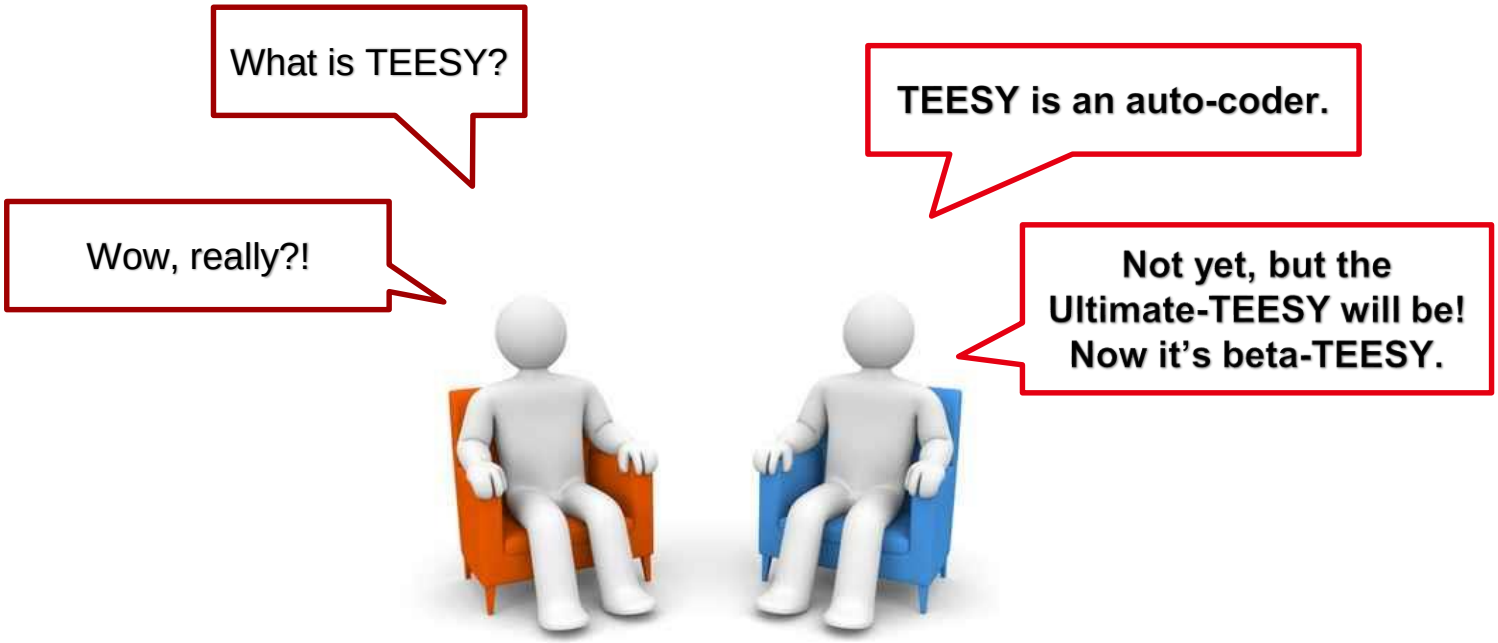
TEESY: Natural Language-Based Training-Free Program Synthesizer for Code Analysis

