Real-world Challenges for JavaScript Analysis

Efficient String Domains and Beyond

Alexander Jordan, Oracle Labs Australia
Roberto Amadini, University of Melbourne

SAPLING16 – Canberra
November 21st 2016

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Oracle Labs Australia

http://labs.oracle.com/locations/australia

WAF3R – Web Application Framework for Exploring, Exposing and Eliminating Risks

Target: Enterprise Java (JEE) Applications
- Server-side: Java
- Client-side: JavaScript

ARC linkage project* with University of Melbourne

* LP140100437

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

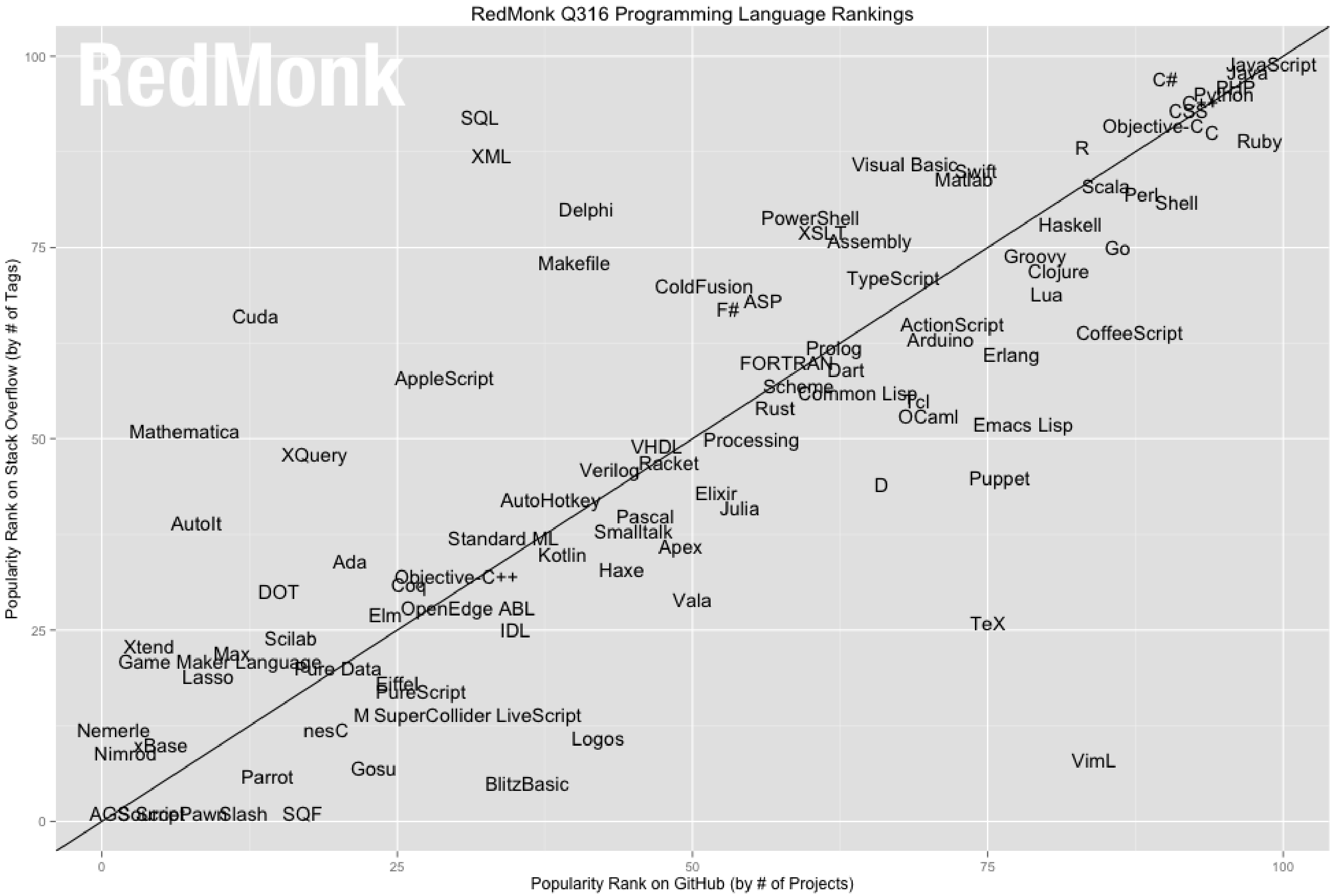
Agenda

1. Motivation
2. Challenges
3. Proposed Approach
4. In Detail: Precise Light-weight String Domains (Roberto)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Motivation: Why Bother?

