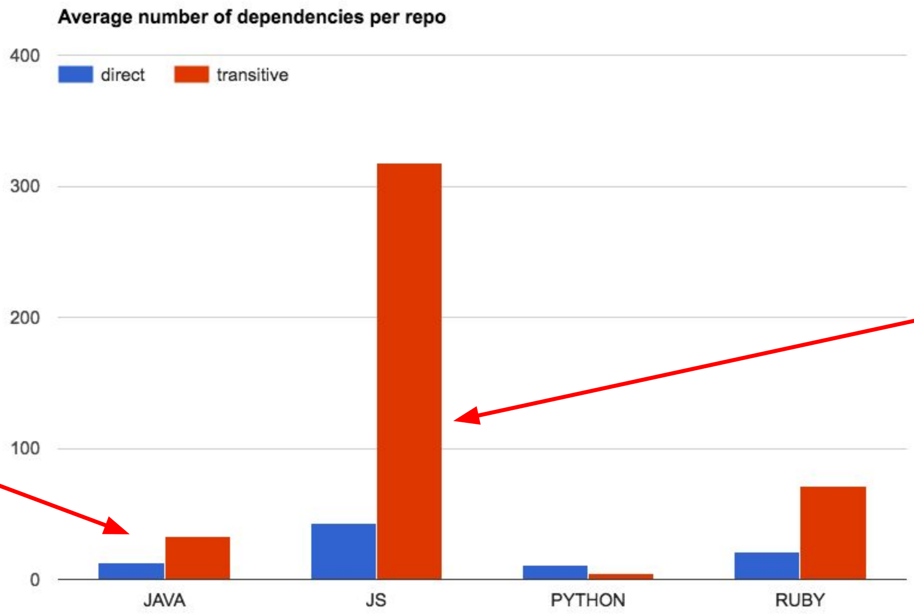
# Efficient Static Checking of Library Updates

Darius Foo, Hendy Chua,
Jason Yeo, Ang Ming Yi,
Asankhaya Sharma

# Motivation

- Applications depend on lots of libraries

For each Java library depended on, 4 others are added




For each JS library depended on,
9 others are added

# Motivation

- Libraries evolve, and we’d like to keep up
  - Security patches
  - Bug fixes
  - New features

# Motivation

- Upgrades are hard!
  - Compile errors
  - API incompatibilities
  - Test failures
  - Dependency conflicts
  - Crashes at runtime
  - Subtle changes in behavior

# Motivation

- Semantic versioning (SemVer)
    - Adherence to conventions; MAJOR.MINOR.PATCH
        - Backwards-incompatible change: bump MAJOR
        - Backwards-compatible addition: bump MINOR
        - Backwards-compatible bug fix: bump PATCH
    - Structured; tooling-friendly
        - ~> operator
    - Inadequately able to capture nuances of change
    - Compliance of source code to scheme must be manually enforced

# What we want

- Automated, safe library upgrades
- Automated pull requests, but with guarantees
    - Can Automated Pull Requests Encourage Software Developers to Upgrade Out-of-Date Dependencies? Mirhosseini, et al.
    - 60% increase in frequency of upgrades
    - Notification fatigue and concerns about breaking changes became bottleneck thereafter
- Fast enough to run in a CI pipeline

# Approach

- Static analysis for detecting API incompatibilities in upgrades
- Compute differences between source-level elements
  - Methods, functions
- Take control flow into account
- Determine if code to be upgraded is calling a changed/deleted method
- Precompute diffs and compose them on request

# Related work

- Automated library upgrades
  - Deppbot, Greenkeeper
    - Update all dependencies within constraints and rely on test suites
  - SemDiff (Dagenais, et al.), Diff-CatchUp (Xing, et al.)
    - Recommend replacements for changed methods by looking at how libraries adapt to their own changes
  - CatchUp! (Henkel, et al.)
    - Capture refactoring actions on an API, replay them on uses of an API

