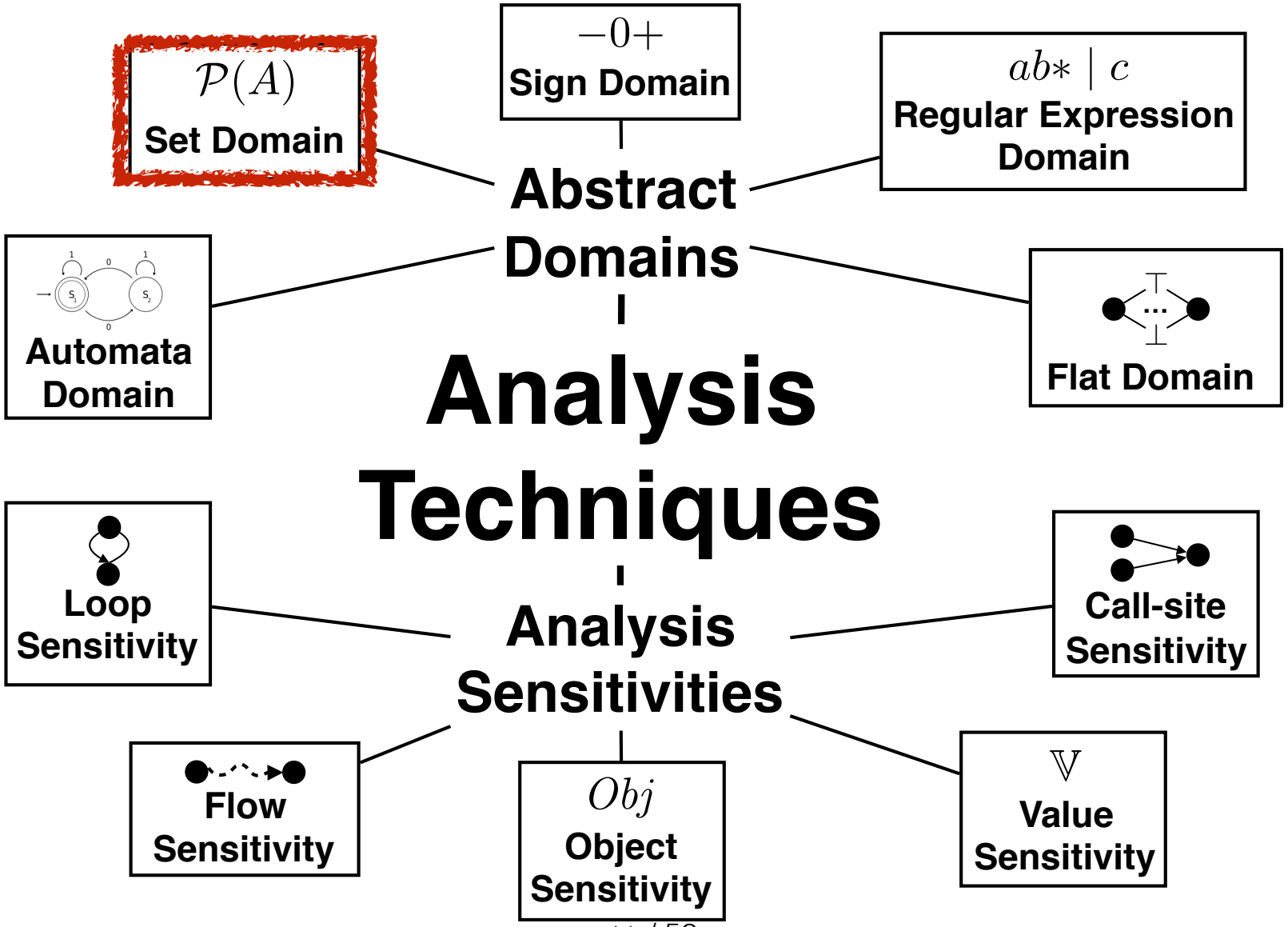
Extensibility

Debuggability

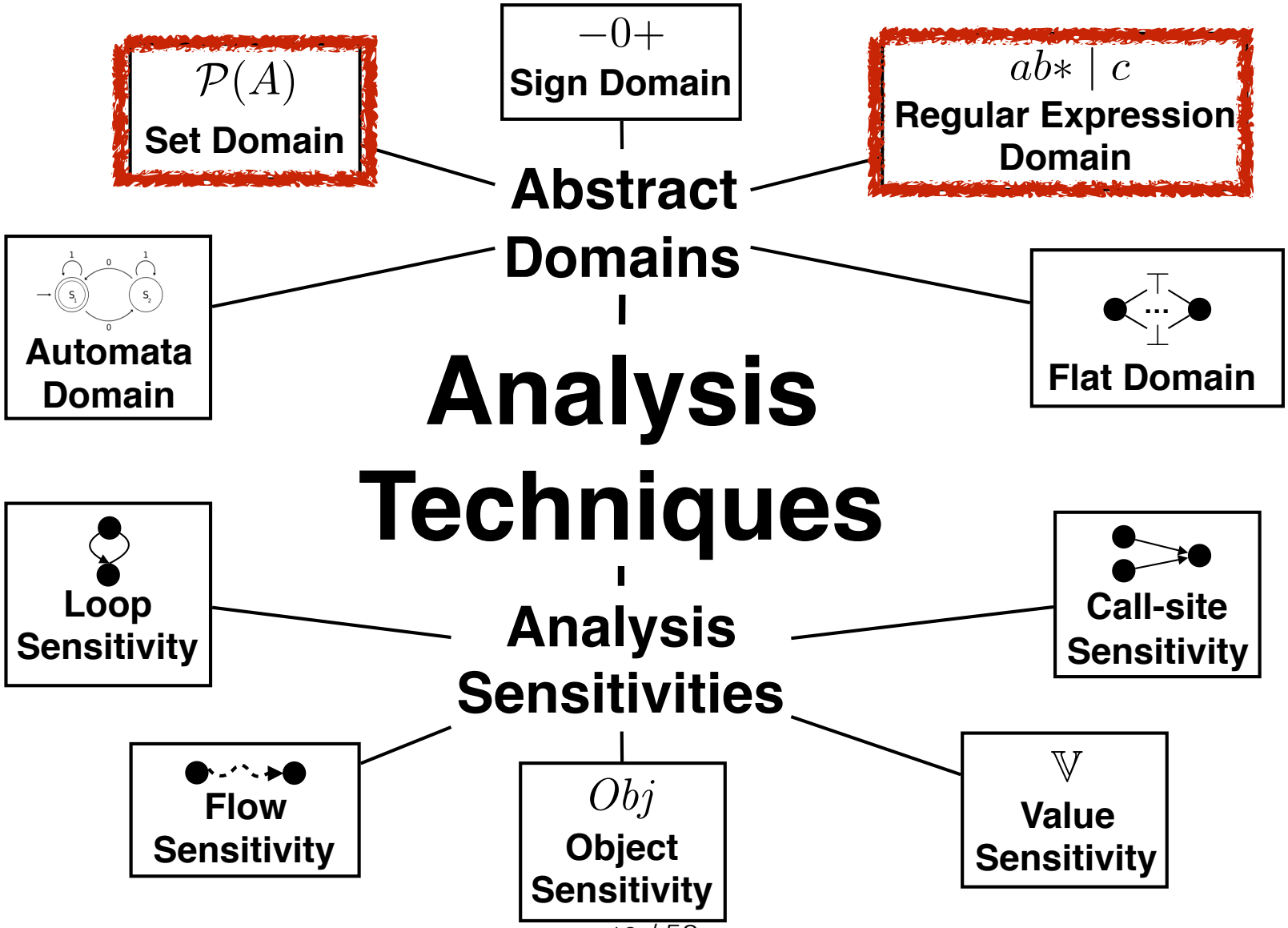
Pluggability



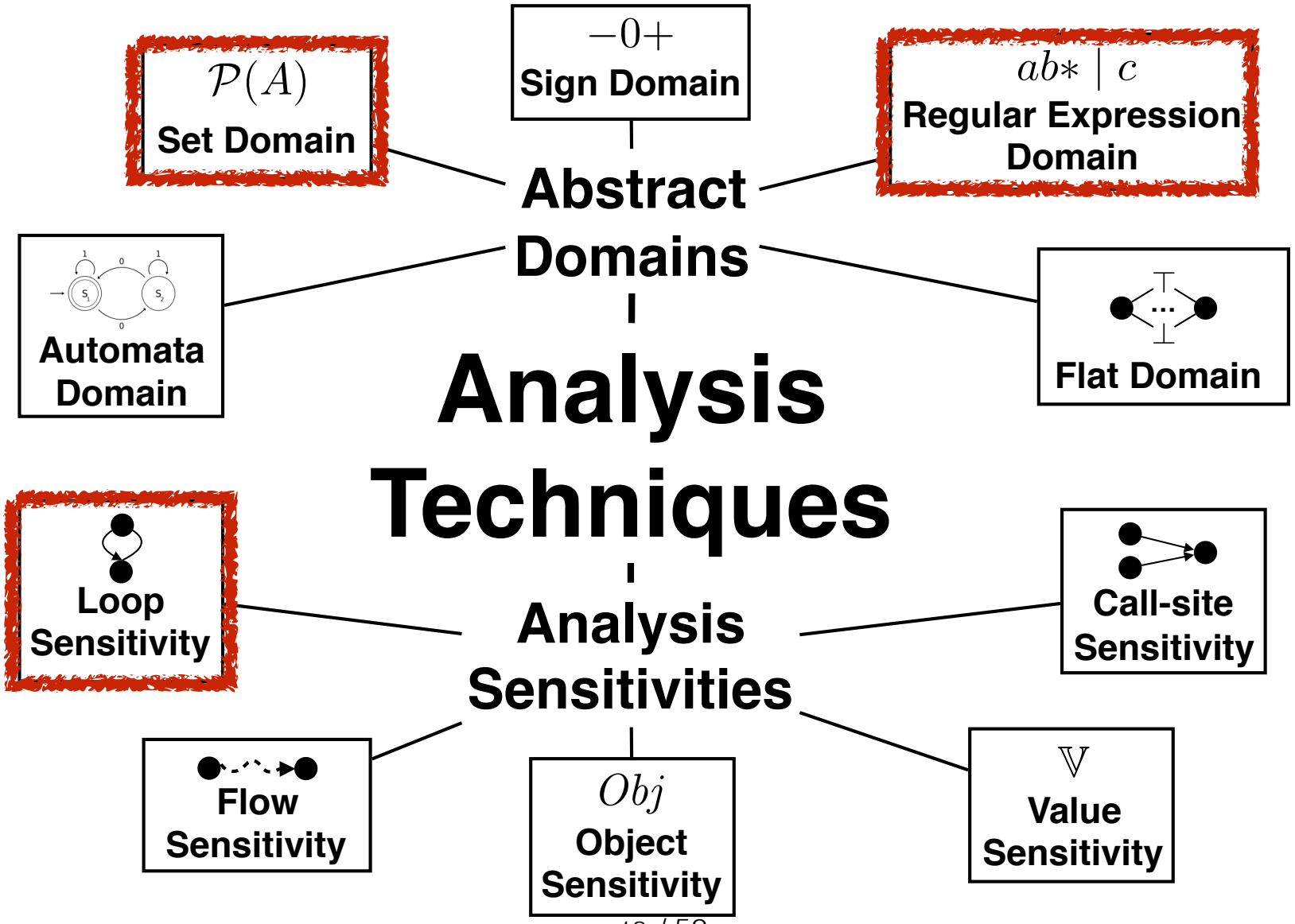
Pluggability



Pluggability

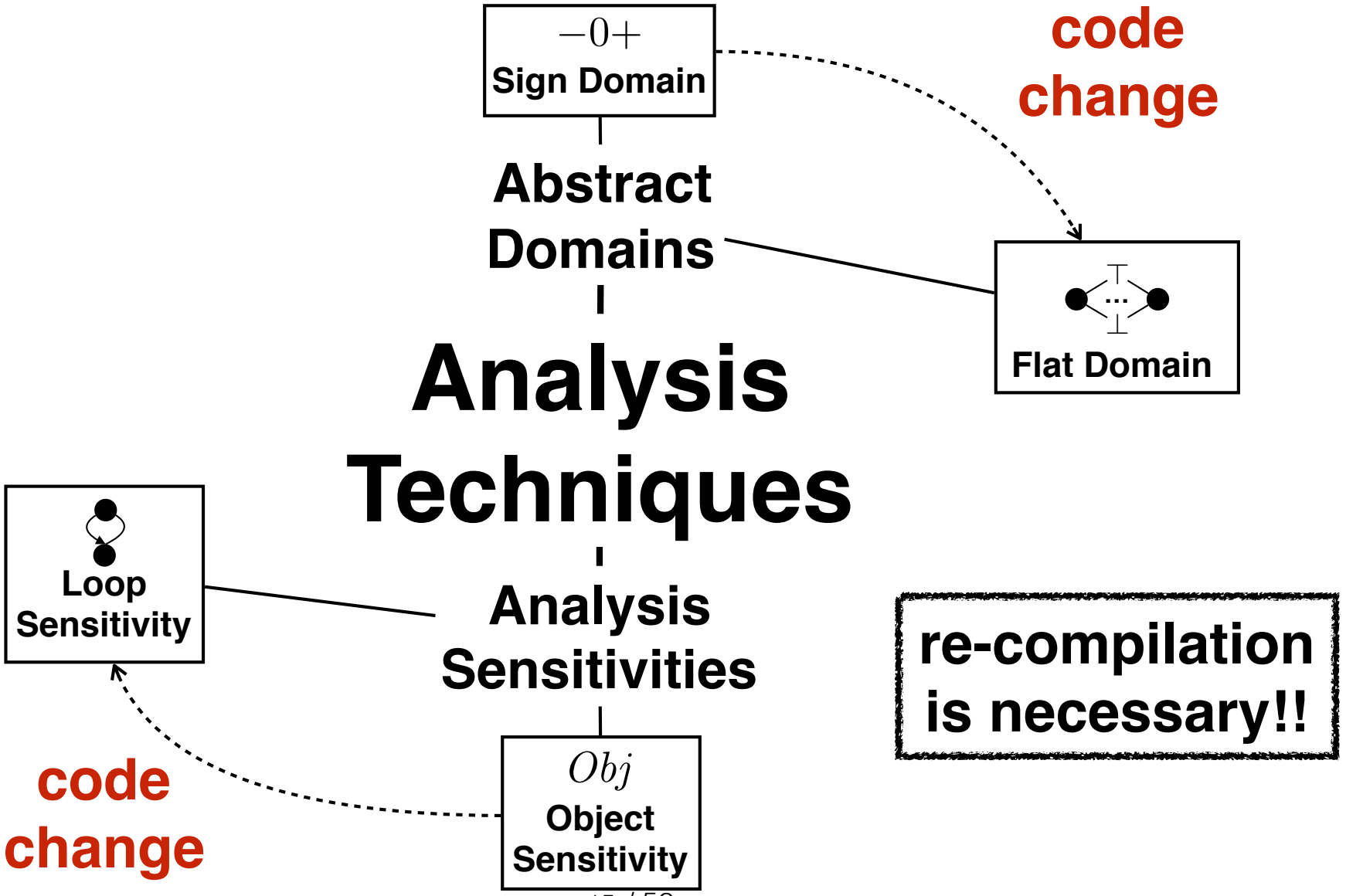


Pluggability



Pluggability

Pluggability



Pluggability

in SAFE 2.0

```
{
  "silent": false,
  "analyzer": {
    "number": "set",
    "string": "regular",
    "loopSens": 10,
  }
}
```

config1.json

Pluggability

in SAFE 2.0

```
{
  "silent": false,
  "analyzer": {
    "number": "set",
    "string": "regular",
    "loopSens": 10,
  }
}
```

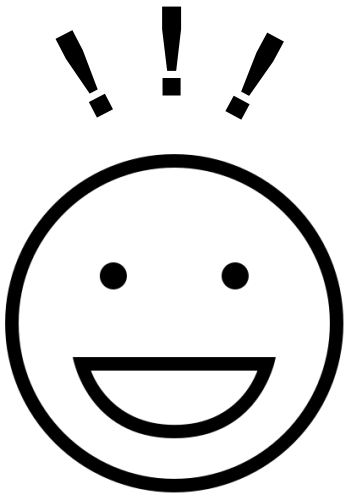
config1.json

```
{
  "silent": false,
  "analyzer": {
    "number": "flat",
    "string": "regular",
    "loopSens": 10,
  }
}
```

