Declarative Code Analysis

Existing Solutions, Challenges and Research Directions

Semyon Grigorev

December 19, 2022

Declarative Code Analysis

- We need a way to specify code analysis in declarative manner

Declarative Code Analysis

- We need a way to specify code analysis in declarative manner
- There are few ways to get it

Declarative Code Analysis

- We need a way to specify code analysis in declarative manner
- There are few ways to get it
- We are focused on graph-like code abstractions declarative analysis
    - Graph querying engines
    - Datalog-like engines

Declarative Code Analysis: Global Questions

What is the goal of analysis?

- ? Analytics
- ? Vulnerability detection
- ? Code smells detection
- ? . . .

Declarative Code Analysis: Global Questions

What is the goal of analysis?

- ? Analytics
- ? Vulnerability detection
- ? Code smells detection
- ? . . .

Where the place of developed tool in software development process?

- ? Part of CI
- ? IDE-level analysis
- ? Compiler-level analysis
- ? Standalone server-side analysis
- ? . . .

Declarative Code Analysis: Global Questions

What is the goal of analysis?

- ? Analytics
- ? Vulnerability detection
- ? Code smells detection
- ? ...

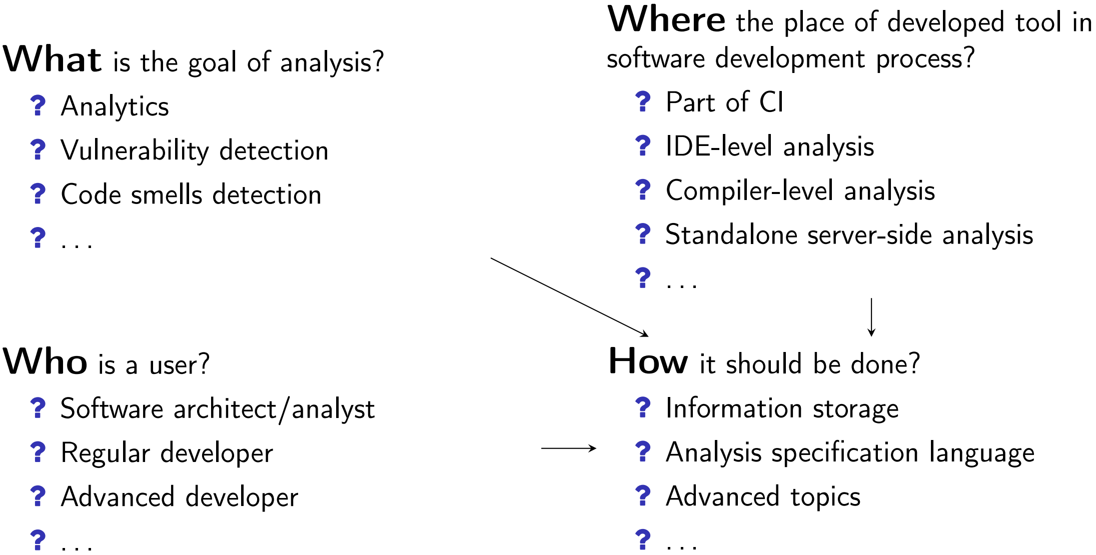
Who is a user?

- ? Software architect/analyst
- ? Regular developer
- ? Advanced developer
- ? ...

Where the place of developed tool in software development process?

- ? Part of CI
- ? IDE-level analysis
- ? Compiler-level analysis
- ? Standalone server-side analysis
- ? ...

Declarative Code Analysis: Global Questions



What is the goal of analysis?

- Analytics
- Vulnerability detection
- Code smells detection
- ...

Where the place of developed tool in software development process?

- Part of CI
- IDE-level analysis
- Compiler-level analysis
- Standalone server-side analysis
- ...

Who is a user?

- Software architect/analyst
- Regular developer
- Advanced developer
- ...

How it should be done?

