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

Chengpeng Wang　　Prism Group, HKUST

Code Querying

- Development assistance
  - How a specific class is used?

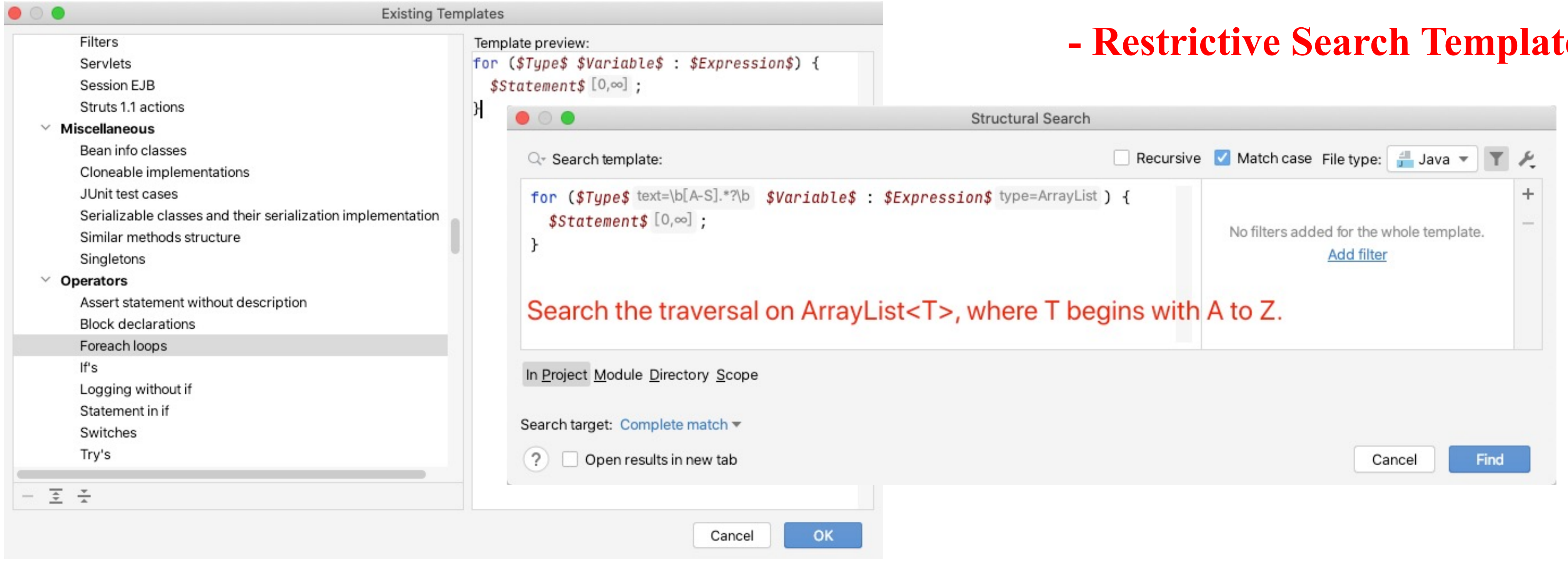
- Patch generation
  - Where is log4j interface invoked?

- Code measurement
  - How many projects import log4j as an external library?

Querying Code in IDEs

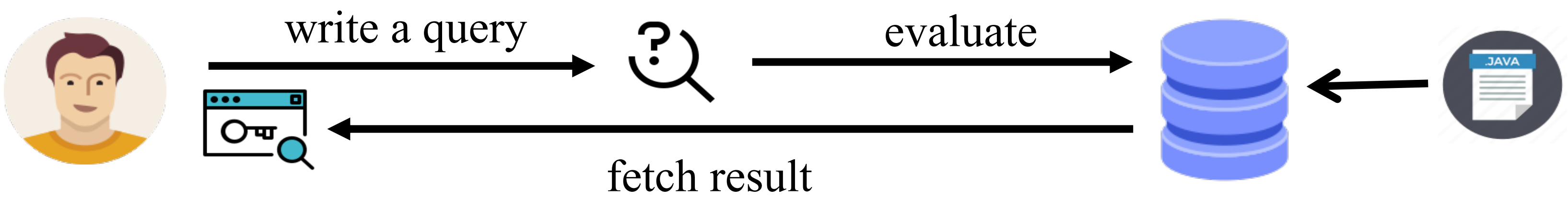
- IDEs
  - Eclipse: String matching
  - IntelliJ: Structural searching

- Restrictive Search Template



Querying Code with Datalog

- Datalog-based program analyzer, e.g., CodeQL
  - Write a Datalog-like query program to specify the querying condition



