Efficient Code Fixes Using Reference Attributed Grammars

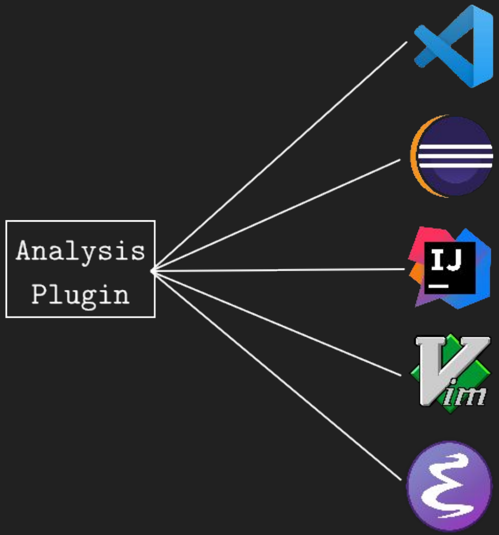
Charlie Mrad

Supervised by Idriss Riouak

Problem

- Static analysis results often unintuitive
- Delays when delivering analysis results are frustrating
- Ideally: fast analysis times and in-editor results

- Use LSP for IDE integration
- LSP is complicated
- Many IDEs don't support all LSP features
- Ideally:

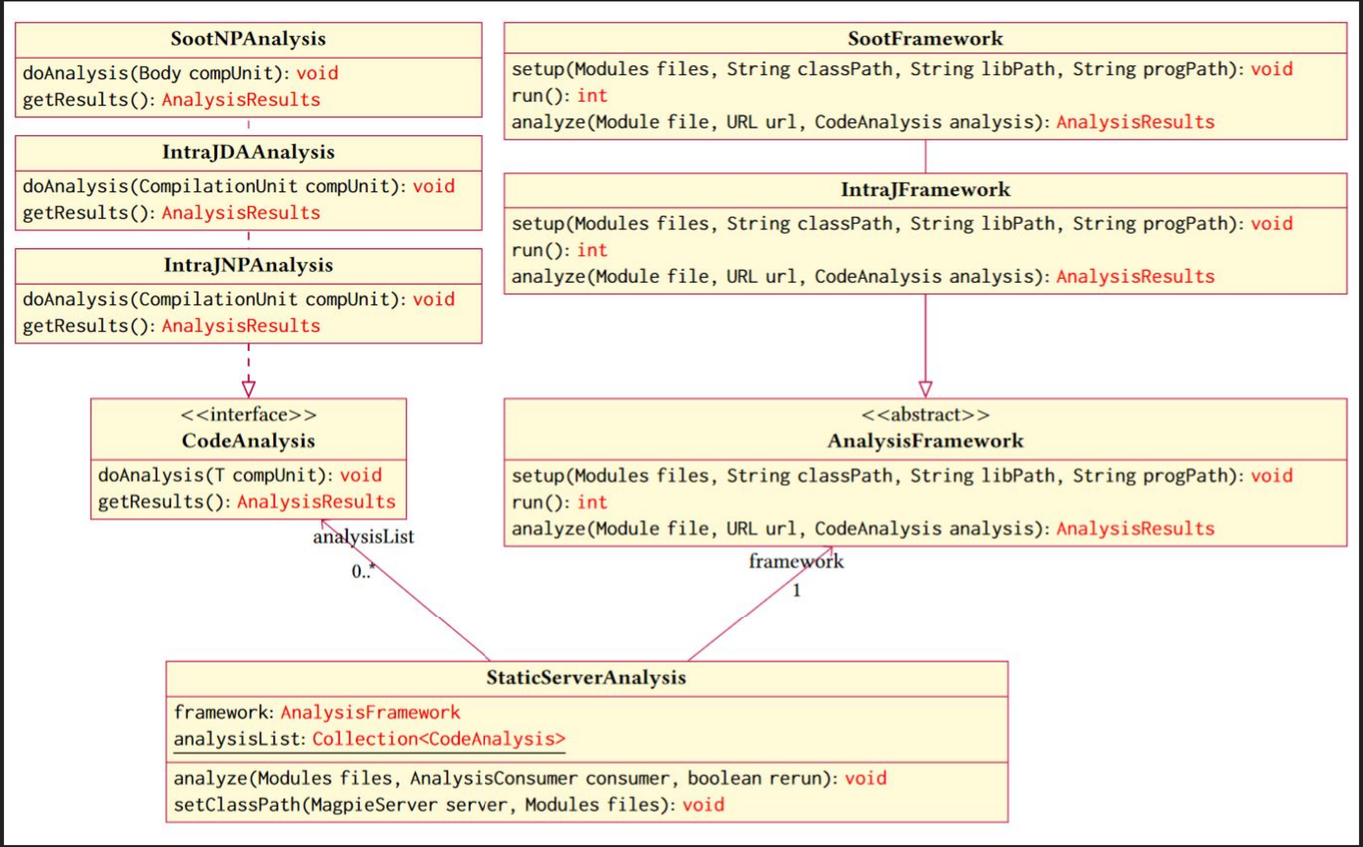


Solution

- `MagpieBridge` abstract LSP code
- Reduces coding overhead
- Makes plugin easy to port from one IDE to another