- Information storage
- Analysis specification language
- Advanced topics
- ...

Declarative Code Analysis: How

How it should be done?

- ? Information storage
- ? Analysis specification language
- ? Advanced topics
- ? ...

Declarative Code Analysis: How

How it should be done?

- ? Information storage
- ? Analysis specification language
- ? Advanced topics
- ? . . .

→

Information storage

- Relational database
- Graph database
- Custom problem-specific storage

Declarative Code Analysis: How

How it should be done?

- ? Information storage
- ? Analysis specification language
- ? Advanced topics
- ? ...

Information storage

- Relational database
- Graph database
- Custom problem-specific storage

Analysis specification language

- Cypher/GQL-like language
- Datalog-like language
- Custom domain-specific language

Declarative Code Analysis: How

How it should be done?

- ? Information storage
- ? Analysis specification language
- ? Advanced topics
- ? . . .

Information storage

- Relational database
- Graph database
- Custom problem-specific storage

Analysis specification language

- Cypher/GQL-like language
- Datalog-like language
- Custom domain-specific language

Advanced topics

- Incremental analysis
- Detailed information to propose fixes, query result analysis
  - **What**: potential null pointer exception
  - **Why**: because there is **this particular path** in your program
- Query debugging (Why my query goes wrong?)

Declarative Code Analysis: Outcomes

- It is unlikely possible to create universal solution
  - Simplicity of analysis specification $\overset{?}{\Longleftrightarrow}$ Ability to specify nontrivial analysis
  - High performance $\overset{?}{\Longleftrightarrow}$ Additional data structures for query debugging, answer analysis, etc
    - Especially for massive code analysis
  - ...

CodeQL (GitHub/Microsoft)

- `https://codeql.github.com/`
- Vulnerabilities detection engine
- Custom analysis specification language: QL
  - Object-oriented DSL
  - Translation to Datalog
- Custom CodeQL database

CodeQL (GitHub/Microsoft)

- `https://codeql.github.com/`
- Vulnerabilities detection engine
- Custom analysis specification language: QL
  - Object-oriented DSL
  - Translation to Datalog
- Custom CodeQL database

+ Detailed query result explanation

− Query debugging

− Incrementalization

NG SAST (ShiftLeft)

- `https://www.shiftleft.io/`
- Static application security testing (vulnerability detection)
- Ocular (Joern) as a graph storage and query engine
  - Custom graph database: OverflowDB
  - Custom Gremlin-based graph query language

NG SAST (ShiftLeft)

- `https://www.shiftleft.io/`
- Static application security testing (vulnerability detection)
- Ocular (Joern) as a graph storage and query engine
  - Custom graph database: OverflowDB
  - Custom Gremlin-based graph query language

? Detailed query result explanation

– Query debugging

– Incrementalization

NG SAST (ShiftLeft)

- `https://www.shiftleft.io/`
- Static application security testing (vulnerability detection)
- Ocular (Joern) as a graph storage and query engine
  - Custom graph database: OverflowDB
  - Custom Gremlin-based graph query language

? Detailed query result explanation

– Query debugging

– Incrementalization

- Good start point ot play with graph query based code analysis
  - LLVM bitcode to Code Property Graph converter

Soufflé (Oracle Labs/The University of Sydney)

- `https://souffle-lang.github.io/index.html`
- General-purpose static code analysis
- Logic programming language inspired by Datalog
  - Translation to C++
  - Can use external storages for relations

Soufflé (Oracle Labs/The University of Sydney)

- `https://souffle-lang.github.io/index.html`
- General-purpose static code analysis
- Logic programming language inspired by Datalog
  - Translation to C++
  - Can use external storages for relations

- ⚙ Query debugging and results analysis (provenance)
- ⚙ Incrementalization
- ⚙ Cloud infrastructure

Soufflé (Oracle Labs/The University of Sydney)