config2.json

SAFE 2.0

- Advanced version of **SAFE**
- **SAFE 2.0** with **three main features**



Analysis Developers

Pluggability

Extensibility

Debuggability

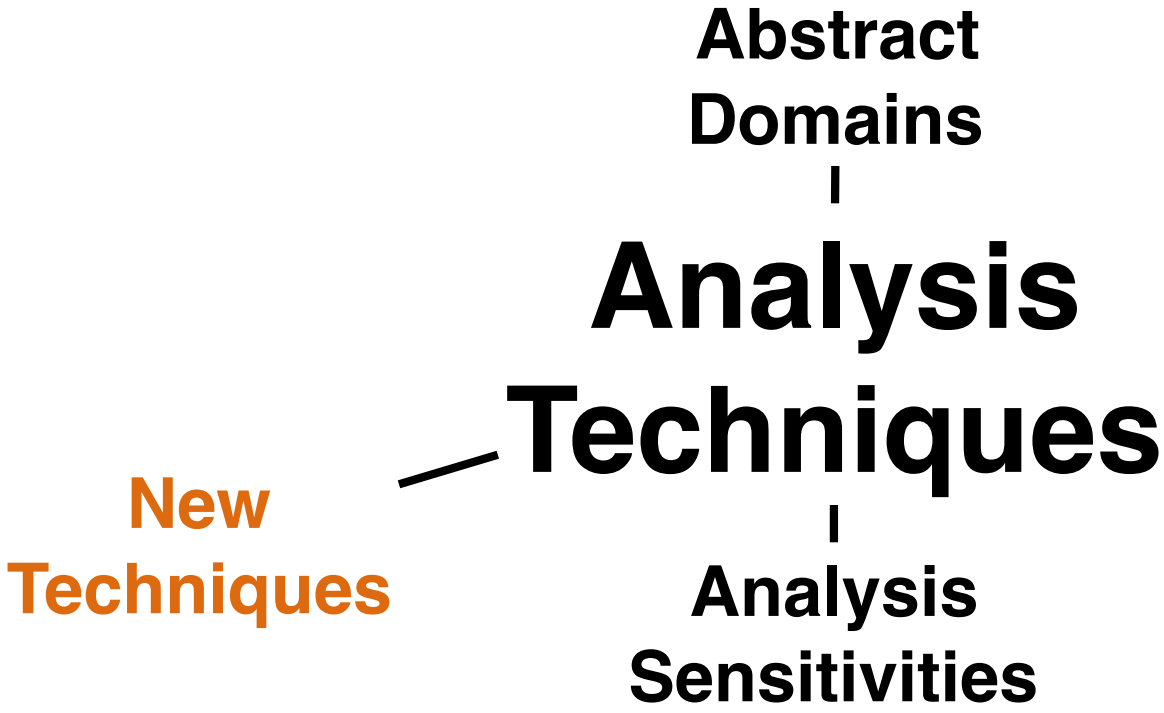
Extensibility

Abstract Domains

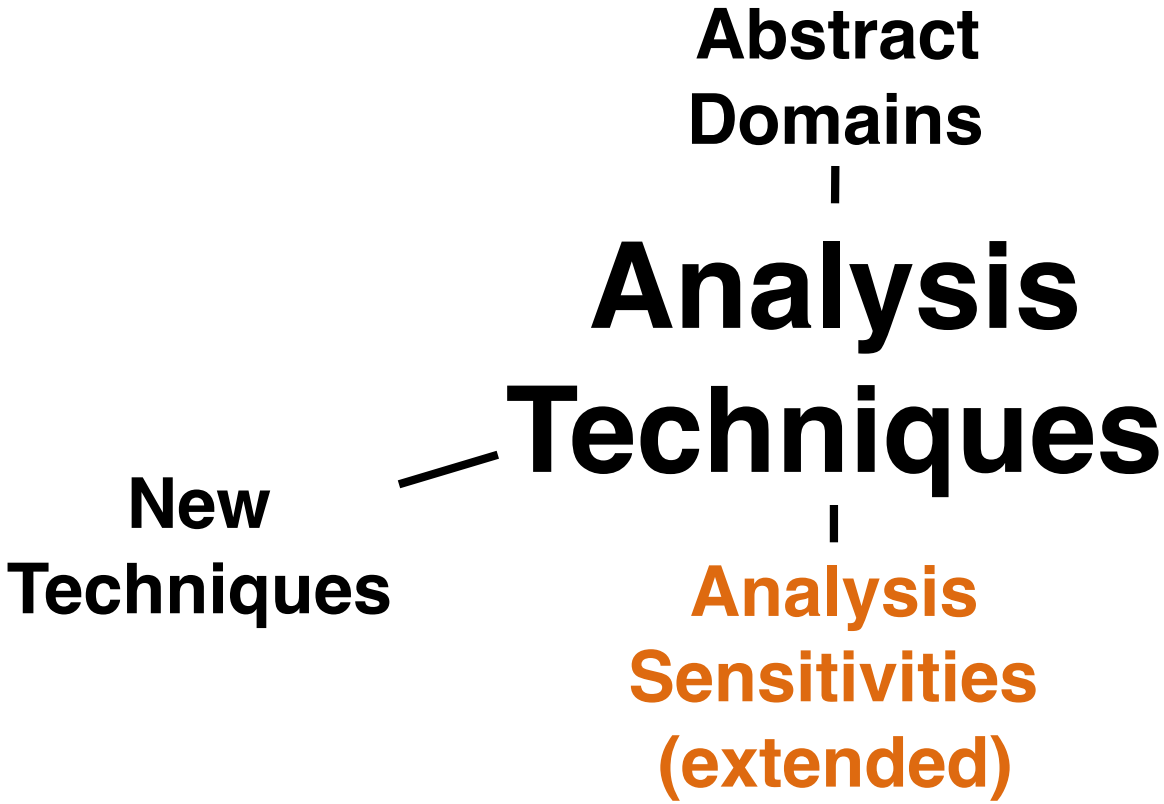
Analysis Techniques

Analysis Sensitivities

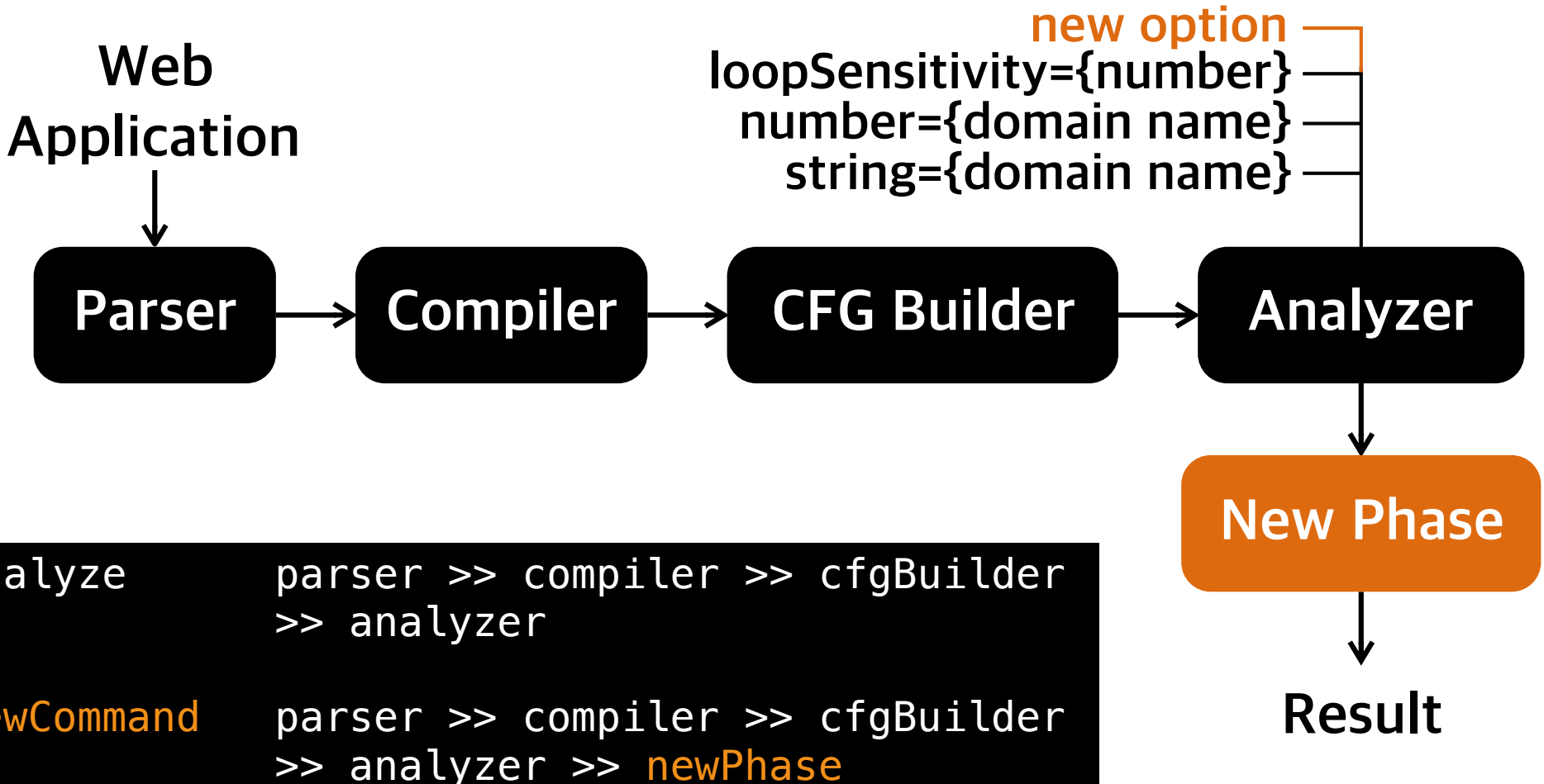
Extensibility



Extensibility



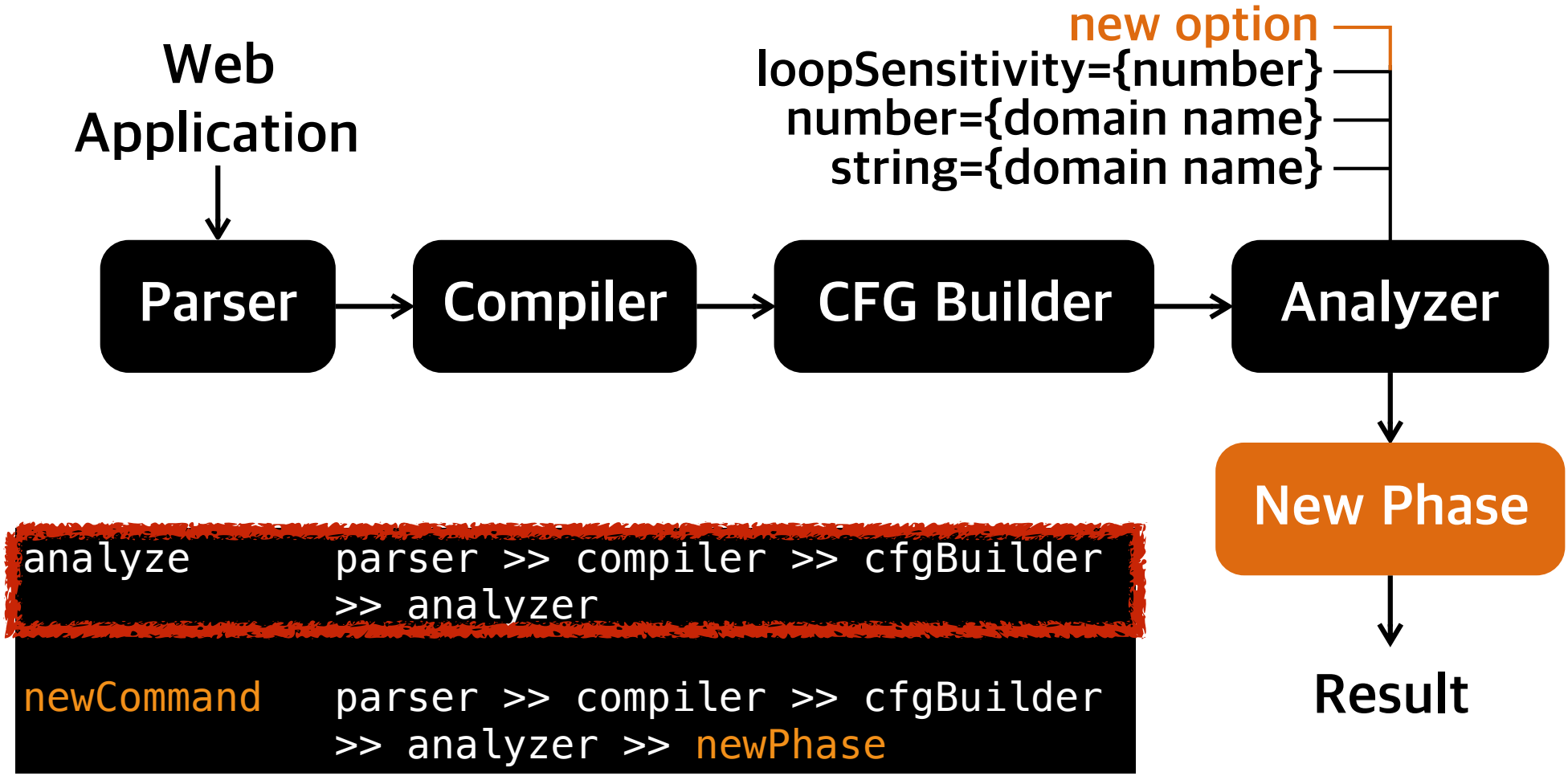
Extensibility




```
analyze       parser >> compiler >> cfgBuilder
              >> analyzer

newCommand    parser >> compiler >> cfgBuilder
              >> analyzer >> newPhase
```