Zifan Nan

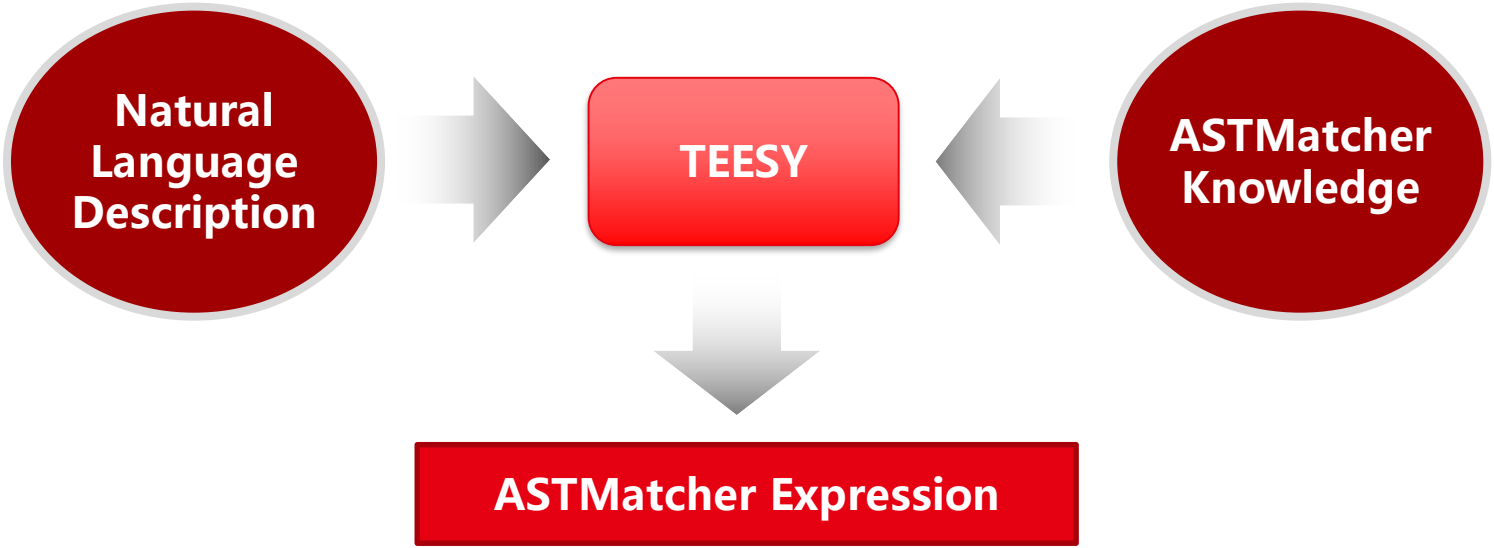
Advisor: Dr. Xipeng Shen

What is TEESY?

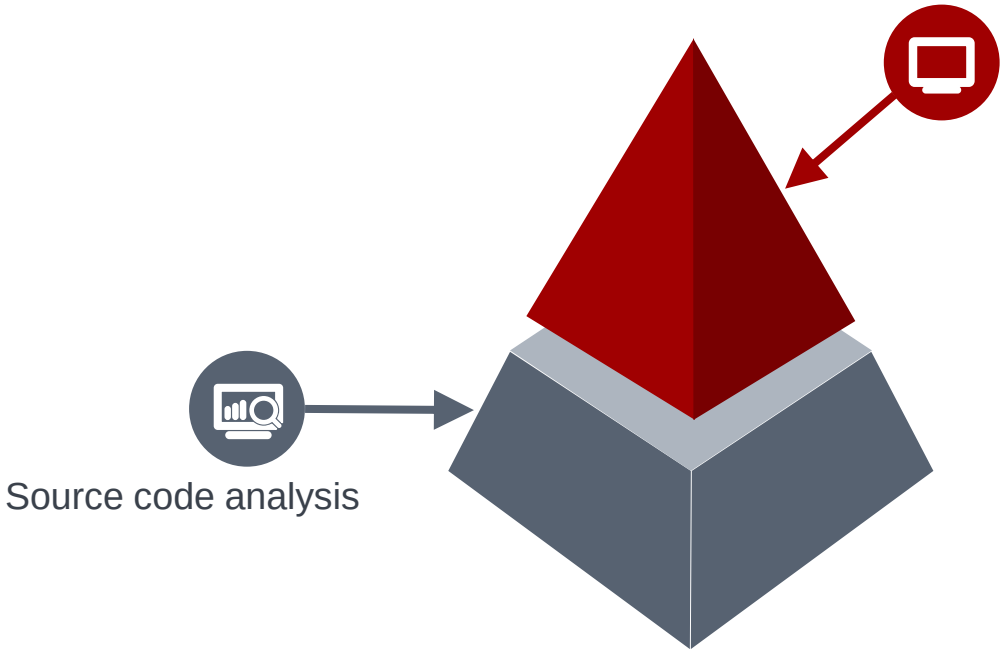


What is TEESY?

- Training-Free Natural Language-Based Synthesizer for Code Analysis.