- `https://souffle-lang.github.io/index.html`
- General-purpose static code analysis
- Logic programming language inspired by Datalog
  - Translation to C++
  - Can use external storages for relations

- ⚙ Query debugging and results analysis (provenance)
- ⚙ Incrementalization
- ⚙ Cloud infrastructure

- Good start point ot play with datalog-based code analysis
  - Doop: Souffle-based framework for Java pointer and taint analysis
  - cclyzer++: Souffle-based global pointer analysis for LLVM code

IncA (Johannes Gutenberg University Mainz)

- `https://gitlab.rlp.net/plmz/inca-scala`
- Incremental static code analysis framework
- Datalog-like DSL
- **Aimed to provide IDE-level incremental analysis**

IncA (Johannes Gutenberg University Mainz)

- `https://gitlab.rlp.net/plmz/inca-scala`
- Incremental static code analysis framework
- Datalog-like DSL
- **Aimed to provide IDE-level incremental analysis**

- ✚ Incrementalization: Eclipse Viatra as a backend
- ⚙ Query debugging
- ⚙ Detailed query result explanation

IncA (Johannes Gutenberg University Mainz)

- `https://gitlab.rlp.net/plmz/inca-scala`
- Incremental static code analysis framework
- Datalog-like DSL
- **Aimed to provide IDE-level incremental analysis**

➕ Incrementalization: Eclipse Viatra as a backend

⚙ Query debugging

⚙ Detailed query result explanation

- Good start point for IDE-level declarative code analysis

ProgQuery

- `https://github.com/OscarRodriguezPrieto/ProgQuery`
- An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations (2020)
- Neo4j-based
  - Cypher query language
  - Gremlin API
  - Java native API
- Evaluation shows (see paper above)
  - Can be more expressive than CodeQL and other tools
  - Can demonstrates better performance than CodeQL and other tools

ProgQuery Against Other Systems[1]

- **Wiggle 1.0** — source-code querying system based on a graph data model stored in Neo4j. The Cypher graph query language is used to express advanced queries, including syntactic (mainly) and some semantic properties of programs.
- **Semmle CodeQL 1.20**, a code analysis platform to perform detailed analyses of source code. Semmle allows writing queries in QL, an object-oriented variant of the Datalog. Semmle CodeQL stores programs in a PostgreSQL relational database.
- **ProgQuery 1.1.** is measured with the same two Neo4j versions we used to measure Wiggle: Neo4j Community 3.5.6 server and Neo4j embedded 3.3.4.

[1] An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations

Expressivity of Cypher[2][3]