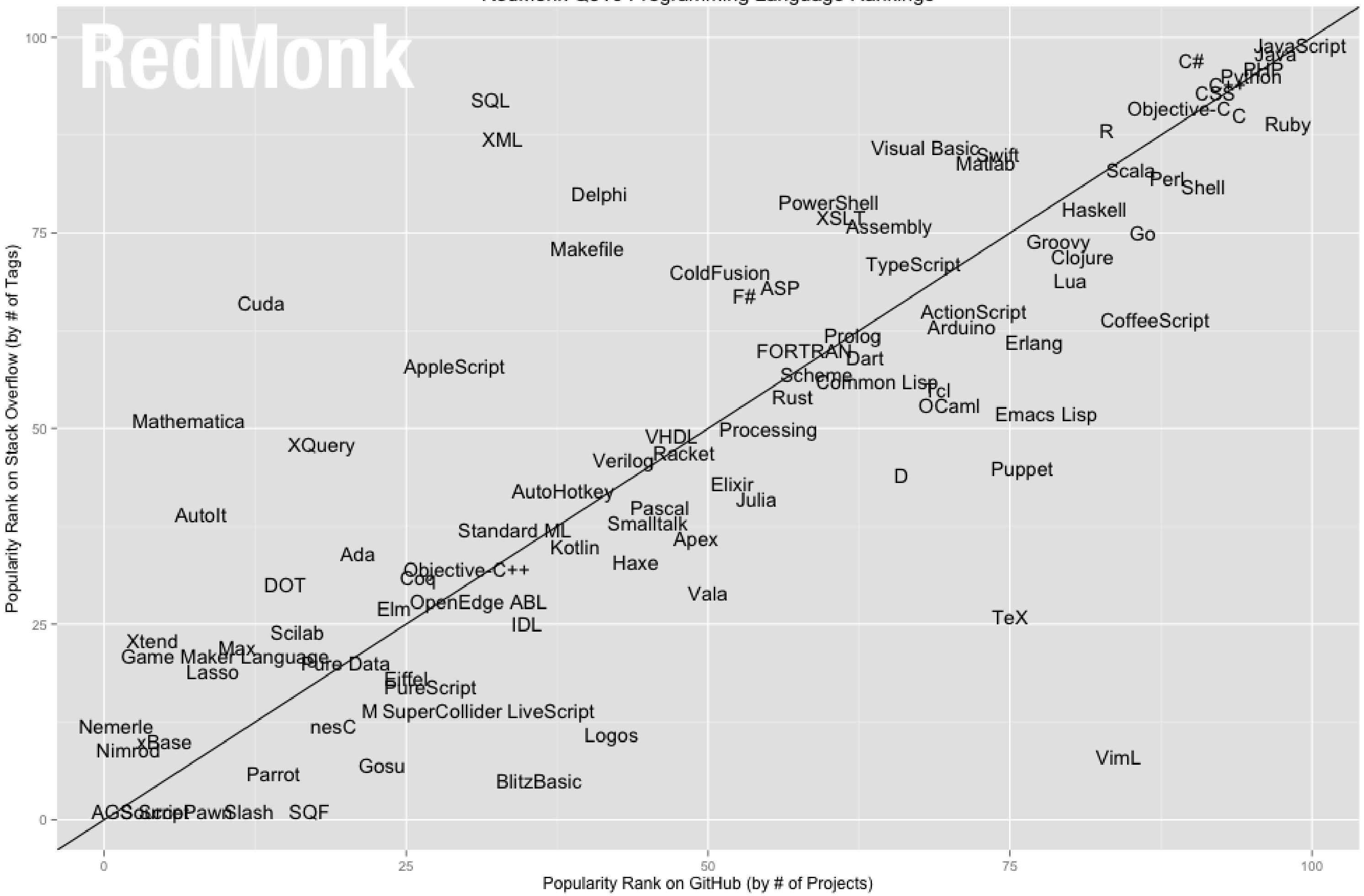


1. JavaScript
2. Java
3. PHP
4. Python
5. C#
5. C++
5. Ruby
8. CSS
9. C
10 Objective-C

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Motivation: Why Bother?