- Example
  - Find all the assignments from float to integer variables

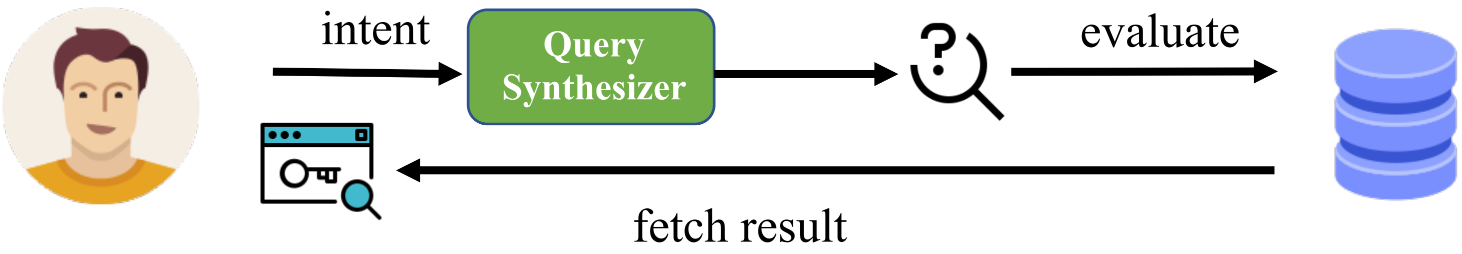
```
from AssignExpr a
where a.getRValue().getType() instanceof FloatingPointType
  and a.getLValue().getType() instanceof IntegralType
select a
```

+ Advanced Querying Support

- Heavy Learning Burden
- Verbose Query Writing

Our Aim: A Better Way

- Automatic synthesizing a conjunctive query



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

Preliminary: Relational Representation

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

Method

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

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

Type

id	name
T1	Log4jUtils
T2	int
T3	void

Modifier

id	name
MDF1	public
MDF2	private

Relations

Semantic constraint:

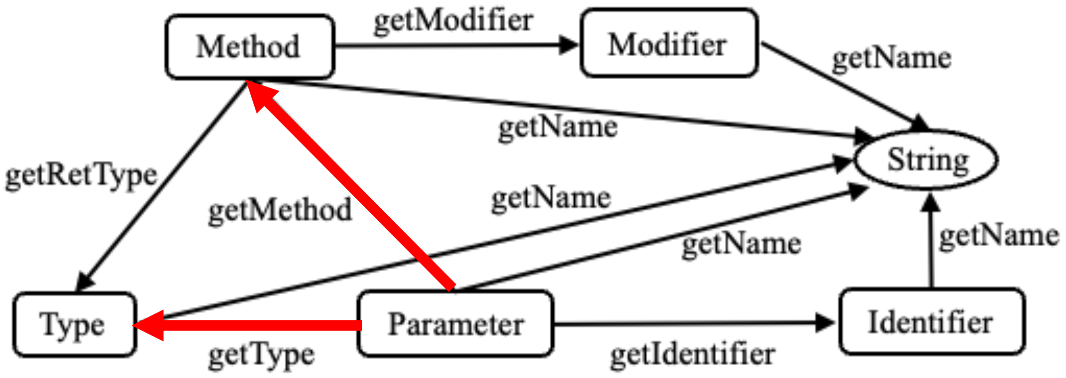
Separating positive tuples from negative tuples

Challenges

- Incredibly large search space
  - Large numbers of relations
  - Flexible combination of relations
- Multiple candidates satisfying the semantic constraint
  - Ineffective selection introduces the over-fitting problem

Stage I: Sketch Generation

- Summarize query sketches by the subgraphs of TTN
  - TTN encodes the type information of the attributes in each relation

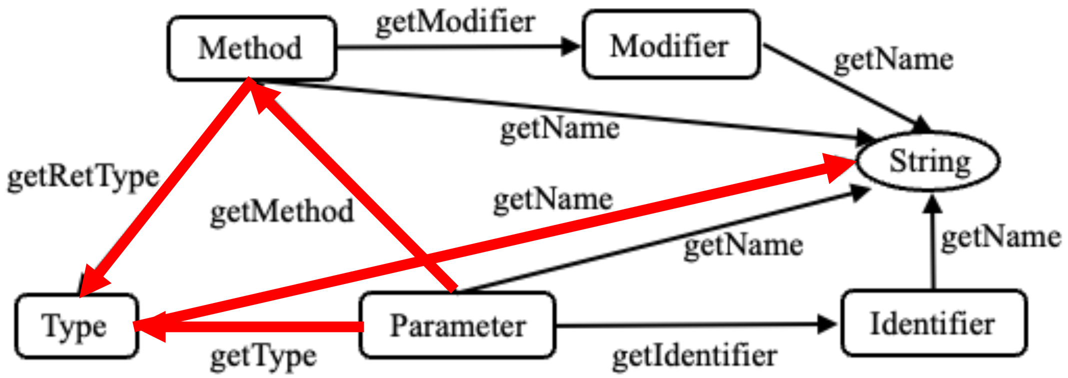


Type Transition Net (TTN)