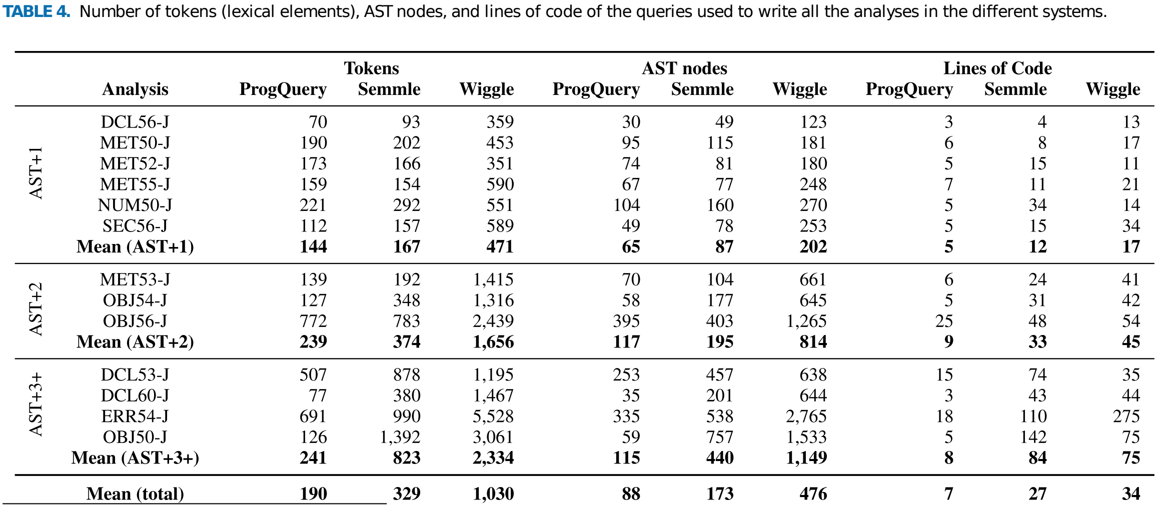


TABLE 4. Number of tokens (lexical elements), AST nodes, and lines of code of the queries used to write all the analyses in the different systems.

		Tokens			AST nodes			Lines of Code		
	Analysis	**ProgQuery**	**Semmle**	**Wiggle**	**ProgQuery**	**Semmle**	**Wiggle**	**ProgQuery**	**Semmle**	**Wiggle**
AST+1	DCL56-J	70	93	359	30	49	123	3	4	13
	MET50-J	190	202	453	95	115	181	6	8	17
	MET52-J	173	166	351	74	81	180	5	15	11
	MET55-J	159	154	590	67	77	248	7	11	21
	NUM50-J	221	292	551	104	160	270	5	34	14
	SEC56-J	112	157	589	49	78	253	5	15	34
	Mean (AST+1)	**144**	**167**	**471**	**65**	**87**	**202**	**5**	**12**	**17**
AST+2	MET53-J	139	192	1,415	70	104	661	6	24	41
	OBJ54-J	127	348	1,316	58	177	645	5	31	42
	OBJ56-J	772	783	2,439	395	403	1,265	25	48	54
	Mean (AST+2)	**239**	**374**	**1,656**	**117**	**195**	**814**	**9**	**33**	**45**
AST+3+	DCL53-J	507	878	1,195	253	457	638	15	74	35
	DCL60-J	77	380	1,467	35	201	644	3	43	44
	ERR54-J	691	990	5,528	335	538	2,765	18	110	275
	OBJ50-J	126	1,392	3,061	59	757	1,533	5	142	75
	Mean (AST+3+)	**241**	**823**	**2,334**	**115**	**440**	**1,149**	**8**	**84**	**75**
	Mean (total)	**190**	**329**	**1,030**	**88**	**173**	**476**	**7**	**27**	**34**

[2] An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations

[3] Highly depends on information stored in DB: DB size vs query size and performance

Performance of ProgQuery[4]

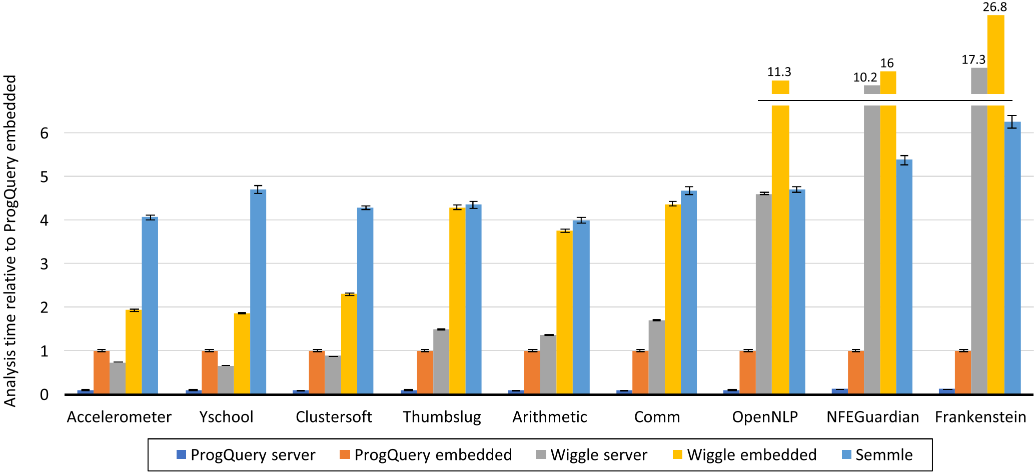
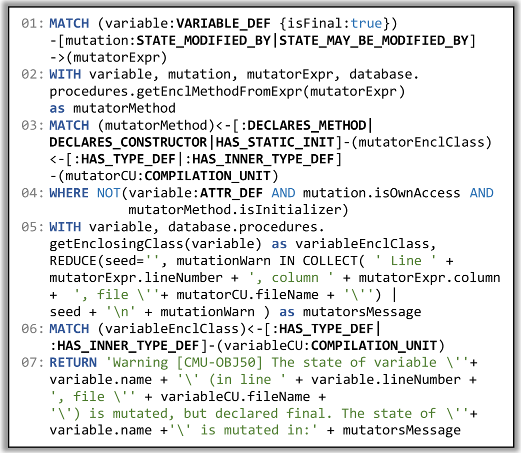


