Graph-based analysis of JavaScript source code repositories

Gábor Szárnyas

Graph Processing devroom @ FOSDEM 2018
JAVASCRIPT

Most Popular Technologies

Programming Languages

<table>
<thead>
<tr>
<th>% of This Category</th>
<th>% of All Respondents</th>
<th>% of Professional Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td></td>
<td>62.5%</td>
</tr>
<tr>
<td>SQL</td>
<td></td>
<td>51.2%</td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td>39.7%</td>
</tr>
<tr>
<td>C#</td>
<td></td>
<td>34.1%</td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td>32.0%</td>
</tr>
</tbody>
</table>

Latest standard: ECMAScript 2017
**STATIC ANALYSIS**

- *Static source code analysis* is a software testing approach performed without compiling and executing the program itself.
STATIC ANALYSIS TOOLS

- C
  - lint -> *linters*

- Java
  - FindBugs
  - PMD

- JavaScript
  - ESLint
  - Facebook Flow
  - Tern.js
  - TAJS

1. NO GLOBAL RULES OR
2. DIFFICULT TO EXTEND
PERFORMANCE CONSIDERATIONS

- Checking global rules is computationally expensive
- Slow for large projects, difficult to integrate even to CI

- Workaround #1: no global rules (ESLint)
- Workaround #2: batching (e.g. 1/day)
- Workaround #3: custom algorithms (e.g. Flow)
PROJECT GOALS

Goal
- Static analysis for JavaScript applications

Design considerations
- Custom analysis rules
  - Both global and local
  - Extensible
- High-performance
  - “real-time” responses
ARCHITECTURE AND WORKFLOW
PROPOSED APPROACH

Design considerations
- Custom analysis rules
- High-performance

Approach
- Use a declarative query language
- Use incremental processing
  - in lieu of batch execution
  - file-granularity
  - maintain results
ARCHITECTURE

VCS
Main.js
Dependency.js
Fiterator.js
Parser.js

Workspace
- discoverer
  - ChangeProcessor.js
  - CommandParser.js
  - FileIterator.js
  - iterators
  - DepCollector.js
  - FileDiscoverer.js
  - InitIterator.js
  - Main.js
  - whitepages
    - ConnectionMgr.js
    - DependencyMgr.js

Abstract Syntax Tree

Abstract Semantic Graph

Analysis rules
<!>
<?>
Validation report
<>

Analysis server

Graph database

Client
a sequence of statements:

\[
\text{var foo = 1 / 0}
\]
**CODE PROCESSING STEPS**

- **tokenizer**
- **parser**
- **scope analyzer**

**TOKENS**

- **tokens**: the shortest meaningful character sequence

```plaintext
var foo = 1 / 0
```

<table>
<thead>
<tr>
<th>Token</th>
<th>Token type</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>VAR (Keyword)</td>
</tr>
<tr>
<td>foo</td>
<td>IDENTIFIER (Ident)</td>
</tr>
<tr>
<td>=</td>
<td>ASSIGN (Punctuator)</td>
</tr>
<tr>
<td>1</td>
<td>NUMBER (NumericLiteral)</td>
</tr>
<tr>
<td>/</td>
<td>DIV (Punctuator)</td>
</tr>
<tr>
<td>0</td>
<td>NUMBER (NumericLiteral)</td>
</tr>
</tbody>
</table>
Abstract Syntax Tree

- Tree representation of
- the grammar structure of
- sequence of tokens.

CODE PROCESSING STEPS

**AST**

Module

VariableDeclarationStatement

VariableDeclaration

VariableDeclarator

BindingIdentifier

name = "foo"

BinaryExpression

operator = "Div"

LiteralNumericExpression

value = 1.0

LiteralNumericExpression

value = 0.0
CODE PROCESSING STEPS

Abstract Semantic Graph

- Not necessarily a tree
- Has scopes & semantic info
- Cross edges
AST VS. ASG

```javascript
var foo = 1 / 0
```

1 LOC -> 20+ nodes
PATTERN MATCHING

- Declarative graph patterns with Cypher

MATCH (binding:BindingIdentifier) <-[:binding]-(())-->
  (be:BinaryExpression)
  -[:right]-(right:LNExpression)
WHERE be.operator = 'Div'
  AND right.value = 0.0
RETURN binding
function foo(x, y) {
    return (x + y);
}
function bar(a, b) {
    return foo(b, a);
}
var quux = bar("goodbye", "hello");

Source: http://marijnhaverbeke.nl/blog/tern.html
Reachability:

- dead code detection
- async/await (ECMAScript 2017)
- potential division by zero
TECH DETAILS
IMPORTS AND EXPORTS

MATCH

// exporter.js: export { name1 as exportedName1 }

(exporter:CompilationUnit)-[:contains]->(:ExportLocals)-[:namedExports]->(:ExportLocalSpecifier:ExportLocalSpecifier)
  -[:name]->(:IdentifierExpression)-[:node]->(:Reference)-[:references]->(:Variable)
  -[:declarations]->(:declarationToMerge:Declaration)-[:node]->(:BindingIdentifier),

// importer.js: import { exportedName1 } from "exporter"

(importer:CompilationUnit)-[:contains]->(:import:Import)-[:namedImports]->(:ImportSpecifier)
  -[:binding]->(:importBindingIdentifierToMerge:BindingIdentifier)-[:node]->(:declarationToDelete:Declaration)
  -[:declarations]->(:importedVariable:Variable)