- `IntraJ` is a static analyzer for Java source code
- RAGs enables on-demand evaluation
- Command line only

Structure



SootNPAnalysis
doAnalysis(Body compUnit): void
getResults(): AnalysisResults
IntraJDAAnalysis
doAnalysis(CompilationUnit compUnit): void
getResults(): AnalysisResults
IntraJNPAnalysis
doAnalysis(CompilationUnit compUnit): void
getResults(): AnalysisResults
<<interface>>
CodeAnalysis
doAnalysis(T compUnit): void
getResults(): AnalysisResults
SootFramework
setup(Modules files, String classPath, String libPath, String progPath): void
run(): int
analyze(Module file, URL url, CodeAnalysis analysis): AnalysisResults
IntraJFramework
setup(Modules files, String classPath, String libPath, String progPath): void
run(): int
analyze(Module file, URL url, CodeAnalysis analysis): AnalysisResults
<<abstract>>
AnalysisFramework
setup(Modules files, String classPath, String libPath, String progPath): void
run(): int
analyze(Module file, URL url, CodeAnalysis analysis): AnalysisResults
analysisList
0..*
framework
1
StaticServerAnalysis
framework: AnalysisFramework
analysisList: Collection<CodeAnalysis>
analyze(Modules files, AnalysisConsumer consumer, boolean rerun): void
setClassPath(MagpieServer server, Modules files): void

What is a static analysis?

Static program analysis is the analysis of computer software that is performed without actually executing programs.

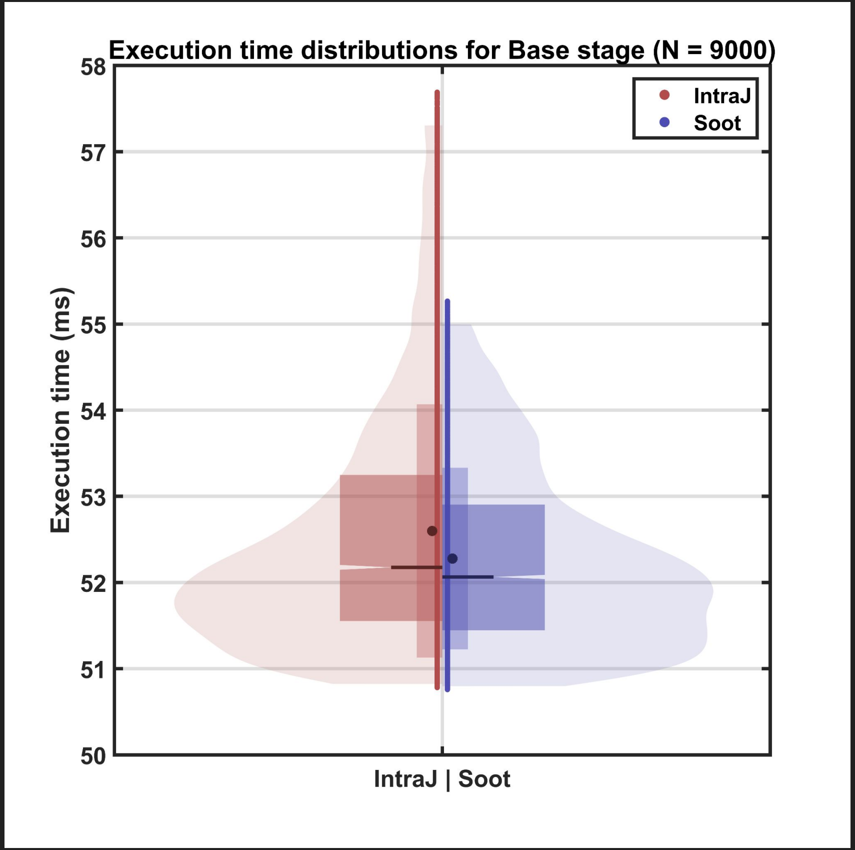
- Wikipedia

- Analysis performed without running program
- Example check string equality for Java
  - If “==” is used warn the developer
- Done by analyzing the source code