FIGURE 6. Average analysis execution time for increasing program sizes (execution times are relative to ProgQuery embedded).

[4] An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations

Example of Code Analysis in Cypher[5]

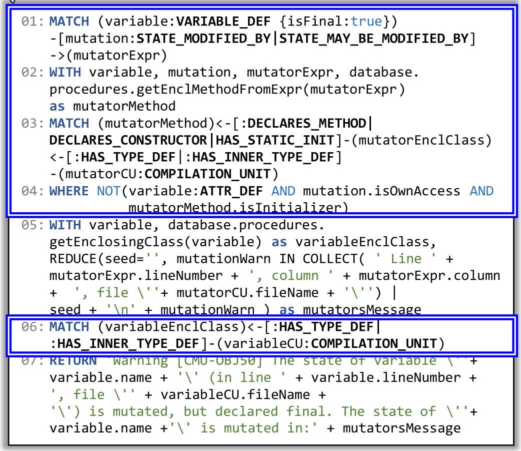


```
MATCH (variable:VARIABLE_DEF {isFinal:true})
-[mutation:STATE_MODIFIED_BY|STATE_MAY_BE_MODIFIED_BY]
->(mutatorExpr)
WITH variable, mutation, mutatorExpr, database.
procedures.getEnclMethodFromExpr(mutatorExpr)
as mutatorMethod
MATCH (mutatorMethod)<-[:DECLARES_METHOD|
DECLARES_CONSTRUCTOR|HAS_STATIC_INIT]-(mutatorEnclClass)
<-[:HAS_TYPE_DEF|:HAS_INNER_TYPE_DEF]
-(mutatorCU:COMPILATION_UNIT)
WHERE NOT(variable:ATTR_DEF AND mutation.isOwnAccess AND
          mutatorMethod.isInitializer)
WITH variable, database.procedures.
getEnclosingClass(variable) as variableEnclClass,
REDUCE(seed='', mutationWarn IN COLLECT( ' Line ' +
mutatorExpr.lineNumber + ', column ' + mutatorExpr.column
+  ', file \''+ mutatorCU.fileName + '\'') |
seed + '\n' + mutationWarn ) as mutatorsMessage
MATCH (variableEnclClass)<-[:HAS_TYPE_DEF|
:HAS_INNER_TYPE_DEF]-(variableCU:COMPILATION_UNIT)
RETURN 'Warning [CMU-OBJ50] The state of variable \''+
variable.name + '\' (in line ' + variable.lineNumber +
', file \'' + variableCU.fileName +
'\') is mutated, but declared final. The state of \''+
variable.name +'\' is mutated in:' + mutatorsMessage
```

FIGURE 2. Cypher code implementing the OBJ50-J CERT CMU Java recommendation.