RedMonk Q316 Programming Language Rankings



- *Lingua franca of the Web*
  - Client-side (Browser)
  - Server-side (node.js)
- Mobile Apps
- IoT
- …

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Challenges: JavaScript vs. Static Analysis

A non-exhaustive list of grievances:

- Few static guarantees (un-typed & dynamic)
- Ubiquitous use of string-based (reflective) property access and object introspection
- Intricate semantics and side-effects (e.g. DOM)
- Prevalent use of libraries and frameworks (lack of a standard library)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Challenges: Precise Analysis

Precision

- High context sensitivity
- Precise domains
- Accurate models (DOM)
- Accurate event relations

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Challenges: Precise Analysis

Imprecision spreads fast

```
var o = { foo: function() { … },
          bar: function() { … } }
var p = "foo"
if (???)
  p = "bar"
  ...
var f = o[p]
  ...
var result = f()
```

Imprecise string

Imprecise prop. access

Imprecise invocation

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Approach: Precise Abstract Interpretation

- Academic tools in this space
  - SAFE (KAIST)
  - TAJS (Aarhus)

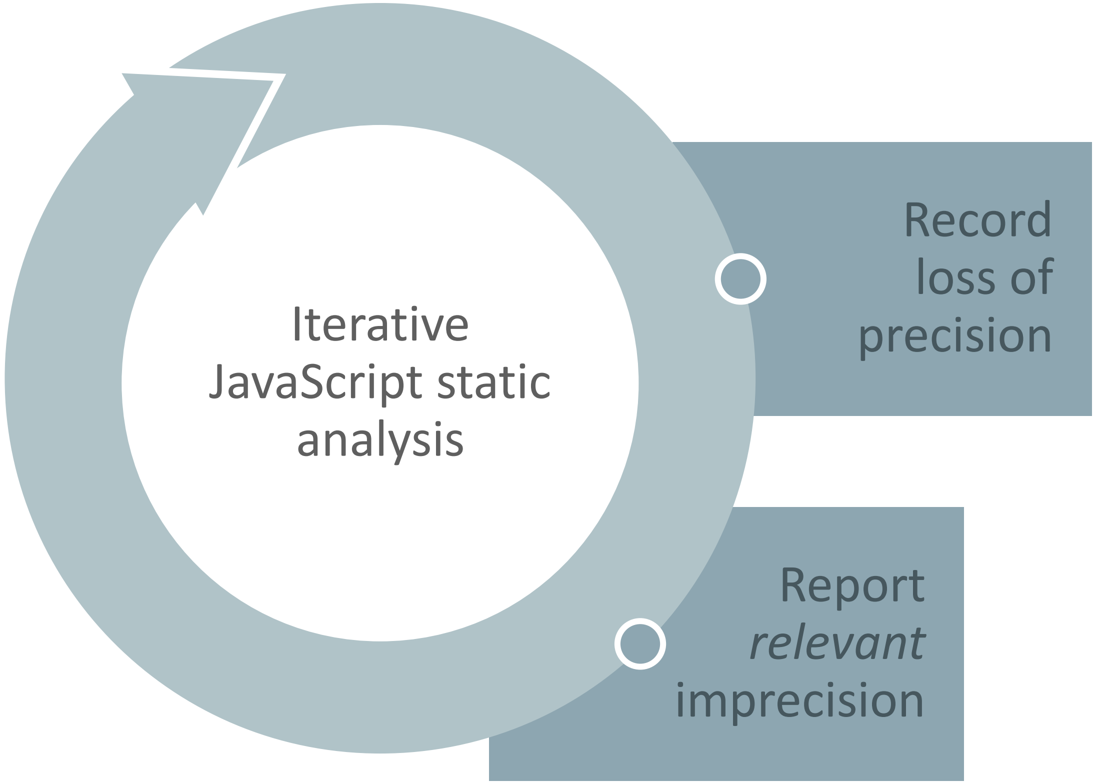
- Perceivable gap to real-world applications
- When analysis fails
- Lack of representative benchmarks