Motivation

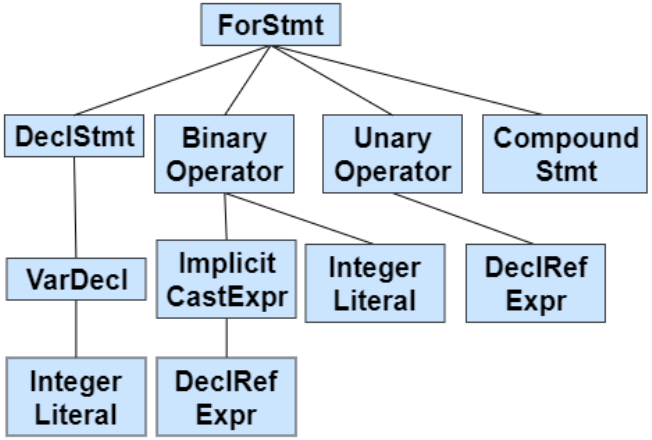


- Program optimizations
- Software debugging,
- Security
- …

- Developing programs to do code analysis has been a **daunting task** for general programmers.
- Some tools has been developed by experts. And **ASTMatcher** is one of them.

Abstract Syntax Tree (AST)

```
for ( int i=0;  i < 5;  i++){
}
```



AST

```
ForStmt 0x2a35568
|  - DeclStmt 0x2a35470
|    `- VarDecl 0x2a353f0  used i 'int' cinit
|        `- IntegerLiteral 0x2a35450 'int' 0
|  - BinaryOperator 0x2a354e8 '_Bool' '<'
|     | -ImplicitCastExpr 0x2a354d0 'int' <LValueToRValue>
|     |   `-DeclRefExpr 0x2a35488 'int' lvalue Var 0x2a353f0 'i' 'int'
|     `-IntegerLiteral 0x2a354b0 'int' 5
| - UnaryOperator 0x2a35538 'int' postfix '++'
|    `-DeclRefExpr 0x2a35510 'int' lvalue Var 0x2a353f0 'i' 'int'
` - CompoundStmt 0x2a35558
```

AST output from Clang

ASTMatcher

```
void FloatLoopCounter::registerMatchers(MatchFinder *Finder) {
    Finder->addMatcher(
        forStmt(hasIncrement(
            expr(hasType(realFloatingPointType())))).bind("for"), this);
}

void FloatLoopCounter::check(const MatchFinder::MatchResult &Result) {
    const auto *FS = Result.Nodes.getNodeAs<ForStmt>("for");
    diag(FS->getInc()->getExprLoc(), "loop induction expression should "
                                     "not have floating-point type");
}
```

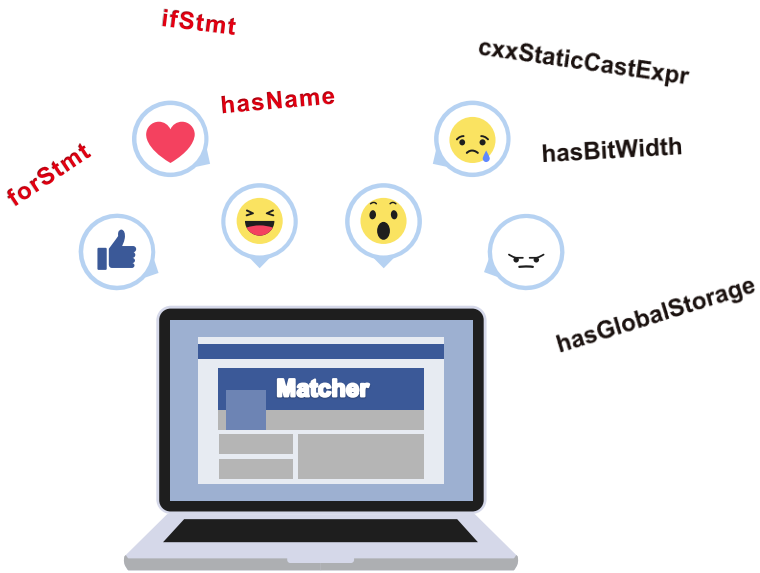
ASTMatcher expression used in Clang-Tidy, aims to find the type error in an for loop

ASTMatcher APIs

Matcher	Category	Input Type	Return Type	Description
forStmt	Node	Matcher<ForStmt>	Matcher<Stmt>	Matches for statements.
varDecl	Node	Matcher<VarDecl>	Matcher<Decl>	Matches variable declarations.
hasLoopInit	Traversal	Matcher<Stmt>	Matcher<ForStmt>	Matches the initialization statement of a for loop.
hasName	Narrowing	std::string	Matcher<NameDecl>	Matches NamedDecl nodes that have the specified name.