Example of Code Analysis in Cypher[5]

Analysis specification

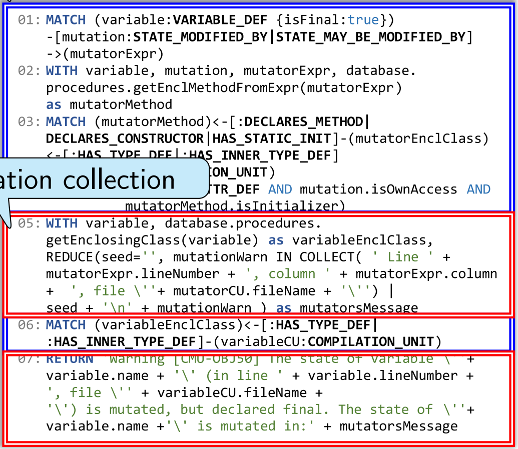


```
MATCH (variable:VARIABLE_DEF {isFinal:true})
-[mutation:STATE_MODIFIED_BY|STATE_MAY_BE_MODIFIED_BY]
->(mutatorExpr)
WITH variable, mutation, mutatorExpr, database.
procedures.getEnclMethodFromExpr(mutatorExpr)
as mutatorMethod
MATCH (mutatorMethod)<-[:DECLARES_METHOD|
DECLARES_CONSTRUCTOR|HAS_STATIC_INIT]-(mutatorEnclClass)
<-[:HAS_TYPE_DEF|:HAS_INNER_TYPE_DEF]
-(mutatorCU:COMPILATION_UNIT)
WHERE NOT(variable:ATTR_DEF AND mutation.isOwnAccess AND
          mutatorMethod.isInitializer)
WITH variable, database.procedures.
getEnclosingClass(variable) as variableEnclClass,
REDUCE(seed='', mutationWarn IN COLLECT( ' Line ' +
mutatorExpr.lineNumber + ', column ' + mutatorExpr.column
+  ', file \''+ mutatorCU.fileName + '\'') |
seed + '\n' + mutationWarn ) as mutatorsMessage
MATCH (variableEnclClass)<-[:HAS_TYPE_DEF|
:HAS_INNER_TYPE_DEF]-(variableCU:COMPILATION_UNIT)
RETURN 'Warning [CMU-OBJ50] The state of variable \''+
variable.name + '\' (in line ' + variable.lineNumber +
', file \'' + variableCU.fileName +
'\') is mutated, but declared final. The state of \''+
variable.name +'\' is mutated in:' + mutatorsMessage
```

FIGURE 2. Cypher code implementing the OBJ50-J CERT CMU Java recommendation.

Example of Code Analysis in Cypher[5]

Analysis specification



```
01: MATCH (variable:VARIABLE_DEF {isFinal:true})
    -[mutation:STATE_MODIFIED_BY|STATE_MAY_BE_MODIFIED_BY]
    ->(mutatorExpr)
02: WITH variable, mutation, mutatorExpr, database.
    procedures.getEnclMethodFromExpr(mutatorExpr)
    as mutatorMethod
03: MATCH (mutatorMethod)<-[:DECLARES_METHOD|
    DECLARES_CONSTRUCTOR|HAS_STATIC_INIT]-(mutatorEnclClass)
    <-[:HAS_TYPE_DEF|:HAS_INNER_TYPE_DEF]
                              ON_UNIT)
                              TR_DEF AND mutation.isOwnAccess AND
              mutatorMethod.isInitializer)
05: WITH variable, database.procedures.
    getEnclosingClass(variable) as variableEnclClass,
    REDUCE(seed='', mutationWarn IN COLLECT( ' Line ' +
    mutatorExpr.lineNumber + ', column ' + mutatorExpr.column
    +  ', file \''+ mutatorCU.fileName + '\'') |
    seed + '\n' + mutationWarn ) as mutatorsMessage
06: MATCH (variableEnclClass)<-[:HAS_TYPE_DEF|
    :HAS_INNER_TYPE_DEF]-(variableCU:COMPILATION_UNIT)
07: RETURN 'Warning [CMU-OBJ50] The state of variable \'' +
    variable.name + '\' (in line ' + variable.lineNumber +
    ', file \'' + variableCU.fileName +
    '\') is mutated, but declared final. The state of \''+
    variable.name +'\' is mutated in:' + mutatorsMessage
```