# Related work

- Structured diffs
  - Textual, subsequence-based diffs (*diff*)
    - Computed quickly, but without considering syntax
  - Syntactic diffs
    - Computed between syntactic elements
    - Google's Android API diffs
    - UMLDiff, Gumtree
  - Semantic diffs
    - Reflects control flow, state
    - Semantic Diff: computes differences in input-output behaviour of functions
    - SymDiff: partial equivalence between programs

# Related work

- SemVer compliance
  - Semantic Versioning versus Breaking Changes: A Study of the Maven Repository, Raemaekers, et al.
  - Similar studies for npm and CRAN

# Approach

- Basic API diffs
  - Extract tuples of method name and bytecode hash
  - Hashes approximate method implementations and detect changes
  - Libraries are sets of methods

| Method name | Hash |
| --- | --- |
| com.example.A.a([B)I | 0xCAFEBEEF |

# Approach

- Basic API diffs
  - Given two library signatures, use Myers' algorithm to compute a diff
  - Three operations: INSERT, CHANGE, DELETE
  - Drop non-public methods to get the API diff

| **Method name** | **Operation** |
|---|---|
| com.example.A.a([B)I | DELETE |

# Approach

- Basic API diffs

```
class A {
  public int a() {
    return 2;
  }
  public int b(int x) {
    return x + 3;
  }
}
```

| Method | Operation |
|---|---|
| A.a()I | DELETE |
| A.b(I)I | CHANGE |
| A.c()I | INSERT |

```
class A {
  // Method a deleted
  public int b(int x) {
    return x + 2; // Modified
  }
  public int c() { // Inserted
    return 1;
  }
}
```

# Approach

- Transitively-changed methods
  - Dropping private methods is problematic...
    - (Public) $m_1$ is unchanged, but calls (private) $m_2$, which has changed
    - Changes to $m_1$ are lost after dropping $m_2$

```
class A {
  public int m1(int x) {
    return m2(x);
  }
  private int m2(int y) {
    return y + 1;
  }
}
```

```
class A {
  public int m1(int x) {
    return m2(x);
  }
  private int m2(int y) {
    return y + 2; // Changed
  }
}
```

# Approach

- Transitively-changed methods
  - Build call graphs and use them to improve diffs
  - Transitive callers of changed/deleted methods must also have changed
  - Private methods may still be dropped, but we no longer lose changes

# Approach

- Fast queries
  - How to compute diffs on demand?
  - Call graph construction is expensive
    - Hours for largest libraries on Maven Central
  - Precompute and store every pair of diffs?
    - $O(v^2)$ space
    - Real-world libraries have *hundreds* of versions
  - Do this for a subset of libraries?
    - How to determine this subset?
    - Does not work outside it

Central (969)

| Version | Scala | Repository |
|---|---|---|
| 1.3.2-4-5b2f1ed | 2.12<br>2.11 | Central |
| 1.3.2-2-3da4986 | 2.12<br>2.11 | Central |
| 1.3.2 | 2.12<br>2.11 | Central |
| 1.3.1-0-f283950 | 2.12<br>2.11 | Central |

# Approach

- Diff composition
  - Precompute diffs only between consecutive pairs of library versions
  - Compose diffs to derive diffs for arbitrary version ranges
  - Linear space, linear time

$$\underbrace{\underbrace{v_1 \qquad\qquad\qquad v_2}_{d_1} \qquad \underbrace{\qquad\qquad\qquad v_3}_{d_2}}_{d_1 \circ d_2}$$

# Approach

- Diff composition
  - DELETE ◦ INSERT = ?

# Approach

- Diff composition
  - DELETE ∘ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different

# Approach

- Diff composition
  - DELETE ◦ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different
  - CHANGE ◦ CHANGE = ?

# Approach

- Diff composition
  - DELETE ∘ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different
  - CHANGE ∘ CHANGE = ?
    - CHANGE: can’t say any more

# Approach

- Diff composition
    - DELETE ◦ INSERT = ?
        - CHANGE: assume conservatively that reinsertion is different
    - CHANGE ◦ CHANGE = ?
        - CHANGE: can’t say any more
    - INSERT ◦ DELETE = ?