ASTMatcher APIs examples

ASTMatcher



Over 500 APIs

- Many subtle differences.
- Various usage conditions.

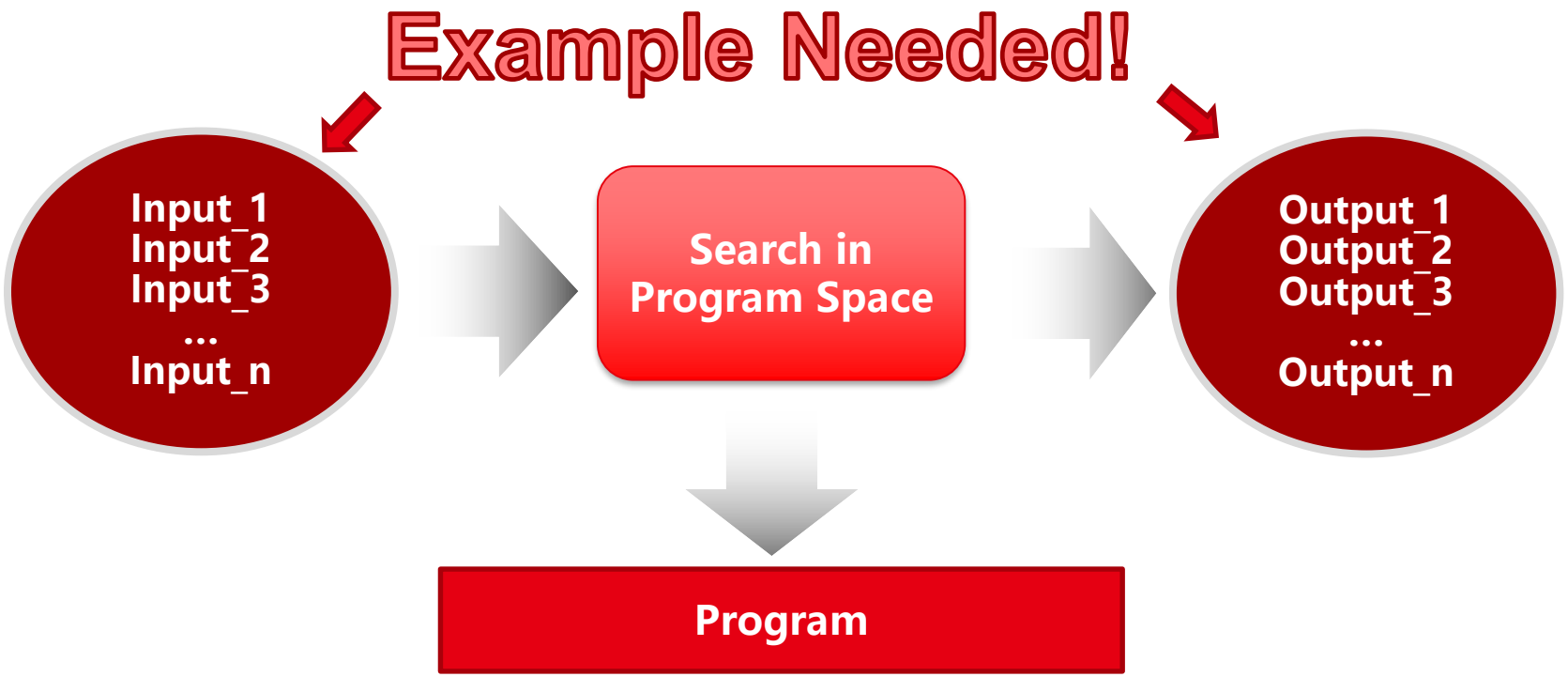
Long learning process

- Need to learn the APIs before using.