Detailed information collection

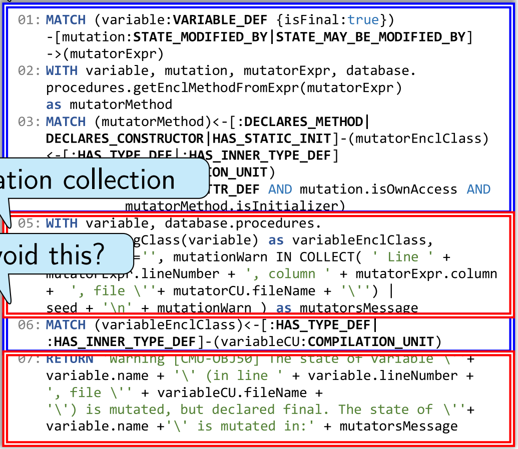
FIGURE 2. Cypher code implementing the OBJ50-J CERT CMU Java recommendation.

Example of Code Analysis in Cypher[5]

Analysis specification

Detailed information collection

Can we avoid this?



```
01: MATCH (variable:VARIABLE_DEF {isFinal:true})
    -[mutation:STATE_MODIFIED_BY|STATE_MAY_BE_MODIFIED_BY]
    ->(mutatorExpr)
02: WITH variable, mutation, mutatorExpr, database.
    procedures.getEnclMethodFromExpr(mutatorExpr)
    as mutatorMethod
03: MATCH (mutatorMethod)<-[:DECLARES_METHOD|
    DECLARES_CONSTRUCTOR|HAS_STATIC_INIT]-(mutatorEnclClass)
    <-[:HAS_TYPE_DEF|:HAS_INNER_TYPE_DEF]
                      ION_UNIT)
                      TTR_DEF AND mutation.isOwnAccess AND
              mutatorMethod.isInitializer)
05: WITH variable, database.procedures.
                 gClass(variable) as variableEnclClass,
                 =''', mutationWarn IN COLLECT( ' Line ' +
                 r.lineNumber + ', column ' + mutatorExpr.column
    +  ', file \''+ mutatorCU.fileName + '\'') |
    seed + '\n' + mutationWarn ) as mutatorsMessage
06: MATCH (variableEnclClass)<-[:HAS_TYPE_DEF|
    :HAS_INNER_TYPE_DEF]-(variableCU:COMPILATION_UNIT)
07: RETURN 'Warning [CMU-OBJ50] The state of variable \'' +
    variable.name + '\' (in line ' + variable.lineNumber +
    ', file \'' + variableCU.fileName +
    '\') is mutated, but declared final. The state of \''+
    variable.name +'\' is mutated in:' + mutatorsMessage
```

FIGURE 2. Cypher code implementing the OBJ50-J CERT CMU Java recommendation.

Conclusion

- Datalog-like languages wore widely used in mature tools (for deep code analysis)
  - Real-world languages are extensions of Datalog, not pure Datalog: arithmetics, aggregation, algebraic data types, ...
  - Can it be extended more? (Formulog: Datalog for SMT-Based Static Analysis)
  - Datalog is like C++: powerful but too complex for non-familiar users