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

Stage II: Query Refinement

- Obtain all the query candidates after query refinement
  - Add atomic formulas until positive and negative tuples are separated.
  - Discard the query if it misses a positive example.



Type Transition Net (TTN)

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)
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() &&
s = t.getName() &&
equals(s, "Log4jUtils")

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

- Select the query covering the entities in the NL description as many as possible with a simple form
  - Dual metrics: Entity coverage (α) , Structural complexity (β)

NL Description: Methods receiving a **parameter** with Log4jUtils **type**.

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

α = 1/3
β = 2

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

α = 3/3
β = 4

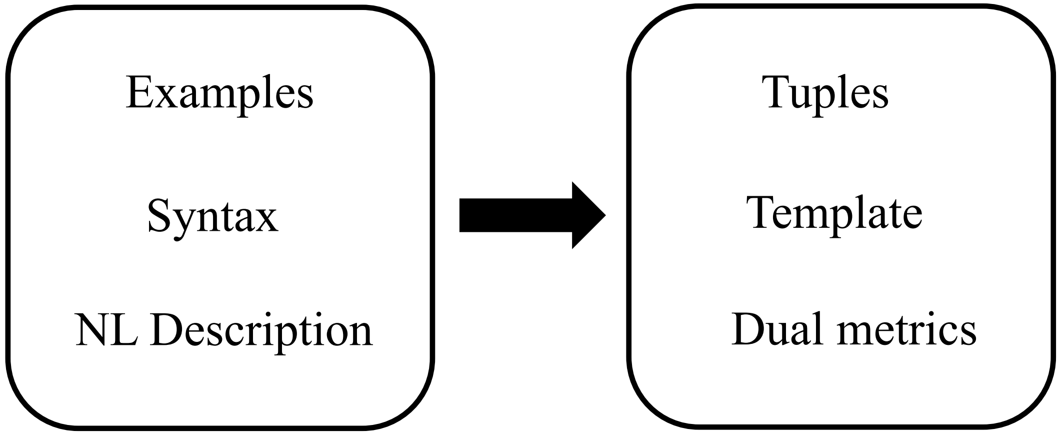
```
query(Method m) :-
   exists(Parameter p, Type t, Modifier f, String s1, String s2)
      p = m.getPara() && t = p.getType() &&
      s1 = t.getName() && equals(s1, "Log4jUtils") &&
      f = m.getModifier() && s2 = f.getName() &&
      equals(s2, "public")
```