WHERE exporter.parsedFilePath CONTAINS import.moduleSpecifier

AND exportLocalSpecifier.exportedName = importBindingIdentifierToMerge.name

MERGE

(importedVariable)-[:declarations]->(:declarationToMerge)-[:node]->(importBindingIdentifierToMerge)

DETACH DELETE

declarationToDelete
FIXPOINT ALGORITHMS

- Lots of propagation algorithms
- „Run to completion” scheduling
  - Mix of Java code and Cypher
EFFICIENT INITIALIZATION

- Initial build of the graph with Cypher was slow
- Generate CSV and bulk load
- Two files: nodes, relationships

```bash
$NEO4J_HOME/bin/neo4j-admin import
   --database=db
   --nodes=nodes.csv
   --relationships=relationships.csv
```

- 10× speedup
REGULAR PATH QUERIES

- Transitive closure on certain combinations

- Workaround:
  - Start transaction
  - Add proxy relationships
  - Calculate transitive closure
  - Rollback transaction

- openCypher proposal for path patterns
  
  \( (:A)-/[:R1 :R2 :R3]+/-\rightarrow(:B) \)
INCREMENTAL QUERIES
The openCypher project aims to deliver a full and open specification of the industry’s most widely adopted graph database query language: Cypher.” (late 2015)

Research prototypes

- Graphflow (University of Waterloo)
- ingraph (incremental graph engine)
Mapping openCypher to relational algebra

Combining and filtering pattern matches

MATCH <p>
\{\text{edges of } p \ (P)\}

MATCH <p1>, <p2>
\{\text{edges of } p1 \text{ and } p2 \ (P1 \land P2)\}

MATCH <p1>, <p2>
\{\text{edges of } p1 \ (P1) \land \text{edges of } p2 \ (P2)\}

MATCH <p1>
\{\text{edges of } p1 \ (P1) \land \text{edges of } p2 \ (P2)\}

MATCH <p1>
OPTIONAL MATCH <p2>
\{\text{edges of } p1 \ (P1) \land \text{edges of } p2 \ (P2)\}

MATCH <p>
WHERE <condition>
\{\text{condition}(r), \text{where condition may specify patterns and arithmetic constraints on existing variables}\}

Result and sub-result operations. Rules for \textbf{RETURN} also apply to \textbf{WITH}.

\textbf{RETURN} <variables> \Rightarrow \pi_{\text{variables}}(r)

\textbf{RETURN} \ <v1> \ AS \ <alias1> \ ... \ \pi_{v1\ldots\text{alias1}\ldots}(r)

\textbf{RETURN DISTINCT} <variables> \Rightarrow \delta_{\pi_{\text{variables}}(r)}(r)

\textbf{RETURN} <variables>, <aggregates> \Rightarrow \gamma_{\text{variables}, \text{aggregates}}(r)

List operations

\textbf{ORDER BY} <v1> \ [\text{ASC|DESC}] \ ... \ \tau_{1/\ldots}(r)

\textbf{LIMIT} \ <1> \ \delta_{1}(r)
STATE OF INGRAPH IN 2018

- Cover a substantial fragment of openCypher
  - MATCH, OPTIONAL MATCH, WHERE
  - WITH, functions, aggregations
  - CREATE, DELETE

- Features on the roadmap
  - MERGE, REMOVE, SET
  - List comprehensions


G. Szárnyas: *Incremental View Maintenance for Property Graph Queries*, SIGMOD SRC, 2018
RELATED PROJECTS
Code comprehension: software to graph

Dirk Mahler,
*Pushing the evolution of software analytics with graph technology*,
Neo4j blog, 2017
SLIZAA

slizaa uses Neo4j/jQAssistant and provides a front end with a bunch of specific tools and viewers to provide an easy-to-use in-depth insight of your software's architecture.

Gerd Wütherich,  
*Core concepts, slizaa*
SLIZAA: ECLIPSE IDE
SLIZAA: XTEXT OPENCYPHERER

- Xtext-based grammar
- Used in the ingraph compiler
- Now has a scope analyzer
- Works in the Eclipse IDE and web UI
WRAPPING UP

Soma Lucz: **Static analysis algorithms for JavaScript**, Bachelor’s thesis, 2017
CONCLUSION

- Some interesting analysis rules require a global view of the code
- Good use case for graph databases
  - Property graph
  - Cypher language
- Very good use case for incremental queries
  - Incrementality on multiple levels
RELATED RESOURCES

Codemodel-Rifle  
github.com/ftsrg/codemodel-rifle

ingraph engine  
github.com/ftsrg/ingraph

Shape Security’s Shift parser  
github.com/shapesecurity/shift-java

Slizaa openCypher Xtext  
github.com/slizaa/slizaa-opencypher-xtext

Thanks to Ádám Lippai, Soma Lucz, Dániel Stein, Dávid Honfi and the ingraph team.
Ω
VISUAL STUDIO CODE INTEGRATION

- Language Server Protocol (LSP) allows portable implementation

```javascript
function jpegParsingError(code) {
    return {
        type: 'JpegParsingError',
        code: code
    };
}

This function is never used or not accessible. (parameter) len: any

function buf_ensure(bytes, ptr, len) {
    if (!buf_has(bytes, ptr, len)) throw jpegParsingError('RangeOutOfBounds');
}

function buf_has(bytes, ptr, len) {
    return ptr + len < bytes.length;
}
```
USE CASES

- Control Flow Graph
  - graph representation of
  - every possible statement sequence
- Basis for type inferencing and test generation