Conclusion

- Datalog-like languages wore widely used in mature tools (for deep code analysis)
    - Real-world languages are extensions of Datalog, not pure Datalog: arithmetics, aggregation, algebraic data types, . . .
    - Can it be extended more? (Formulog: Datalog for SMT-Based Static Analysis)
    - Datalog is like C++: powerful but too complex for non-familiar users
- But Cypher can be expressive enough against custom and Datalog-like DSLs
    - Especially for simple checkers specification
    - Graph database can be an appropriate storage (even Neo4j)
    - Cypher is like Python: simple, widely-spread but not so powerful

Conclusion

- Datalog-like languages wore widely used in mature tools (for deep code analysis)
  - Real-world languages are extensions of Datalog, not pure Datalog: arithmetics, aggregation, algebraic data types, . . .
  - Can it be extended more? (Formulog: Datalog for SMT-Based Static Analysis)
  - Datalog is like C++: powerful but too complex for non-familiar users
- But Cypher can be expressive enough against custom and Datalog-like DSLs
  - Especially for simple checkers specification
  - Graph database can be an appropriate storage (even Neo4j)
  - Cypher is like Python: simple, widely-spread but not so powerful
- There is no production ready solutions for IDE-level declarative code analysis

Conclusion

- Datalog-like languages wore widely used in mature tools (for deep code analysis)
  - Real-world languages are extensions of Datalog, not pure Datalog: arithmetics, aggregation, algebraic data types, . . .
  - Can it be extended more? (Formulog: Datalog for SMT-Based Static Analysis)
  - Datalog is like C++: powerful but too complex for non-familiar users
- But Cypher can be expressive enough against custom and Datalog-like DSLs
  - Especially for simple checkers specification
  - Graph database can be an appropriate storage (even Neo4j)
  - Cypher is like Python: simple, widely-spread but not so powerful
- There is no production ready solutions for IDE-level declarative code analysis
- Incremental analysis is a nontrivial challenge

Conclusion

- Datalog-like languages wore widely used in mature tools (for deep code analysis)
  - Real-world languages are extensions of Datalog, not pure Datalog: arithmetics, aggregation, algebraic data types, . . .
  - Can it be extended more? (Formulog: Datalog for SMT-Based Static Analysis)
  - Datalog is like C++: powerful but too complex for non-familiar users
- But Cypher can be expressive enough against custom and Datalog-like DSLs
  - Especially for simple checkers specification
  - Graph database can be an appropriate storage (even Neo4j)
  - Cypher is like Python: simple, widely-spread but not so powerful
- There is no production ready solutions for IDE-level declarative code analysis
- Incremental analysis is a nontrivial challenge
- Query debugging and results analysis is a nontrivial challenge

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison
- Query languages evaluation

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison
- Query languages evaluation
  - Whether advanced DSL needed?

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison
- Query languages evaluation
  - Whether advanced DSL needed?
  - Can GQL be an appropriate language?
  - GQL is SQL for graphs: **ISO standard** for graph query language
  - Cypher-like
  - Friendly to non-advanced users, widely used

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison
- Query languages evaluation
  - Whether advanced DSL needed?
  - Can GQL be an appropriate language?
  - GQL is SQL for graphs: **ISO standard** for graph query language
  - Cypher-like
  - Friendly to non-advanced users, widely used

- Dynamic data analysis
  - Incremental view maintenance
  - Incremental static code analysis
  - Persistent queries
  - ...

Challenges/Research Directions

- Graph databases evaluation
  - Code analysis related scenarios
  - Graph representations comparison
  - Low-level API comparison
- Query languages evaluation
  - Whether advanced DSL needed?
  - Can GQL be an appropriate language?
  - GQL is SQL for graphs: **ISO standard** for graph query language
  - Cypher-like
  - Friendly to non-advanced users, widely used
- Dynamic data analysis
  - Incremental view maintenance
  - Incremental static code analysis
  - Persistent queries
  - ...
- Query debugging and results analysis
  - Appropriate data structures
  - Quick fixes
  - ...
- Datalog extensions and Datalog engines
  - Performance evaluation
  - Expressivity comparison
  - ...