Extensibility




```
analyze       parser >> compiler >> cfgBuilder
              >> analyzer

newCommand    parser >> compiler >> cfgBuilder
              >> analyzer >> newPhase
```

Extensibility

2015 30th IEEE/ACM International Conference on Automated Software Engineering

Static Analysis of JavaScript Web Applications in the Wild via Practical DOM Modeling

Changhee Park, KAIST, changhee.park@kaist.ac.kr
Sooncheol Won, KAIST, woncsh@kaist.ac.kr
Joonho Jin, KAIST, myfriend12@kaist.ac.kr
Sukyoung Ryu, KAIST, sryu.cs@kaist.ac.kr

Abstract—We present SAFE$_{\text{TAGS}}$, an open-source static analysis framework for JavaScript web applications. It provides a faithful (partial) model of web application execution environments of various browsers, based on empirical data from the main web pages of the 9,465 most popular websites. A main feature of SAFE$_{\text{TAGS}}$ is the configurability of DOM tree abstraction levels to allow users to adjust a trade-off between analysis performance and precision, depending on their applications. We evaluate SAFE$_{\text{TAGS}}$ on the 5 most popular JavaScript libraries and the main web pages of the 10 most popular websites. Additionally, to show its performance, precision, and modeling coverage, we build a bug detector as an application of SAFE$_{\text{TAGS}}$; we build a bug detector for JavaScript web applications that uses static analysis results from SAFE$_{\text{TAGS}}$. Our bug detector found previously undiscovered bugs including ones from wikipedia.org and amazon.com.

I. Introduction

JavaScript was originally developed as a simple scripting language to develop interactive and dynamic web pages, but now it is used to construct large web applications with the evolution of web contents. Although JavaScript is the most dominant programming language on the web [1], its quirky features make it difficult to understand behaviors of complex JavaScript programs. Many researchers have developed various program analysis techniques to alleviate this problem, but many of them have focused on analyzing stand-alone JavaScript programs [28], [4], [13], which is not enough for analyzing real-world web applications.

To analyze JavaScript web applications in the wild, modeling browser environments is critical since JavaScript programs often reside in web pages and interact with users and browsers via events. However, one of the main challenges is that no single standard document describes browser environments formally. While some commonly-used specifications are available online [6], [7], their descriptions are informal, and incomplete and modeling all of them manually is tedious, time consuming, and error prone. Furthermore, non-standard and inconsistent behaviors of various browsers complicate modeling. While previous approaches presented partial models of browser environments selected from several documents, they are not applicable to real-world web applications because they support only subsets of JavaScript or limited models of web documents [4], [8], [9], [10], [11], [12].

In this paper, we present SAFE$_{\text{TAGS}}$, an open-source framework that statically analyzes JavaScript web applications and show how to extend the existing static analyzer SAFE [13] to support an extensive model of browser environments. By doing so, SAFE$_{\text{TAGS}}$ extends the applicability of any client applications that are built on top of SAFE from stand-alone JavaScript programs to web applications. To implement a practical model of browser environments, we investigated 9,465 popular websites and summarized uses of native JavaScript objects and APIs provided by most major web browsers. Based on extensive empirical data, we design a model of browser environments. A main feature of SAFE$_{\text{TAGS}}$ is the configurability of DOM tree abstraction levels in three ways, two from the literature [11], [12] and one new, which maintains precise tree structures of web documents. We show how DOM tree abstraction levels affect analysis performance and precision with the analysis results of the 5 most popular JavaScript libraries and the main web pages of the 10 most popular websites. Furthermore, we implement a bug detector capable of detecting language-level bugs as a client application of SAFE$_{\text{TAGS}}$ and show that it detects previously undiscovered bugs in the main web pages of popular websites. Note that our approach is not tied to its base analyzer, SAFE, and thus it is applicable to other static analyzers for JavaScript programs.

The contributions of this paper include the followings:

- We present *empirical data from the 9,465 most popular websites* to show the usage of native objects and APIs that browsers support. This data will be helpful to model browser environments practically.
- We explain how to extend a static analyzer that supports only stand-alone JavaScript programs to the one that supports web applications with a model of browser environments practically. This approach is applicable to other static JavaScript analyzers.
- We present SAFE$_{\text{TAGS}}$, *a framework that statically analyzes real-world JavaScript web applications*. It provides *configurable DOM tree abstraction models* in three levels, two existing ones [11], [12] and a new one that captures DOM tree structures of web documents precisely. We present experimental results about the effects of DOM tree abstraction levels on analysis results of real-world web applications such as JavaScript libraries and websites.
- To show the usefulness of SAFE$_{\text{TAGS}}$, we implement a bug detector as an application of SAFE$_{\text{TAGS}}$ and evaluate it on the main web pages of the 10 most popular websites. We *found previously undiscovered bugs* in `wikipedia.org` and `amazon.com`.
- We make SAFE$_{\text{TAGS}}$ and the bug detector *publicly available* [14].

978-1-5090-0025-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ASE.2015.27

Bug Detector

ASE'15

By New Phase

Extensibility


```
23 // BugDetect phase
24 case object BugDetector extends PhaseObj[
25   AnalysisResult, BugDetectConfig, BugReport
26 ] {
27   val name: String = "bugDetector"
28   val help: String = "Detect possible bugs in
29                       JavaScript source files."
30   val options: List[PhaseOption[BugDetectConfig]] = List(
31     ("silent", BoolOption(c => c.silent = true),
32       "messages during bug detection are muted.")
33   )
34
35   def apply(
36     in: AnalysisResult,
37     safeConfig: SafeConfig,
38     config: BugDetectConfig
39   ): Try[BugReport] = { ... }
```

Extensibility


```
23 // BugDetect phase
24 case object BugDetector extends PhaseObj[
25   AnalysisResult, BugDetectConfig, BugReport
26 ] {
27   val name: String = "bugDetector"
28   val help: String = "Detect possible bugs in
29                       JavaScript source files."
30   val options: List[PhaseOption[BugDetectConfig]] = List(
31     ("silent", BoolOption(c => c.silent = true),
32       "messages during bug detection are muted.")
33   )
34
35   def apply(
36     in: AnalysisResult,
37     safeConfig: SafeConfig,
38     config: BugDetectConfig
39   ): Try[BugReport] = { ... }
```

Extensibility


```
23 // BugDetect phase
24 case object BugDetector extends PhaseObj[
25   AnalysisResult, BugDetectConfig, BugReport
26 ] {
27   val name: String = "bugDetector"
28   val help: String = "Detect possible bugs in
29                       JavaScript source files."
30   val options: List[PhaseOption[BugDetectConfig]] = List(
31     ("silent", BoolOption(c => c.silent = true),
32       "messages during bug detection are muted.")
33   )
34
35   def apply(
36     in: AnalysisResult,
37     safeConfig: SafeConfig,
38     config: BugDetectConfig
39   ): Try[BugReport] = { ... }
```

Extensibility


```
23 // BugDetect phase
24 case object BugDetector extends PhaseObj[
25   AnalysisResult, BugDetectConfig, BugReport
26 ] {
27   val name: String = "bugDetector"
28   val help: String = "Detect possible bugs in
29                       JavaScript source files."
30   val options: List[PhaseOption[BugDetectConfig]] = List(
31     ("silent", BoolOption(c => c.silent = true),
32       "messages during bug detection are muted.")
33   )
34
35   def apply(
36     in: AnalysisResult,
37     safeConfig: SafeConfig,
38     config: BugDetectConfig
39   ): Try[BugReport] = { ... }
```

