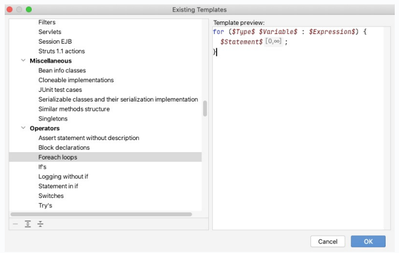
CodeSpider: Automatic Code Querying with Multi-modal Conjunctive Query Synthesis

Chengpeng Wang, Prism Group, The Hong Kong University of Science and Technology

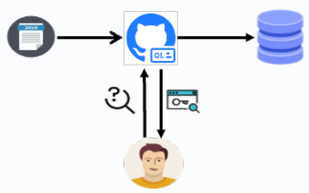
Existing Code Querying Tools

- IDEs
  - String match and structural search
  - E.g., IntelliJ



Restrictive search template

- Datalog-based Analyzer
  - E.g., CodeQL

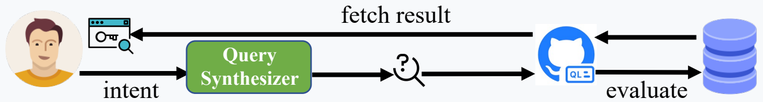


```
import java

from IfStmt ifstmt, Block block
where ifstmt.getThen() = block and
  block.getNumStmt() = 0
select ifstmt, "This 'if' statement is redundant."
```

Laborious manual effort

CodeSpider: Synthesizing Conjunctive Queries



Methods receiving a parameter with Log4jUtils type.

```
// positive example
public void foo(Log4jUtils a) {    return;    }

// negative example
private void goo(int a) {    return;    }
```

```
query(Method m) :-
  exists(Parameter p, Type t, String s)
    p = m.getPara() &&
    t = p.getType() &&
    s = t.getName() &&
    equals(s, "Log4jUtils")
```

Ease of use: Use Datalog-based analyzers as a black box
Capability: Leverage various relations describing program properties

Program IR: Relational Representation

```
// positive example
public void foo(Log4jUtils a)
{
   return;
}
// negative example
private void goo(int a) {
   return;
}
```

Parameter

id	idf_id	type_id	method_id
P1	*I3*	*T1*	*M1*
P2	*I4*	*T2*	*M2*

Identifier

id	name
I1	foo
I2	goo
I3	a
I4	a

Method

id	idf_id	ret_type_id	mdf_id
M1	*I1*	*T3*	*MDF1*
M2	*I2*	*T3*	*MDF2*

Type

id	name
T1	Log4jUtils
T2	int
T3	void

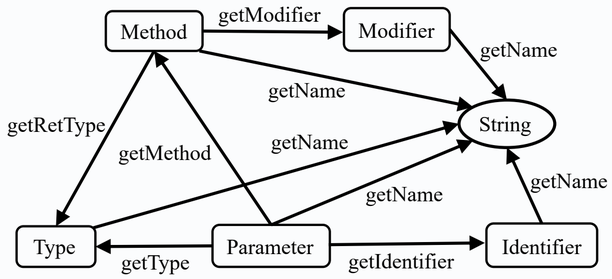
Modifier

id	name
MDF1	public
MDF2	private

Relations

Separating positive tuples from negative tuples

Stage I: Sketch Generation



Type Transition Net

Stage II: Query Refinement

```
query(Method m) :- true
```

```
query(Method m) :-
  exists(Parameter p, Type t)
    m = p.getMethod() &&
    t = p.getType()
```

```
query(Method m) :-
  exists(Parameter p, Type t, String s)
    m = p.getMethod() &&
    t = p.getType() &&
    s = t.getName() &&
    equals(s, "Log4jUtils")
```

```
query(Method m) :-
  exists(Parameter p, Type t)
    m = p.getMethod() &&
    t = p.getType() &&
    t = m.getRetType()
```

```
query(Method m) :-
  exists(Parameter p, Type t, String s)
    m = p.getMethod() &&
    t = p.getType() &&
    t = m.getRetType()
    s = t.getName() &&
    equals(s, "Log4jUtils")
```