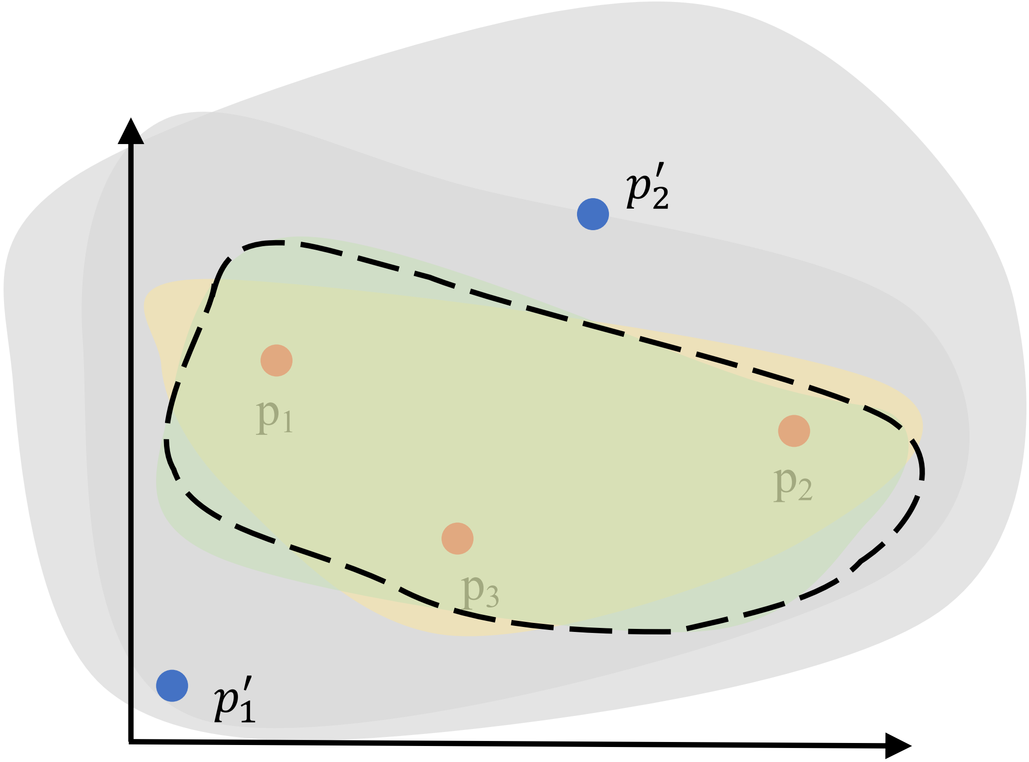
α = 3/3
β = 7

Another Perspective

Find the *best* abstraction for given tuples:

- Syntax: Conjunctive query
- Soundness: Cover positive tuples and exclude negative ones
- Optimality: Optimize the dual metrics





- positive tuple
- negative tuple

Implementation

- Implement *CodeSpider* in Python
  - Leverage GSA(General Suffix Automaton) to guide the synthesis of string constraints
    - Support the string predicates, including *prefixOf*, *suffixOf*, *equals*, and *contains*.
  - CodeSpider supports synthesizing queries for Sparrow, a commercial Datalog-based analyzer developed by Ant Group.
    - 173 relations with 1,093 attributes

ANT GROUP

Evaluation: Capability

- Code querying tasks

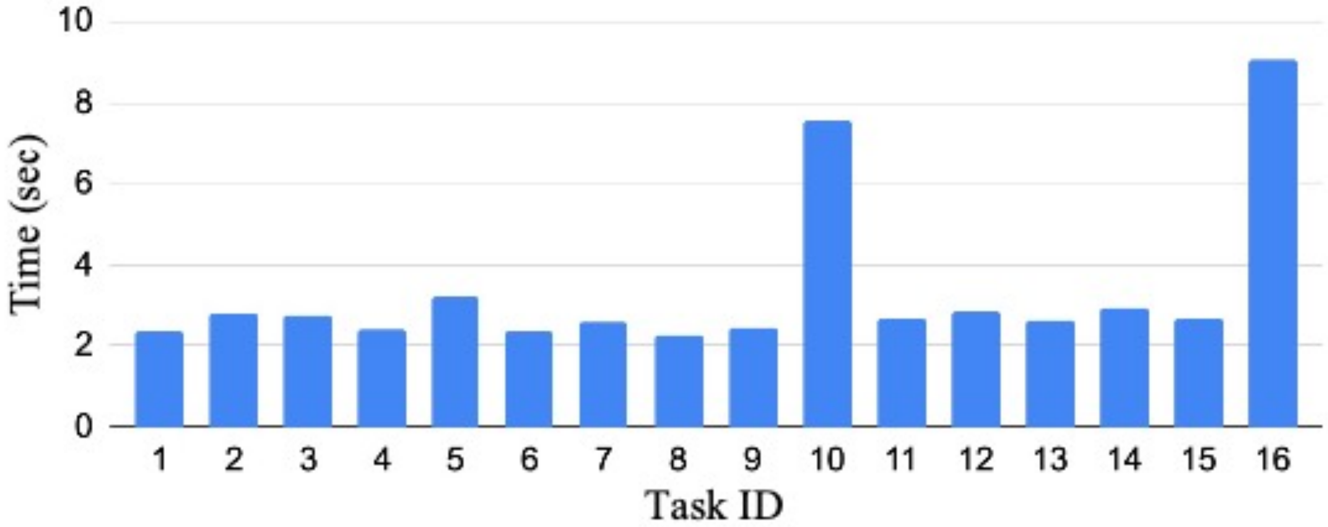
The numbers of positive/negative examples

The numbers of relations and clauses

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

Evaluation: High Efficiency

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



Conclusion

- (Conceptual) We define a multi-modal program synthesis problem for code querying.
- (Technical) We propose an efficient algorithm for synthesizing a conjunctive query.
- (Empirical) We evaluate our synthesis algorithm upon real-world code querying tasks and obtain the target queries efficiently.

Thank you for your listening!