Basic Info

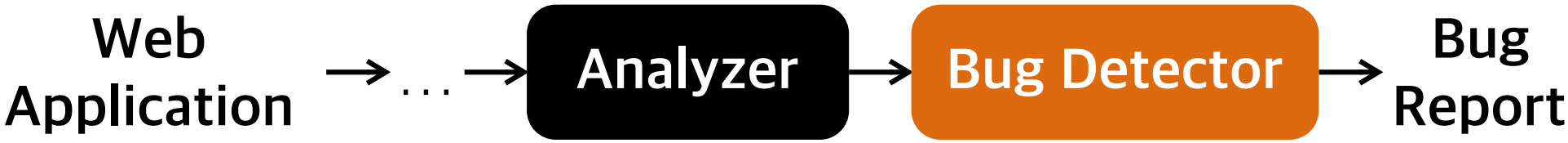
Extensibility


```
23 // BugDetect phase
24 case object BugDetector extends PhaseObj[
25   AnalysisResult, BugDetectConfig, BugReport
26 ] {
27   val name: String = "bugDetector"
28   val help: String = "Detect possible bugs in
29                       JavaScript source files."
30   val options: List[PhaseOption[BugDetectConfig]] = List(
31     ("silent", BoolOption(c => c.silent = true),
32       "messages during bug detection are muted.")
33   )
34
35   def apply(
36     in: AnalysisResult,
37     safeConfig: SafeConfig,
38     config: BugDetectConfig
39   ): Try[BugReport] = { ... }
```

Basic Info

Core functionality

Extensibility




```
136 ...
137 // androidCheck
138 case object CmdAndroidCheck extends CommandObj("androidCheck",
139   CmdBase >> AndroidCheck
140 )
141
142 // help
143 case object CmdHelp extends CommandObj("help", CmdBase >> Help)
144
145 // bugDetect
146 case object CmdBugDetect extends CommandObj("bugDetect",
147   CmdAnalyze >> BugDetector
148 )
```

bugDetect Command

Extensibility


```
$ safe bugDetect wikipedia.org.htm
wikipedia.org.htm:179:7-36:
  [Warning] The conditional expression "uiLang.match(new
            RegExp("^\w+", "")) === lang" is always false.
wikipedia.org.htm:192:5-18:
  [Warning] The property "chrome" of the object "window"
            is absent.
The command 'bugDetect' took 2547 ms.
$
```

False Alarm!!

Extensibility

Battles with False Positives in Static Analysis of JavaScript Web Applications in the Wild

Joonyoung Park, KAIST, gmb55@kaist.ac.kr
Inho Lim, Samsung Electronics, inho0212.lim@samsung.com
Sukyoung Ryu, KAIST, sryu.cs@kaist.ac.kr

ABSTRACT

Now that HTML5 technologies are everywhere from web services to various platforms, ensuring quality of web applications become very important. While web application developers use syntactic checkers and type-related bug detectors, extremely dynamic features and diverse execution environments of web applications make it particularly difficult to statically analyze them leading to too many false positives. Recently, researchers have developed static analyzers for JavaScript web applications addressing quirky JavaScript language semantics and browser environments, but they lack empirical studies on the practicality of such analyzers.

In this paper, we collect 30 JavaScript web applications in the wild, analyze them using SAFE, the state-of-the-art JavaScript static analyzer with bug detectors, and investigate false positives in the analysis results. After manually inspecting them, we classify 7 reasons that cause the false positives: W3C APIs, browser-specific APIs, JavaScript library APIs, dynamic file loading, dynamic code generation, asynchronous calls, and others. Among them, we identify 4 cases which are the sources of false positives that we can practically reduce. Rather than striving for sound analysis with unrealistic assumptions, we choose to be intentionally unsound to analyze web applications in the real world with less false positives. Our evaluation shows that the approach effectively reduces false positives in statically analyzing web applications in the wild.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

Static analysis; JavaScript; web applications; false positives

1. INTRODUCTION

Using multiple devices with multiple platforms becomes normal, which makes cross-platform web applications prevalent on a variety of platforms including the web and mobile devices. Web applications are built on top of HTML5 [5] technologies consisting HTML, CSS, and JavaScript; they can run anywhere that supports browsers. Because they are cross-platform, end-users can enjoy the same applications on different devices, and developers can build one application for multiple platforms. Indeed, major operating systems support most browsers and various mobile platforms such as Tizen[1], Chromium OS[2], and Firefox OS[3].

The more web applications become prevalent, the more quality assurance of web applications gets important, but developer tools for building reliable web applications are not yet ready. While JSLint[4], extremely dynamic and syntactic checkers like JSLint[4], extremely dynamic and functional features of JavaScript make it difficult to syntactically analyze them precisely. Researchers have reported that the lack of static types, dynamic addition and removal of object properties, object property names as first-class values, and dynamic code generation by eval-like functions all contribute to the complication of JavaScript static analysis [27]. Analysis of such features are beyond the scope of syntactic checking. In addition, diverse execution environments of web applications make the situation even worse because different environments provide different sets of functionalities that are targets of analysis [6].

Recent development in static analysis of JavaScript web applications enables analyzers to address quirky JavaScript language semantics and different browser environments, but their usability has not been evaluated with web applications in the wild. WALA [10, 29], the first static analyzer for JavaScript applications, provides a flow-insensitive points-to analysis, but it does not support the HTML DOM and the browser APIs required for analyzing web applications. TAJS [23, 13, 12] is based on the abstract interpretation framework [8], and it supports a flow-sensitive type analysis with a limited model of web documents. SAFE [15, 18] is similar to TAJS but it additionally supports detection of Web API misuses [6] and a systematic modeling of web documents [25]. They are actively expanding the set of JavaScript web applications that they can analyze, but they are still in the stage of striving for producing sound but possibly imprecise analysis results with unrealistic assumptions.

[1] https://www.tizen.org
[2] https://www.chromium.org/chromium-os
[3] https://www.mozilla.org/en-US/firefox/os
[4] http://www.jslint.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE '16 May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4205-6...$15.00
DOI: 10.475/123.4

Snapshot (Dynamic Info.)

ICSE'16

By New Option

Extensibility

Analyzer — new option

```
127       ...
128       ("jsModel", BoolOption(c => c.jsModel = true),
129         "analysis with JavaScript models."),
130       ("snapshot", StrOption((c, s) => c.snapshot = Some(s)),
131         "analysis with an initial heap generated from
132           a dynamic snapshot(*.json).")
133     )
134 }
135
136 // Analyze phase config
137 case class AnalyzeConfig(
138   var snapshot: Option[String] = None,
139   var silent: Boolean = false,
140   var console: Boolean = false,
141   ...
```

Extensibility

Analyzer — new option

```
127       ...
128       ("jsModel", BoolOption(c => c.jsModel = true),
129         "analysis with JavaScript models."),
130       ("snapshot", StrOption((c, s) => c.snapshot = Some(s)),
131         "analysis with an initial heap generated from
132          a dynamic snapshot(*.json).")
133     )
134 }
135
136 // Analyze phase config
137 case class AnalyzeConfig(
138   var snapshot: Option[String] = None,
139   var silent: Boolean = false,
140   var console: Boolean = false,
141   ...
```

Extensibility

Analyzer — new option

```
127       ...
128       ("jsModel", BoolOption(c => c.jsModel = true),
129         "analysis with JavaScript models."),
130       ("snapshot", StrOption((c, s) => c.snapshot = Some(s)),
131         "analysis with an initial heap generated from
132          a dynamic snapshot(*.json).")
133     )
134 }
135
136 // Analyze phase config
137 case class AnalyzeConfig(
138   var snapshot: Option[String] = None,
139   var silent: Boolean = false,
140   var console: Boolean = false,
141   ...
```

Extensibility

Analyzer — new option

```
127       ...
128       ("jsModel", BoolOption(c => c.jsModel = true),
129         "analysis with JavaScript models."),
130       ("snapshot", StrOption((c, s) => c.snapshot = Some(s)),
131         "analysis with an initial heap generated from
132         a dynamic snapshot(*.json).")
133     )
134 }
135
136 // Analyze phase config
137 case class AnalyzeConfig(
138   var snapshot: Option[String] = None,
139   var silent: Boolean = false,
140   var console: Boolean = false,
141   ...
```

Extensibility

Analyzer — new option

Parse

```
52    ...
53    def addSnapshot(st: AbsState, snapshot: String): AbsState = {
54      val concreteHeap = Heap.parse(snapshot)
55      val abstractHeap = AbsHeap.alpha(concreteHeap)
56      AbsState(st.heap + abstractHeap, st.context)
57    }
58    ...
```

Abstract

Join

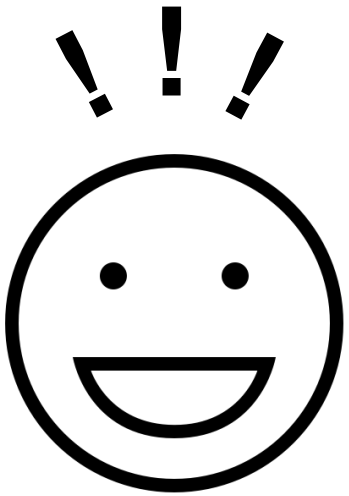
Extensibility

No False Alarm!!

```
$ safe bugDetect -analyzer:snapshot=chrome wikipedia.org.htm
wikipedia.org.htm:179:7-36:
  [Warning] The conditional expression “uiLang.match(new
            RegExp("^\w+", "")) === lang" is always false.
The command 'bugDetect' took 2168 ms.
$
```