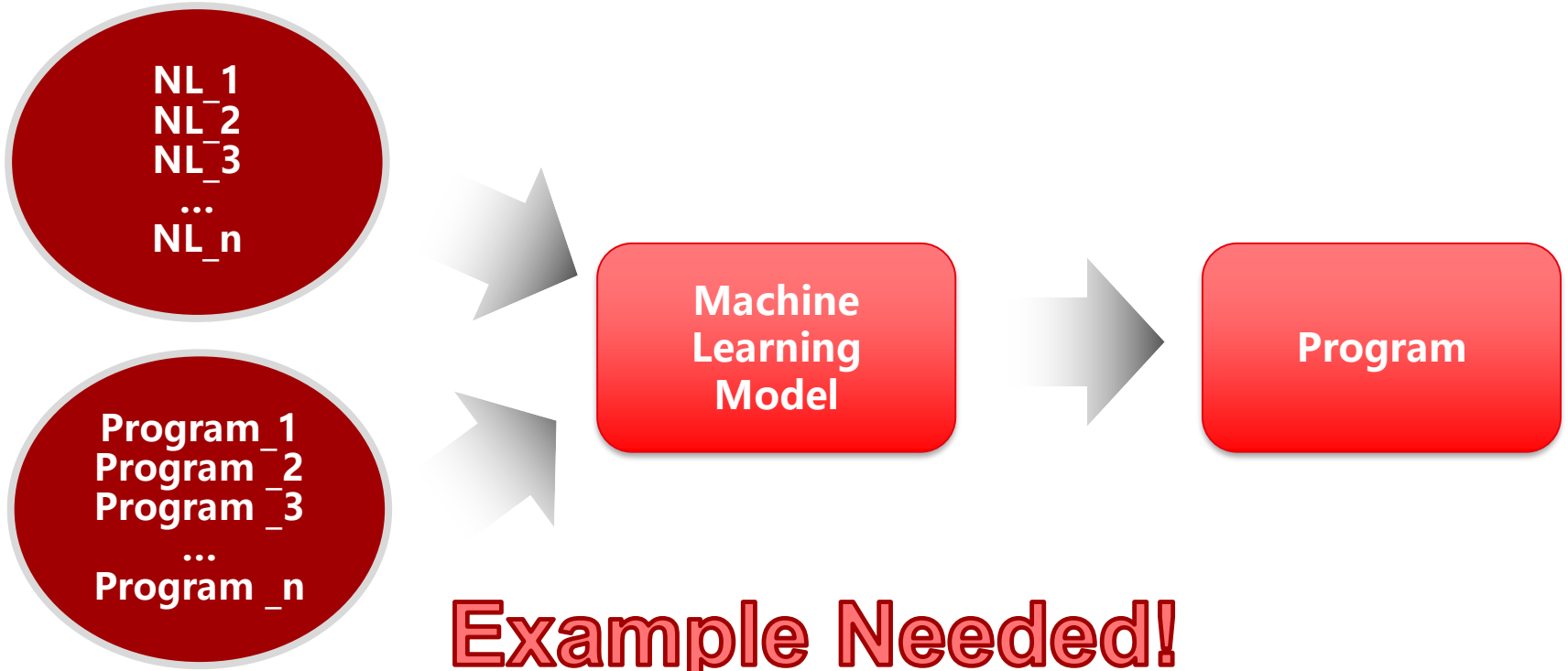
Not much adoption beyond experts

- Widely used in Clang, but rarely in outside project.

Example-Driven Program Synthesis

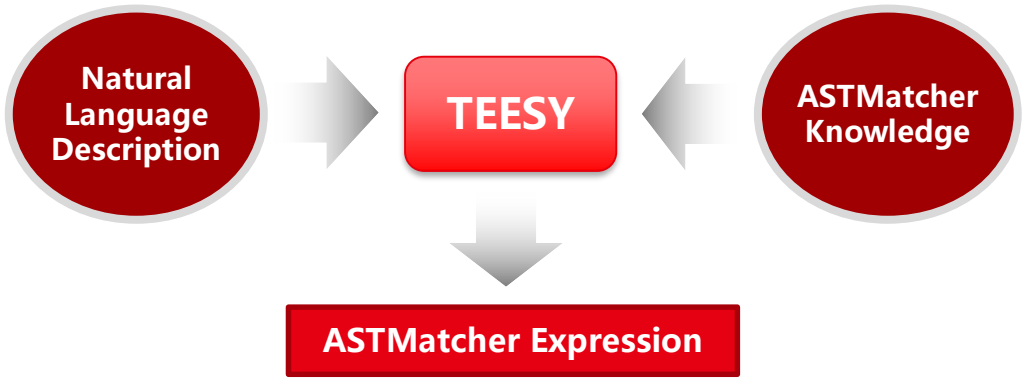


Machine Learning-Driven Program Synthesis



No example!