High context sensitivity	✓
Precise domains	
Accurate models (DOM)	✗
Accurate event relations	✗

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Approach: Precise Abstract Interpretation

Improving traceability of precision



ORACLE
Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Further Challenges With Web Applications

It's not just JavaScript, but how it is used…

- Event-driven applications → combinatorial analysis
- *Very* dynamic code loading
- Client-server (AJAX) communication

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Further Challenges With Web Applications

It's not just JavaScript, but how it is used…

- Event-driven applications → combinatorial analysis
- *Very* dynamic code loading
- Client-server (AJAX) communication

Possible solutions

- Hybrid analysis (combined static & dynamic) → back-and-forth?
- Hinted analysis (TypeScript, Flow) → not for legacy applications

ORACLE
Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Precise Light-weight String Domains

Over to Roberto…

ORACLE

Static Analyisis of (JavaScript) Strings

Roberto Amadini[1] Graeme Gange[1] François Gauthier[2] Alexander Jordan[2]
Peter Schachte[1] Harald Søndergaard[1] Peter Stuckey[1] Chenyi Zhang[2]

[1]The University of Melbourne

[2]Oracle Labs Brisbane

SAPLING 2016 — Canberra, ACT, Australia

Motivations

- JavaScript is highly dynamic and flexible
  - dynamic property access, `eval`, prototype-based inheritance, coercion, reflection, ...
- Precise reasoning about strings is critical for its static analysis
  - source-based analysis performed *without executing* a program
- Static analysis can detect *(absence of)* properties or undesired behaviours
  - `NaN`, `undefined + string`, SQL injections, ...
- Abstract Interpretation is a well-known theory for static analysis
  - based on approximations of concrete executions

Abstract Interpretation, informally

- Mathematical framework for static analysis introduced by Patrick Cousot in 1977
- We can't analyse all possible concrete executions of a program
  - concrete semantics not computable (loops, recursion, ...)
- We can abstract ($\equiv$ *approximate*) set of concrete values and operations and analyse such abstractions
  - computable, *but* possible precision losses and "false alarms"
- **Example:** concrete values $x = \{2, 8, 76, 100\}$ can be abstracted by abstract value $\widehat{x} = [1, 100] \supseteq x$. We introduce imprecision (values in $\widehat{x} \setminus x$) and possible false alarms (e.g., we can still say that x is positive, but no longer say that x is odd)

Abstract Interpretation and Strings

- Fixed an alphabet Σ, a string abstract domain is the set $\mathcal{S}$ of the abstract values that can approximate concrete strings $s \in \Sigma^*$
  - Formally, it should be a *lattice* $\langle \mathcal{S}, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle \ldots$
- An abstraction function $\alpha : \mathcal{P}(\Sigma^*) \to \mathcal{S}$ mapping set of concrete strings to an abstract string
  - **Example:** approximate the string value of `x` after:

```
if (???)  x = "foo"; else x = "zoo"
```

  - $\alpha_{\text{chars}}(\{\texttt{foo}, \texttt{zoo}\}) = \{\texttt{f}, \texttt{o}, \texttt{z}\} \subseteq \Sigma$
- A concretisation function $\gamma : \mathcal{S} \to \mathcal{P}(\Sigma^*)$ mapping set of an abstract string to a set of concrete strings
  - $\gamma_{\text{chars}}(\{\texttt{f}, \texttt{o}, \texttt{z}\}) = \{x \in \Sigma^* \mid \texttt{f}, \texttt{o}, \texttt{z} \in x\} \subseteq \Sigma^*$
  - over-approximation! $\texttt{zoff} \in \gamma_{\text{chars}}(\{\texttt{f}, \texttt{o}, \texttt{z}\}) \setminus \{\texttt{foo}, \texttt{zoo}\}$
  - α is not the inverse of γ, but often form a *Galois connection*...
  - γ often returns an infinite set of strings!