SAFE 2.0

- Advanced version of **SAFE**
- **SAFE 2.0** with **three main features**



Analysis Developers

Pluggability

Extensibility

Debuggability

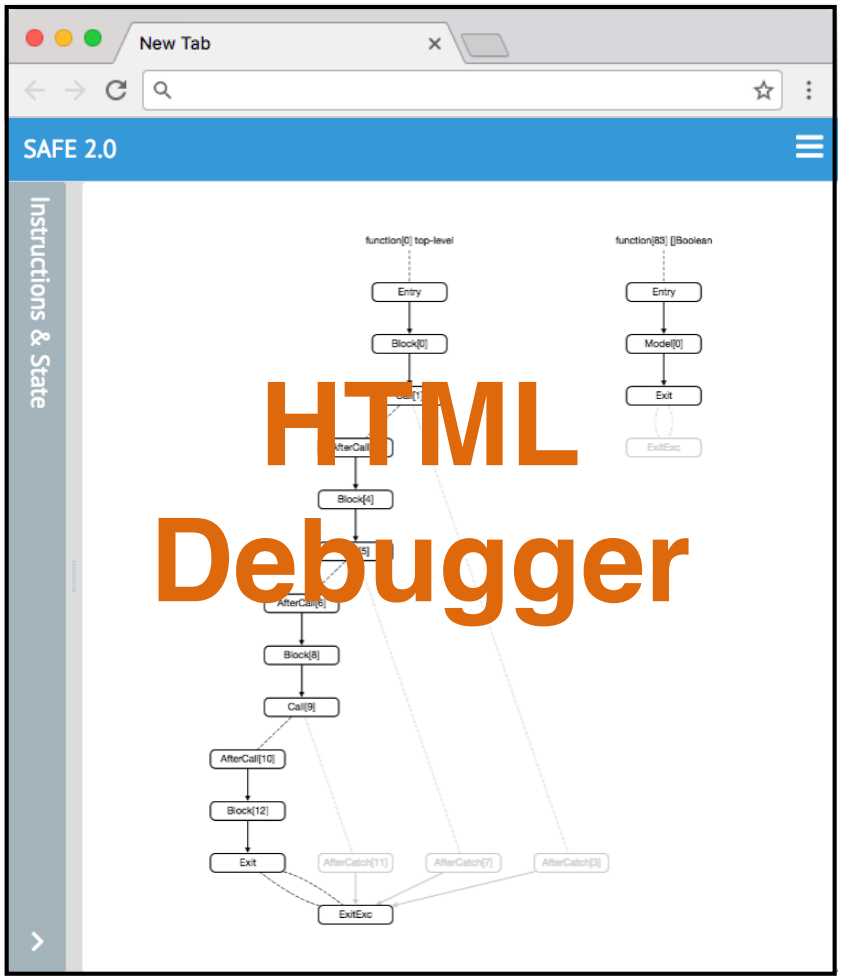
Debuggability

The official ECMAScript conformance suite

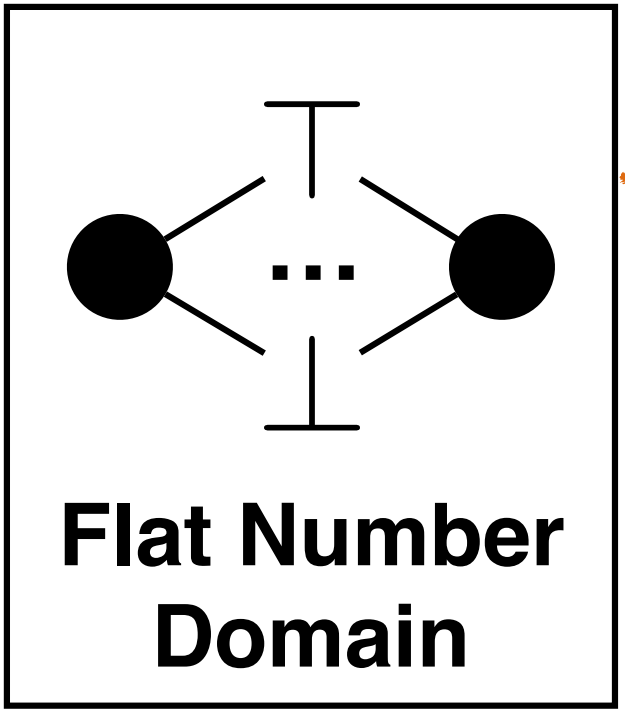
↑

Test262

tests



Debuggability



SAFE 2.0

Debuggability

```
> test262Test
[info] Run completed in 14 minutes, 49 seconds.
[info] Total number of tests run: 5997
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 5983, failed 14, canceled 0, ignored 0, pending 0
[info] *** 14 TESTS FAILED ***
[error] Failed tests:
[error]         kr.ac.kaist.safe.CoreTest
[error] (test:testOnly) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 896 s, completed Nov 16, 2016 12:45:02 AM
```

Test262 tests

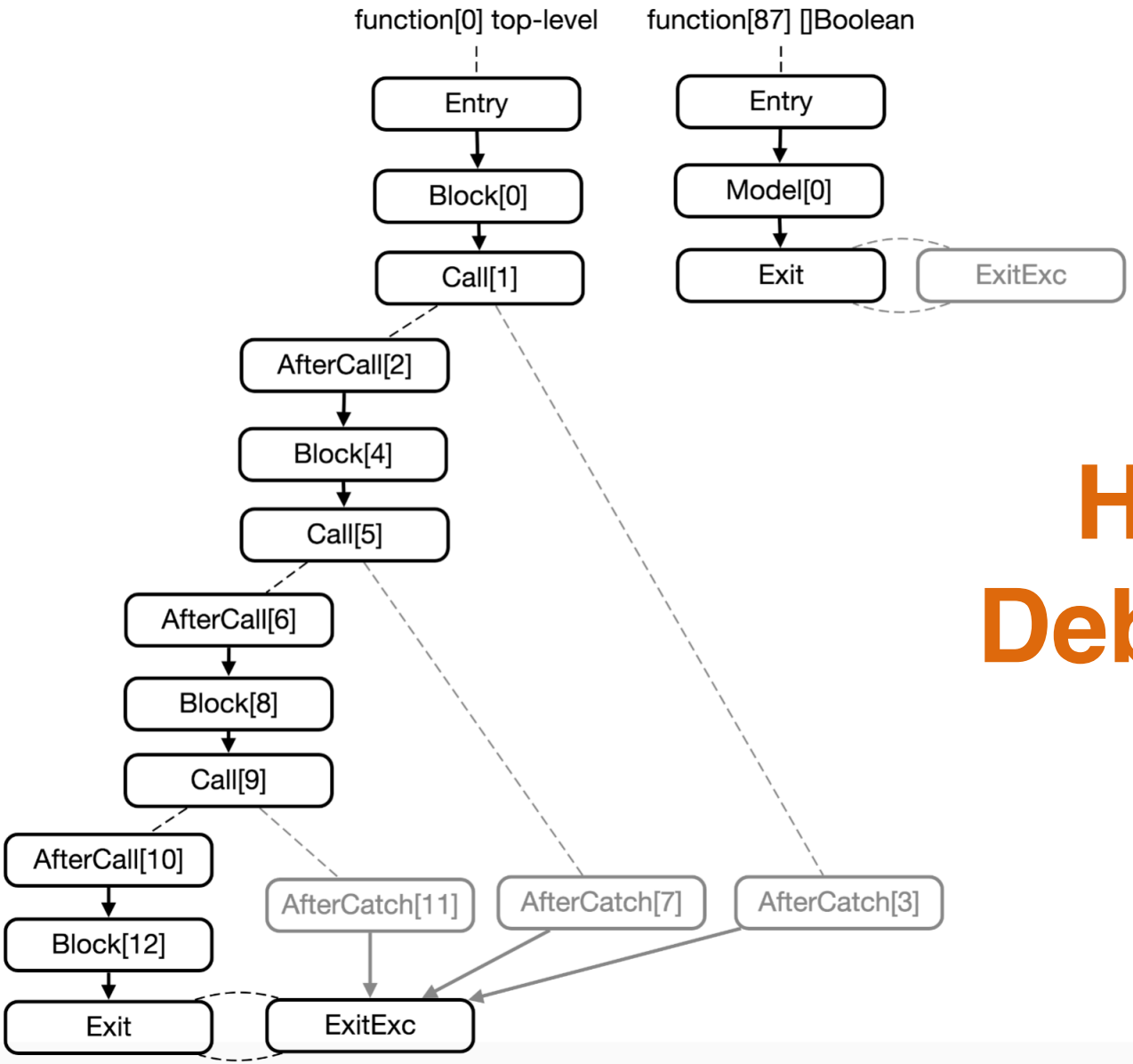
Debuggability

```
{
  var __result1 = Boolean(+ 0) !== false;
  var __expect1 = false;
}
{
  var __result2 = Boolean(- 0) !== false;
  var __expect2 = false;
}
{
  var __result3 = Boolean(Number.NaN) !== false;
  var __expect3 = false;
}
```