# Approach

- Diff composition
  - DELETE ◦ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different
  - CHANGE ◦ CHANGE = ?
    - CHANGE: can’t say any more
  - INSERT ◦ DELETE = ?
    - MISSING (not yet defined)

# Approach

- Diff composition
  - DELETE ◦ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different
  - CHANGE ◦ CHANGE = ?
    - CHANGE: can’t say any more
  - INSERT ◦ DELETE = ?
    - MISSING (not yet defined)
  - DELETE ◦ DELETE = ?
  - INSERT ◦ INSERT = ?

# Approach

- Diff composition
  - DELETE ◦ INSERT = ?
    - CHANGE: assume conservatively that reinsertion is different
  - CHANGE ◦ CHANGE = ?
    - CHANGE: can’t say any more
  - INSERT ◦ DELETE = ?
    - MISSING (not yet defined)
  - DELETE ◦ DELETE = ?
  - INSERT ◦ INSERT = ?
    - Make no sense
    - Composition is partial

# Approach

- Diff composition
  - Five operations:
    - CHANGE, INSERT, DELETE
    - UNCHANGED: when a method remains the same in a diff
    - MISSING: when a method is missing from a diff altogether
  - UNCHANGED and MISSING are never produced when diffs are computed, only during composition
  - We include (and distinguish) them because composition is partial
    - e.g. INSERT requires that a method be absent before, and present after
    - `INSERT :: Absent → Present`

# Approach

- Diff composition
  - `INSERT :: Absent → Present`
  - `CHANGE :: Present → Present`
  - `DELETE :: Present → Absent`
  - `UNCHANGED :: Present → Present`
  - `MISSING :: Absent → Absent`

# Approach

- Diff composition
  - These 'types' tell us that the composition function on diffs has this type:

    `compose :: (a → b) → (b → c) → (a → c)`

  - i.e. consecutive diff operations between versions $v_1$, $v_2$, $v_3$, must agree on the state of $v_2$
  - compose is uniquely defined on many inputs
    - `compose DELETE MISSING` *must be* `DELETE`

# Approach

- Diff composition
  - Ambiguity only arises when selecting between CHANGE and UNCHANGED
    - We've not modelled hashes
    - We pick CHANGE conservatively where required

```
  CHANGE :: Present → Present
UNCHANGED :: Present → Present
```

# Approach

- Diff composition
  - compose is associative, but not symmetric:
    - `compose INSERT DELETE = MISSING`
    - `compose DELETE INSERT = CHANGE`

| | **I** | **C** | **D** | **U** | **M** |
|---|---|---|---|---|---|
| **I** | ⊥ | I | M | I | ⊥ |
| **C** | ⊥ | C | D | C | ⊥ |
| **D** | C | ⊥ | ⊥ | ⊥ | D |
| **U** | ⊥ | C | D | U | ⊥ |
| **M** | I | ⊥ | ⊥ | ⊥ | M |

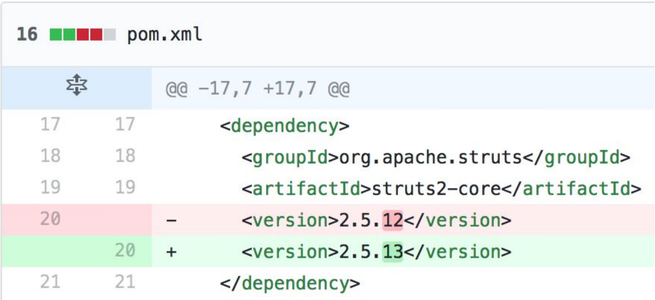
# Approach

- Conflating operations
  - We can conflate UNCHANGED and MISSING into a single operation UNKNOWN, because they occur in mutually exclusive scenarios
  - Useful because we implicitly represent them in practice, e.g. when an item is absent
  - Does not change composition semantics

| | **I** | **C** | **D** | **UM** |
|---|---|---|---|---|
| **I** | ⊥ | I | UM | I |
| **C** | ⊥ | C | D | C |
| **D** | C | ⊥ | ⊥ | D |
| **UM** | I | C | D | UM |