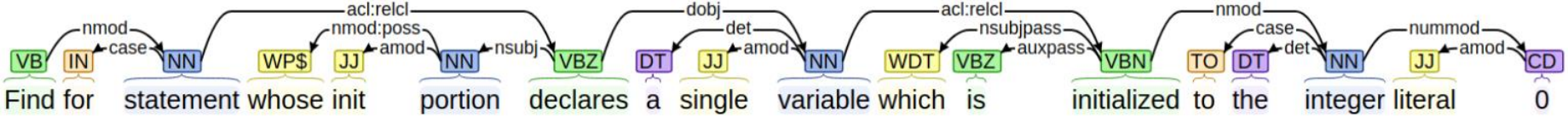
We propose TEESY



1 Training - Free

2 Dependency-Tree based co-evolvement Tech

Dependency structure



Dependency parsing

An automatic approach that analyzes binary asymmetrical relations (called dependency relations) between words within a sentence.

Dependency Tree

The result of dependency parsing.

Dependency Relation

- Dependent: a subordinate word.
- Governor: the word that dependent depends.
- Dependency type: relation between dependent and governor.

Challenges

Ambiguous matching

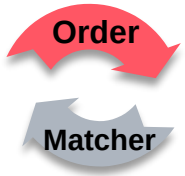
- “Declare” matches 113 matchers.
- Which one is correct?

Order mismatching

- User Intent: …. single variable…
- Possible matcher expression:
  - variable(Single)
  - Single(variable)