Debuggability

SAFE 2.0

Instructions & State

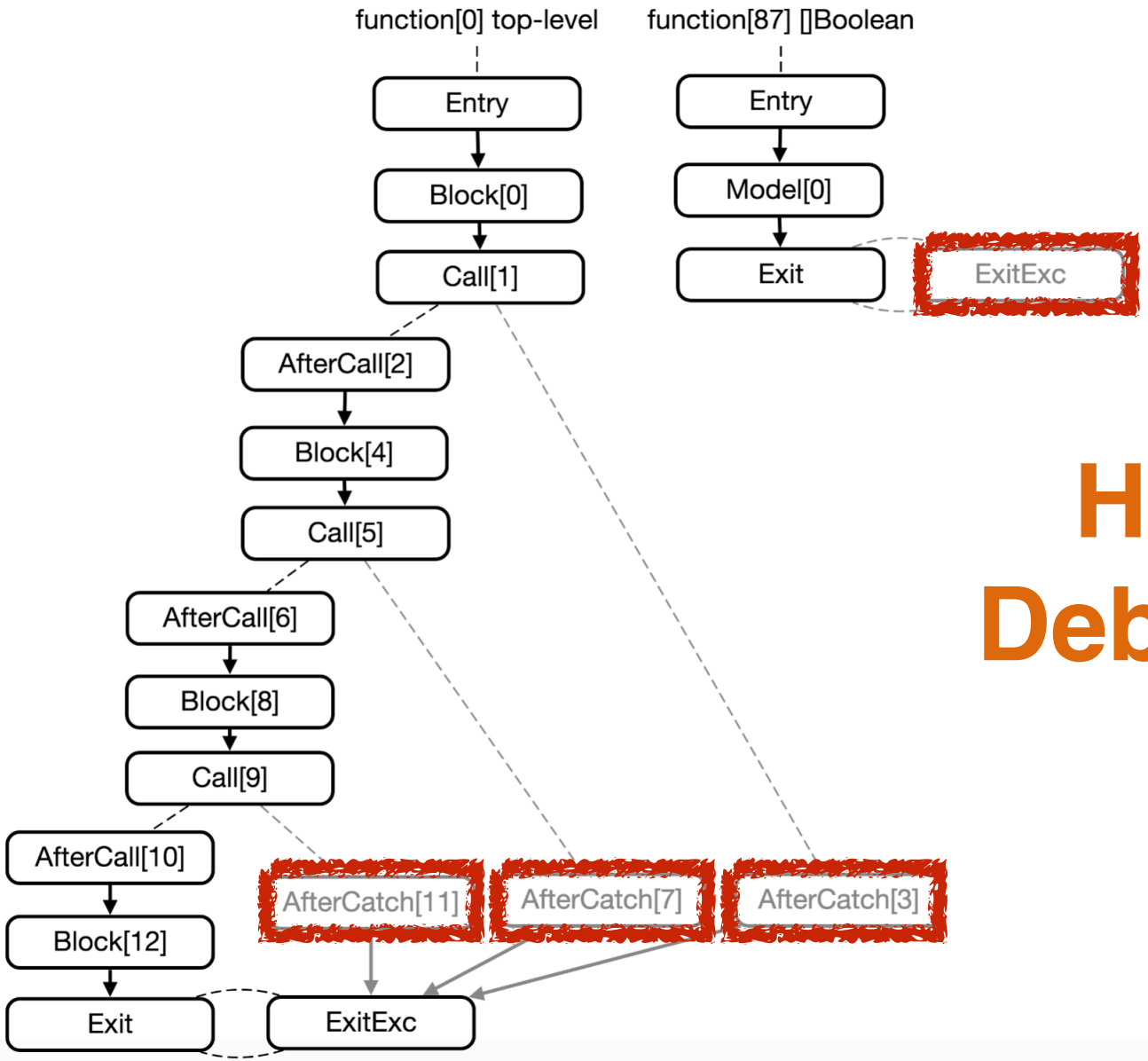


HTML Debugger

Debuggability

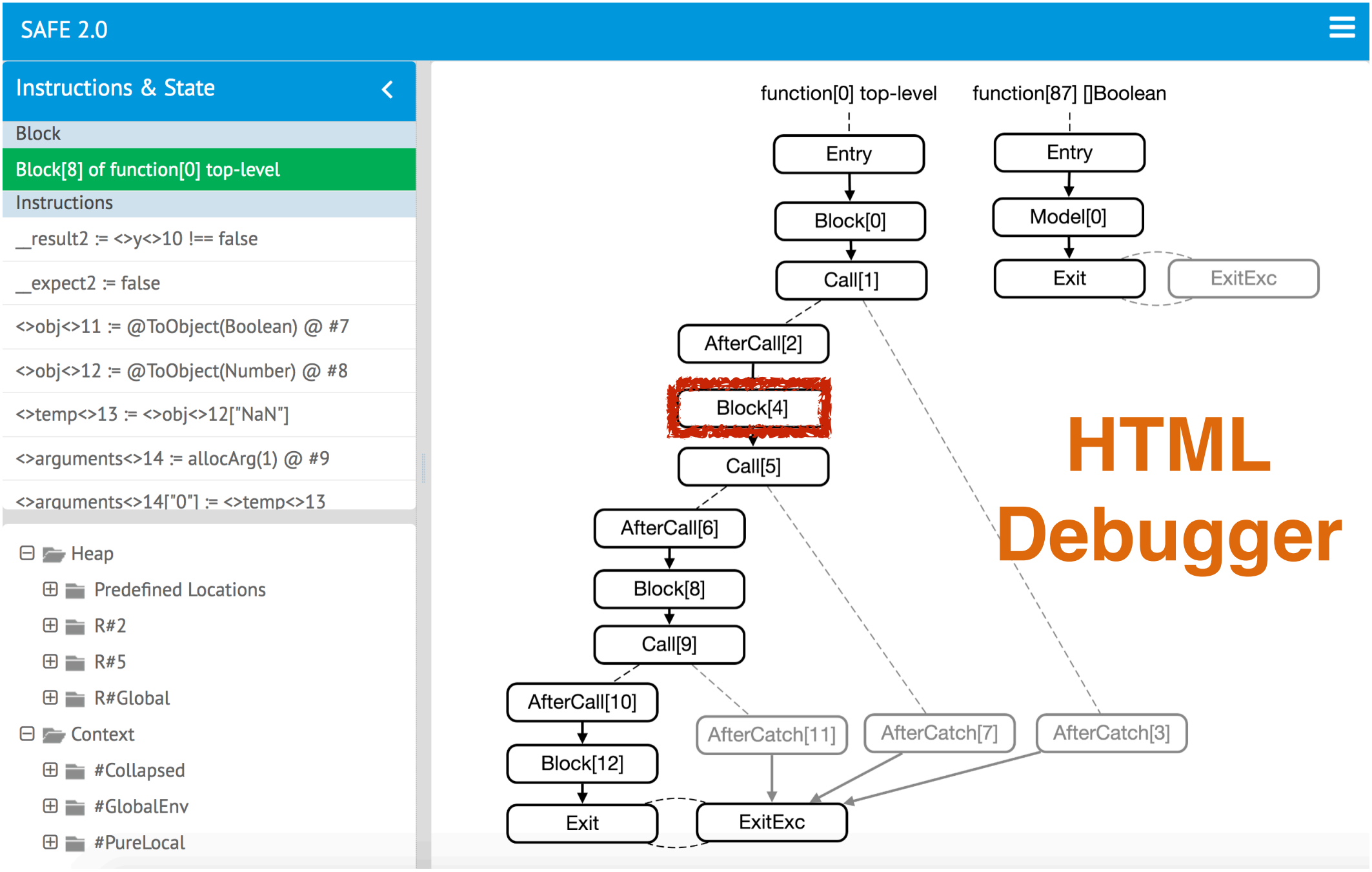
SAFE 2.0

Instructions & State

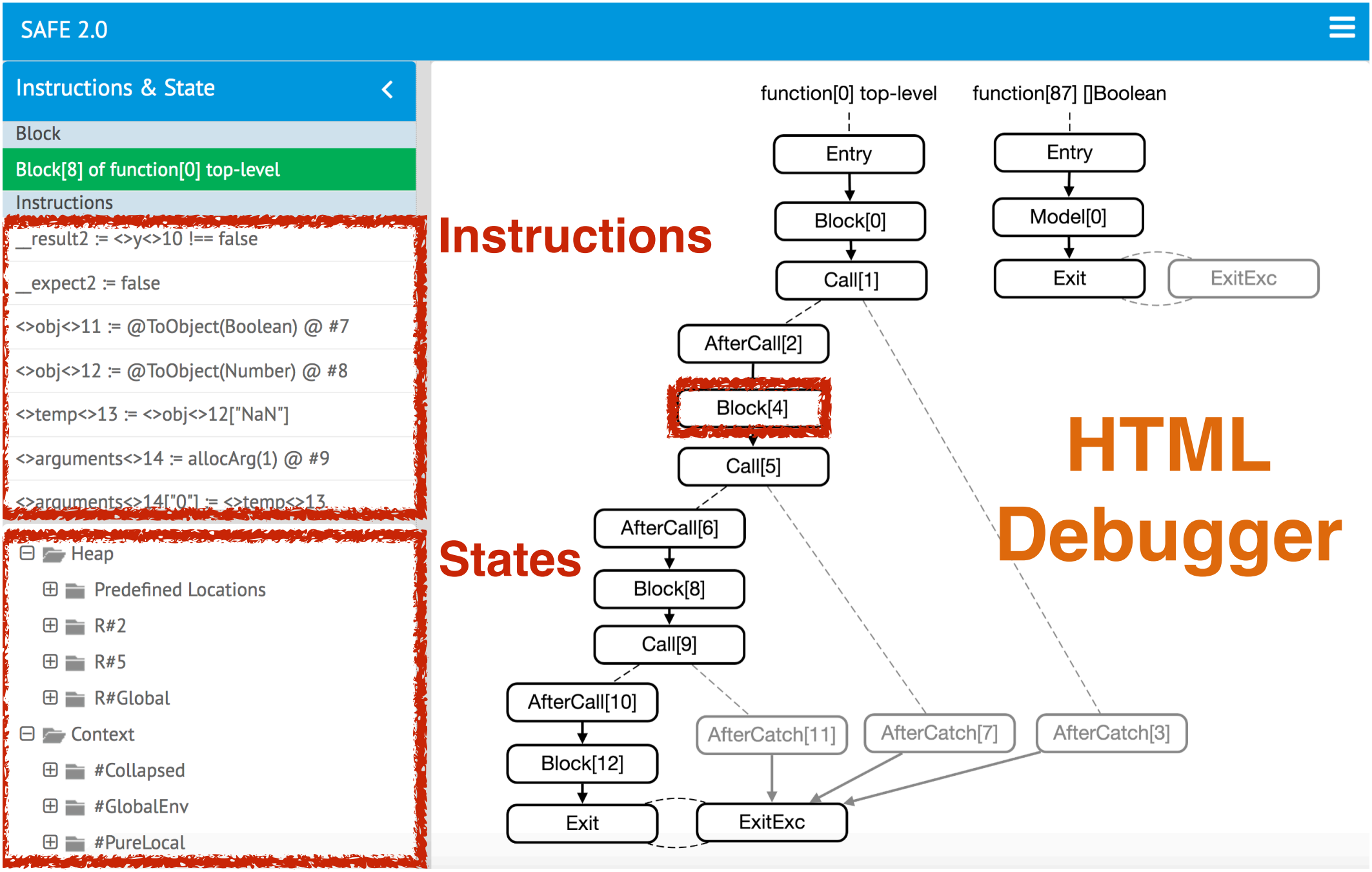


HTML Debugger

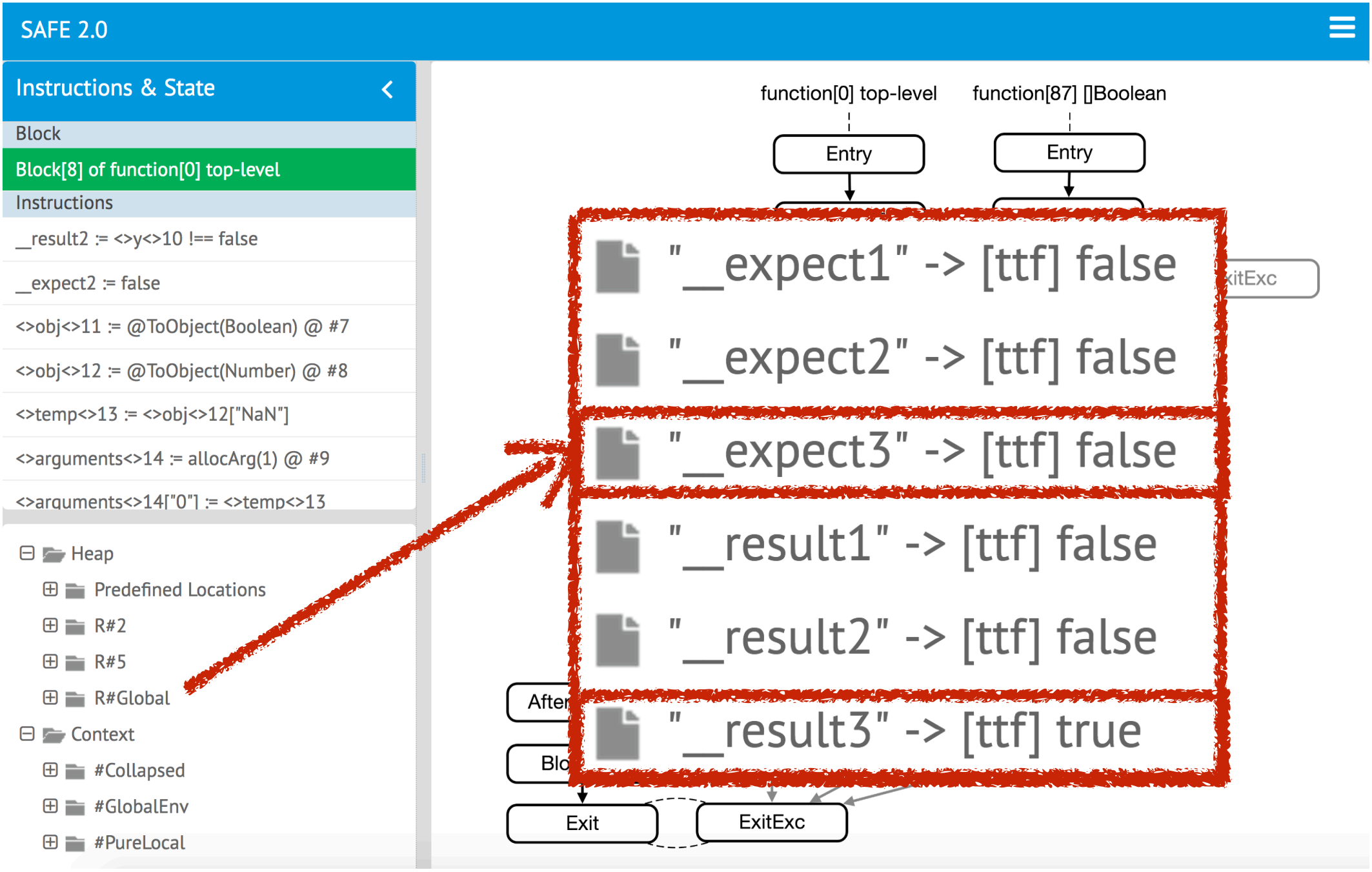
Debuggability



Debuggability



Debuggability



Debuggability

```
{
  var __result1 = Boolean(+ 0) !== false;
  var __expect1 = false;
}
{
  var __result2 = Boolean(- 0) !== false;
  var __expect2 = false;
}
{
  var __result3 = Boolean(Number.NaN) !== false;
  var __expect3 = false;
}
```

Debuggability

```
// 9.2 ToBoolean
def toAbsBoolean: AbsBool = this match {
  case Bot => AbsBool.Bot