# Approach

- Suggesting upgrades
  - Software composition analysis
  - Given a library with a range of versions,
    - Pick a version which succeeds the current and has no known vulns
    - Compute diff
    - Check if any missing/changed methods are called
    - Make a pull request



# Demo

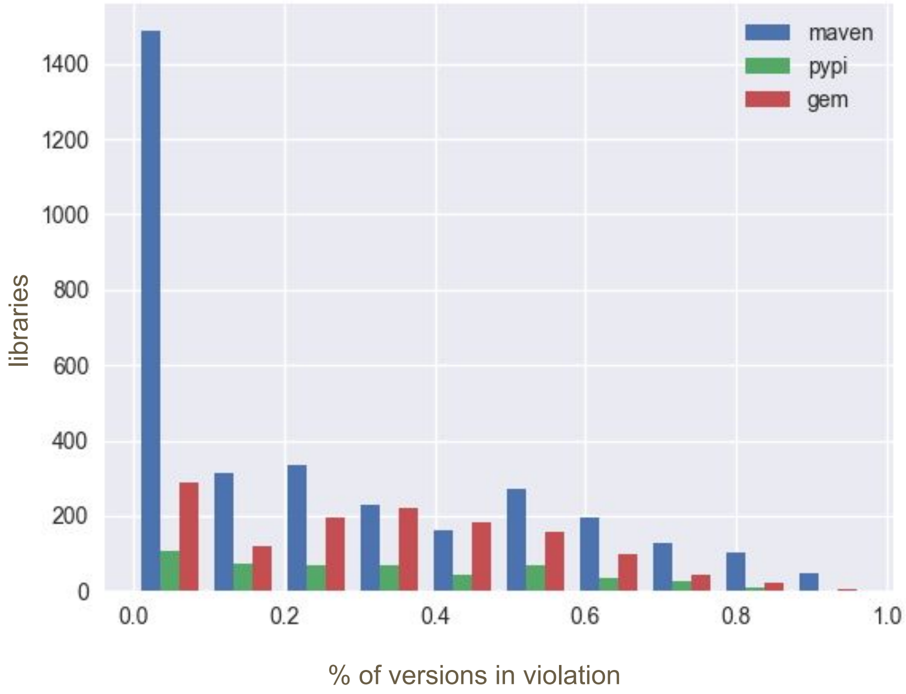
- Update Advisor in SourceClear

# Experiments and Evaluation

- SemVer compliance
  - Computed diffs for 114,199 versions across 5,106 libraries from Maven Central, RubyGems, and PyPI
  - 72% of libraries violate SemVer in *some* version
    - RubyGems: 80%
    - Maven Central: 67%
    - PyPI: 82%

# Experiments and Evaluation

- SemVer compliance
  - 26% of library versions violate SemVer
    - Maven Central: 24%
      - Raemaekers, et al.
      - 28.4% to 23.7% over time
    - PyPI: 31%
    - RubyGems: 31%



# Experiments and Evaluation

- SemVer compliance
  - Concrete example: `requests`
  - Between 2.3.0 and 2.4.0, `requests.structures.IteratorProxy` was deleted

# Experiments and Evaluation

- SemVer compliance
  - Concrete example: `requests`
  - Between 2.3.0 and 2.4.0, `requests.structures.IteratorProxy` was deleted
  - Difficult to determine if it was part of public API
    - Python has no access modifiers, only special handling of _ prefix
    - Checked changelogs, commits, documentation

# Experiments and Evaluation

- API incompatibilities in open source projects
  - Attempted to perform upgrades automatically on open source projects
  - On average, 10% of upgrades were non-breaking

| | Java | Python | Ruby |
|---|---|---|---|
| **Projects** | 274 | 422 | 503 |
| **Direct dependencies** | 4777 | 2572 | 4096 |
| **Direct vulnerable** | 246 | 110 | 250 |
| **Suggested upgrades** | 150 | 64 | 123 |
| **Non-breaking** | 28 (19%) | 0 (0%) | 7 (6%) |