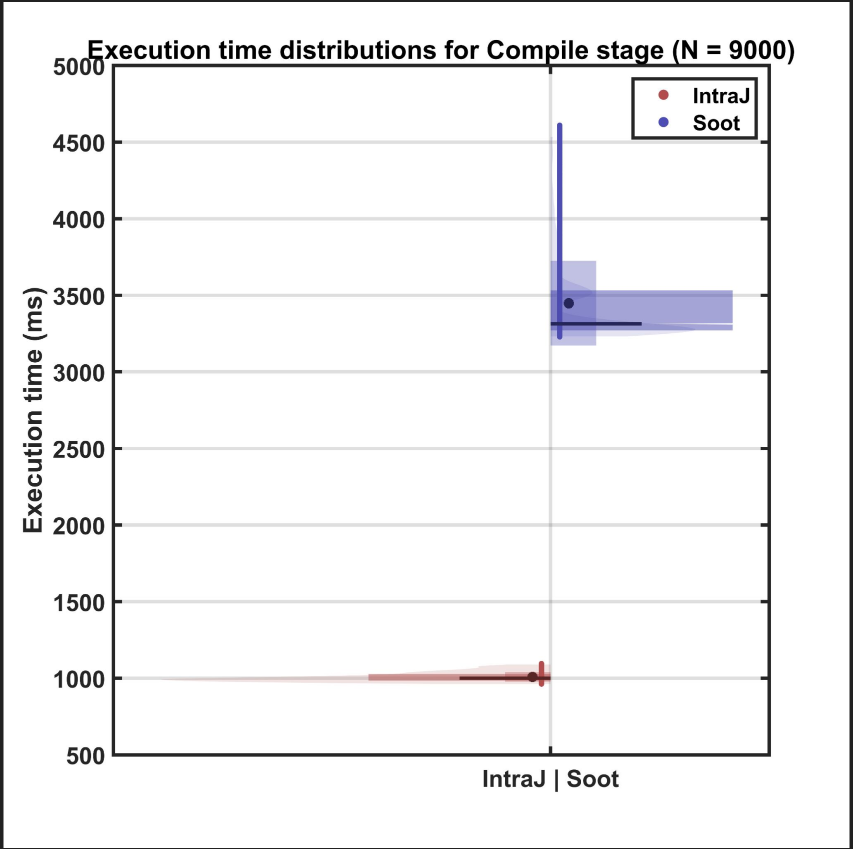
Demo

- For evaluation used null-pointer analysis
- Analysis executed on `ANTLRParser.java`
- Around 3000 lines of code

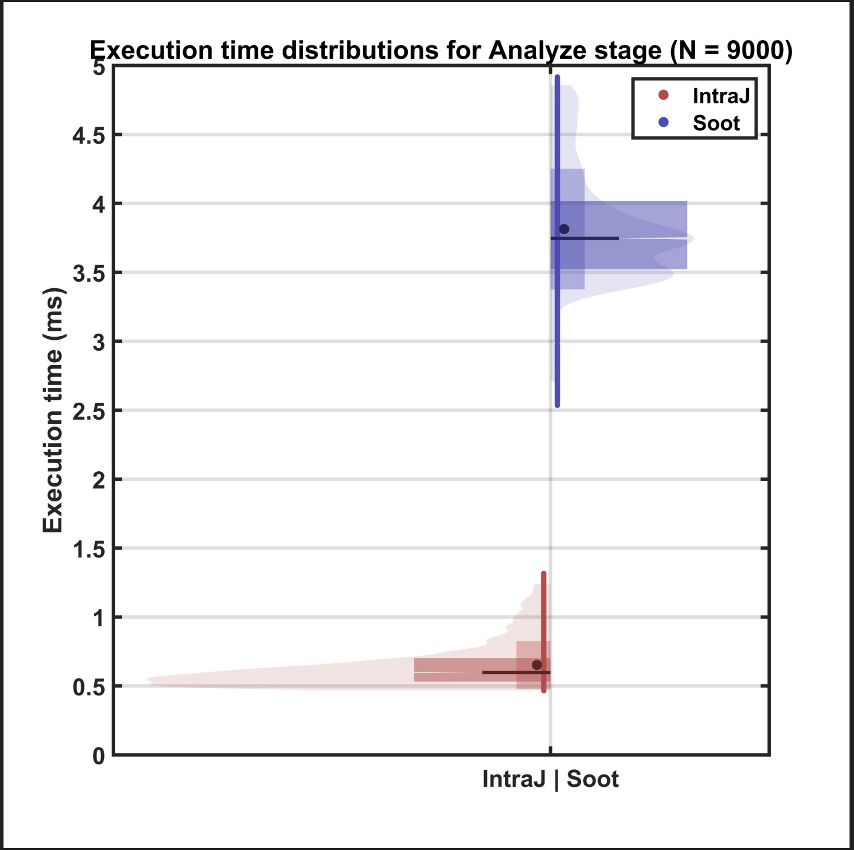
Base stage evaluation results



Compile stage evaluation results



Analyze stage evaluation results



Roughly 4.5X faster

Summary

- LSP is hard
- Static analysis often unintuitive
- `MagpieBridge` helps with LSP and intuitive analysis
- IntraJ provides performant analysis
- Flexible implementation for future extension
- `IntraJ` compared to `Soot` indicates `IntraJ` is faster
- Speed increase possibly due to RAGs