Stage III: Query Selection

```
query(Method m) :-
  exists(String s)
    s = m.getName() && equals(s, "foo")
```

$\alpha = 1/3$, $\beta = 2$

Dual metrics

- Entity coverage (α) ↑
- Structural complexity (β) ↓

```
query(Method m) :-
  exists(Parameter p, Type t, String s)
    m = p.getMethod() && t = p.getType() &&
    s = t.getName() && equals(s, "Log4jUtils")
```

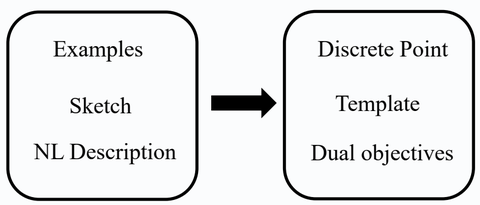
$\alpha = 3/3$, $\beta = 4$

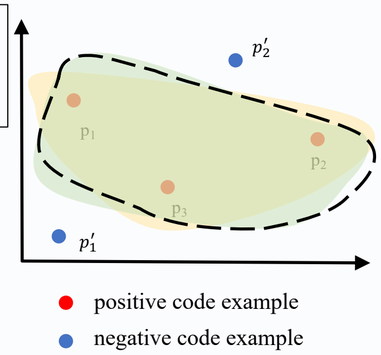
```
query(Method m) :-
  exists(Parameter p, Type t, Modifier f, String s1, String s2)
    m = p.getMethod() && t = p.getType() &&
    s1 = t.getName() && equals(s1, "Log4jUtils") &&
    f = m.getModifier() && s2 = f.getName() && equals(s2, "public")
```

$\alpha = 3/3$, $\beta = 7$

CodeSpider: Finding Best Abstraction

Find the *best* abstraction for given examples:
- Syntax: Conjunctive query
- Soundness: Cover positive points and exclude negative ones
- Optimality: Optimize the dual metrics





Evaluating CodeSpider

- Utilize a Datalog-based analyzer in Ant Group
- 173 relations with 1,093 attributes
- Synthesize string predicates with general suffix automaton
- Extract entities from the NL description with Stanford Named Entity Recognizer

ID	Description	(#P, #N)	(#C, #A)	Kind
1	Float variables of which the identifier contains "cash"	(3, 1)	(4, 4)	Var
2	Cast expressions from double-type to float type	(1, 2)	(6, 7)	Expr
3	Expressions comparing long int with int	(1, 2)	(3, 6)	Expr
4	Cast expressions casting long to int	(2, 1)	(6, 7)	Expr
5	Expressions comparing a variable and Boolean literal	(1, 3)	(4, 5)	Expr
6	New expressions of ArrayList	(1, 1)	(3, 3)	Expr
7	Logical-and expressions with literal as an operand	(2, 2)	(4, 5)	Expr
8	The import of LocalTime	(2, 1)	(3, 4)	Stmt
9	The import of the classes in log4j	(1, 1)	(2, 2)	Stmt
10	Labeled statements	(2, 2)	(1, 0)	Stmt
11	If-statements with a Boolean literal as a condition	(2, 1)	(2, 1)	Stmt
12	For-statements with a Boolean literal as a condition	(2, 1)	(2, 1)	Stmt
13	Public methods with void return type	(2, 1)	(5, 6)	Method
14	Methods receiving a parameter with Log4jUtils type	(2, 1)	(4, 4)	Method
15	Classes with a login method	(2, 1)	(3, 3)	Class
16	Classes containing a field with float type	(1, 1)	(4, 4)	Class

High Efficiency

- Average time cost: 3.35 sec
- Maximal time cost: 8.91 sec
- Minimal time cost: 2.23 sec
- 14 tasks finished in 4 sec