Abstract Interpretation and JavaScript

- In JavaScript (JS) is fundamental to approximate strings as precisely as possible
- **Example:** dynamic access to `obj[x]`, where property `x` is an unknown string. If the static analysis approximates `x` with the set of all possible JS string values, we have big loss of precision and efficiency!
  - `obj[x]` would point to any property of `obj`, and any property of its *prototype*
  - side-effects propagated in the analysis!
- Static analyisis = trade-off between precision and efficiency

Examples of String Abstract Domains

- String domains can be generic, e.g.:

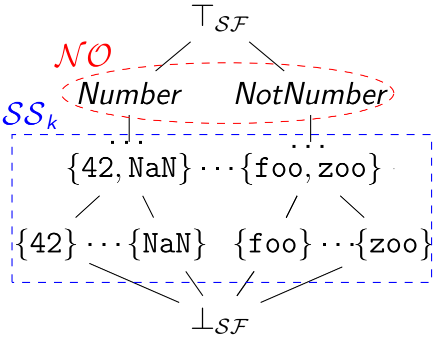
Domain	$\{\texttt{foo, zoo}\}$ **example**
Character Inclusion ($\mathcal{CI}$)	$\langle\{\texttt{o}\}, \{\texttt{f}, \texttt{o}, \texttt{z}\}\rangle$
Prefix/Suffix ($\mathcal{PS}$)	$\langle\epsilon, \texttt{oo}\rangle$
String Length ($\mathcal{SL}$)	$[3, 3]$
String Set ($\mathcal{SS}_k$)	$k = 1 \Rightarrow \top$, $k > 1 \Rightarrow \{\texttt{foo}, \texttt{zoo}\}$

- ...but also specific to JS language!
  - e.g., domains that discriminate whether a string literal represents a JS numeric expression, e.g. `"-1.7"` or `"NaN"`. In the above example, the abstract string is *NotNumber*

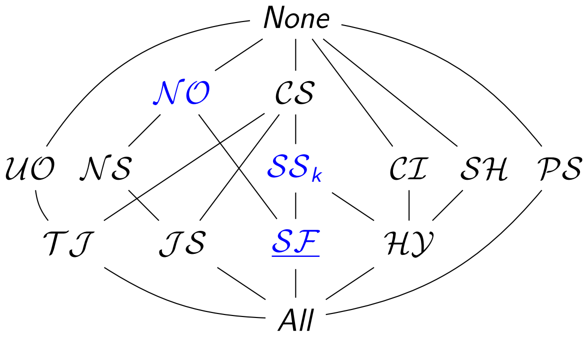
String Abstract Domains and SAFE

- We implemented 12 different string abstract domains into SAFE framework
  - SAFE is a static analyser for ECMAScript developed by KAIST University (South Korea) for the JavaScript community
- SAFE only had one string domain, that we called $\mathcal{SF}$
  - $\mathcal{SF}$ uses $\mathcal{SS}_k$ for tracking $k \geq 1$ concrete strings, if k threshold is exceeded only discriminates between *Numeric*/*NotNumber* strings ($\mathcal{NO}$ domain)
  - In the example above, with $k = 1$, the abstract string is *NotNumber*

SAFE String Abstract Domain



Overview of Implemented Domains



Combining String Abstract Domains

- Can we get more precise analysis by combining different string domains?
  - The whole should be more than the sum of the parts!
- We implemented a systematic way of combining an arbitrary collection of single domains without any implementation effort
  - Formally, this combination is called direct product and generalises the notion of Cartesian product

Empirical Evaluation

- We evaluated different domain combinations on benchmarks of JS programs
  - most of them relying on well-known jQuery library
- Main Result: while a single domain often leads to severe loss of precision, a suitable combination of domains can outperform the precision of state-of-the-art JavaScript analysers (e.g., SAFE).
  - $\mathcal{CI}$ and $\mathcal{NO}$ domains appear to be beneficial!
- A paper describing this evaluation has been submitted and currently under review at TACAS conference

Conclusions

- Static Analysis is hard, in JS is harder!
- Precise string approximation is crucial for meaningful analyisis
- Several orthogonal string abstract domains can be used
- A good strategy is to take advantage of their combination

Future Works

- Evaluate new domains (e.g., a regular domain) and benchmarks
- From direct product to reduced product of domains
  - informally, a "refinement" that removes redundant combinations
- Integrate our implementation into SAFE 2.0
- ...

Finally ...

... done!

Questions?