# Threats to validity

- Limitations of static analysis (FP)
  - Call graphs overapproximate dynamic control flow
  - Hashing to detect changes
- Unsupported language features (FN)
- Computing library diffs in isolation (FN)
  - Cannot pick up breaking changes due to calls to methods in transitive upgrades
- Insufficient semantic information (FP)
  - `requests` example; must guess if upgrade is really breaking
- Binaries compiled for different platforms (FP)
  - .NET, Java 9

# Future work

- Improve false positive and false negative rates
  - Augment static call graphs with dynamic call graphs
  - More sophisticated change detection than hashing
- Upgrade transitive dependencies
  - Find a direct upgrade that performs a transitive upgrade
  - Find the fewest such upgrades
- Handle dependency conflicts
- Suggest better upgrades
  - Constraints, e.g. does not cross a major version
  - Weigh breaking changes vs severity of vulns fixed
- Infer API usage and suggest replacements

# Thank you!

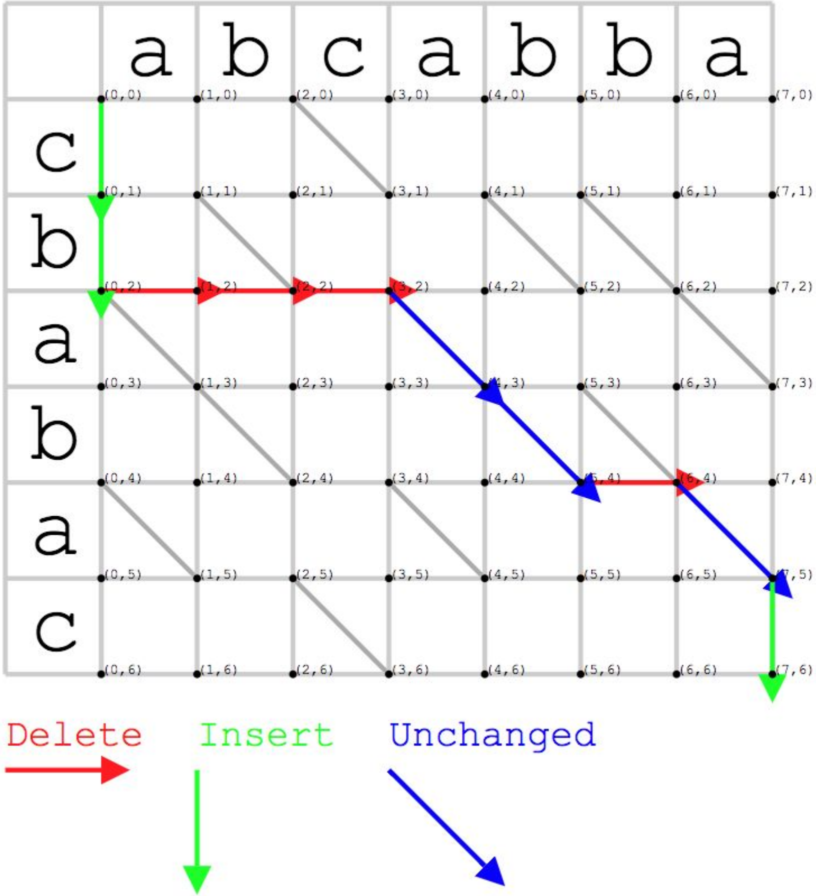
Q&A

# Try it out

- www.sourceclear.com
- Free trial
- `SRCCLR_ENABLE_PR=true SRCCLR_PR_ON=low SRCCLR_IGNORE_CLOSED_PRS=true srcclr scan --url https://github.com/srcclr/example-java-maven --gen-pr`

# Approach

- Myers' algorithm




Credit: http://blog.robertelder.org/diff-algorithm/