  case Const(0.0) => AbsBool.False
  case Const(n) if n.isNaN => AbsBool.False
  case Const(_) => AbsBool.True
  case _ => AbsBool.Top
}
```

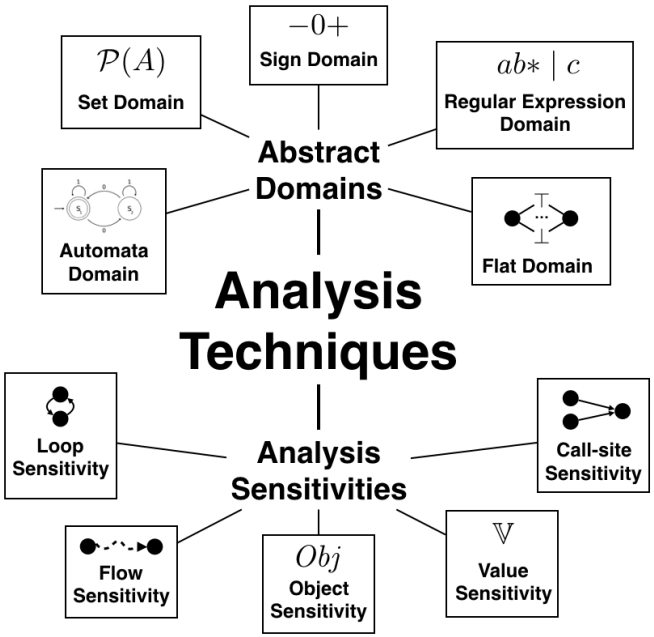
Flat Number Domain

Debuggability

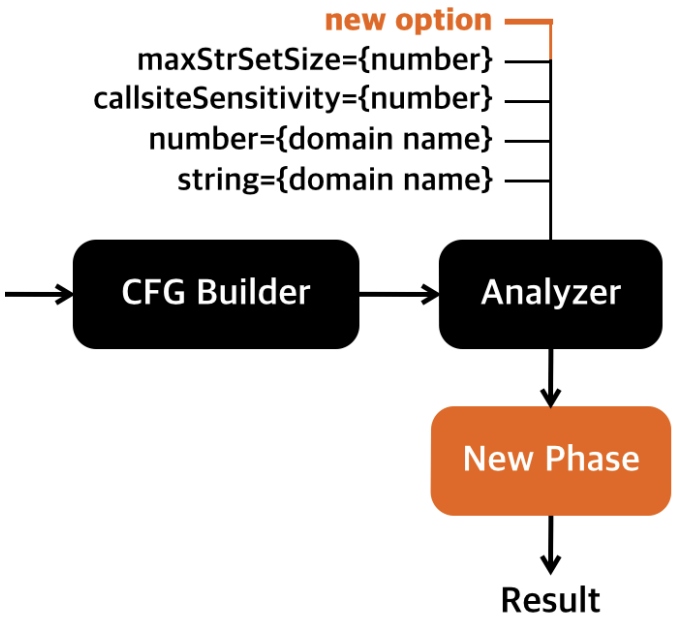
```
> test262Test
[info] Run completed in 17 minutes, 59 seconds.
[info] Total number of tests run: 5997
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 5997, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 1082 s, completed Nov 16, 2016 1:39:28 AM
```

Test262 tests

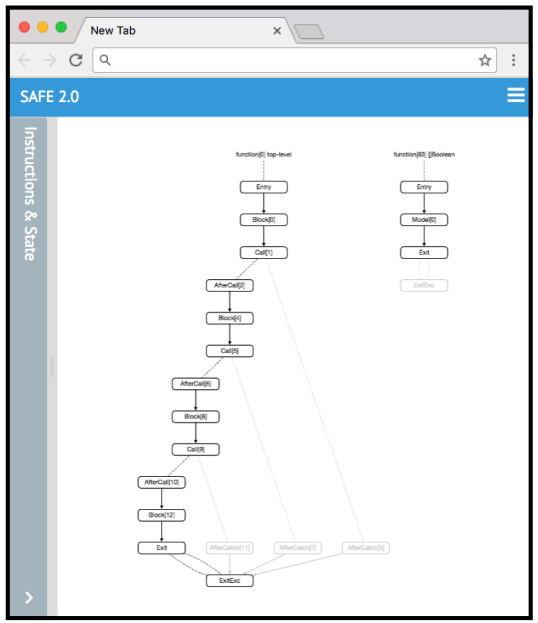
SAFE 2.0



Pluggability



Extensibility



Debuggability

https://github.com/sukyoung/safe