Training-free NLP-based synthesis

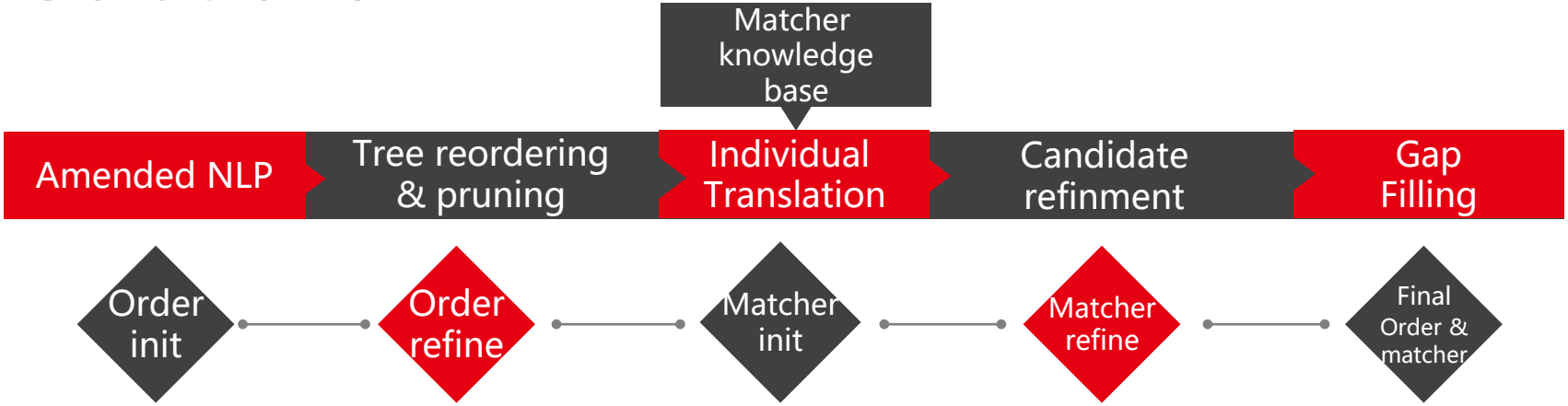
Tangled concerns of order and matcher



Terminology confusion

- User intent: Find for loops ...
- NLP: Find for loops …

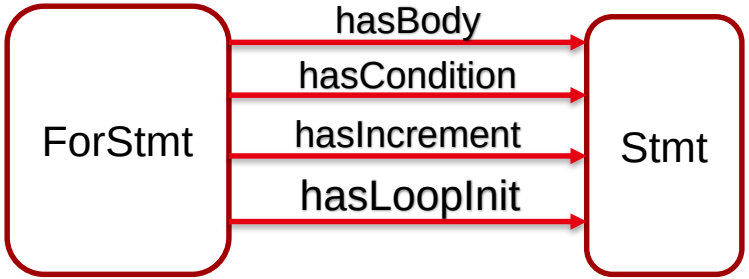
Solutions


Matcher knowledge base
Amended NLP
Tree reordering & pruning
Individual Translation
Candidate refinment
Gap Filling
Order init
Order refine
Matcher init
Matcher refine
Final Order & matcher
• Dependency analysis with light regulations
• Adjustment of relative clause
• Auxiliary leaves pruning
• Word translation
• Longest-match phrase trans
• Order-based type-driven focusing
• Scope-based prioritization
• Rule-directed gap filling
• Reorder-based gap filling
• Search-based gap filling

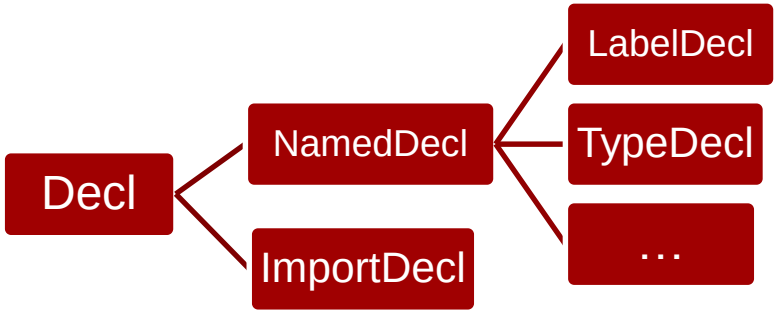
Matcher Knowledge Base Construction

All information of ASTMatcher API

Matcher	Category	Input Type	Return Type	Description
forStmt	Node	Matcher<ForStmt>	Matcher<Stmt>	Matches for statements.

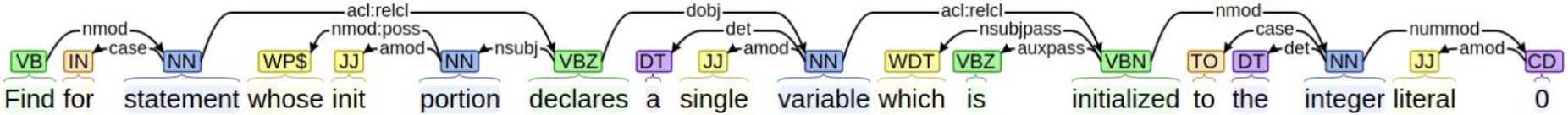


Matcher graph

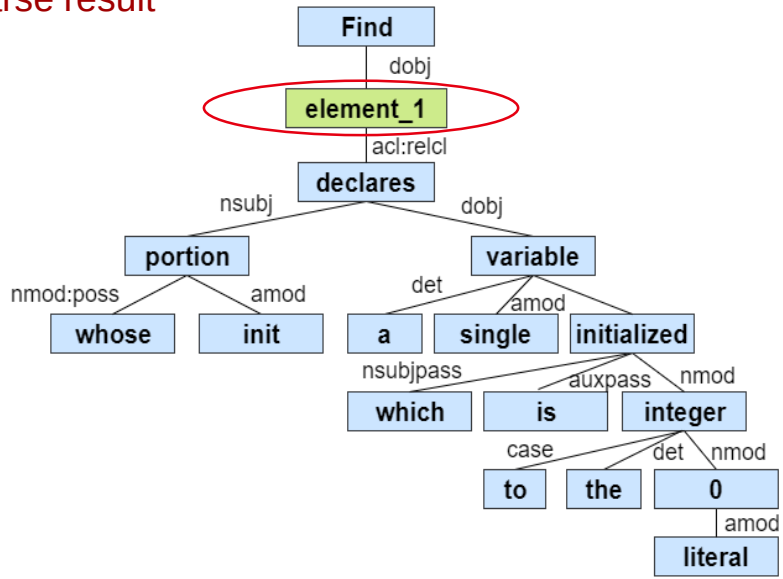


Type hierarchy

Amended NLP



Dependency parse result

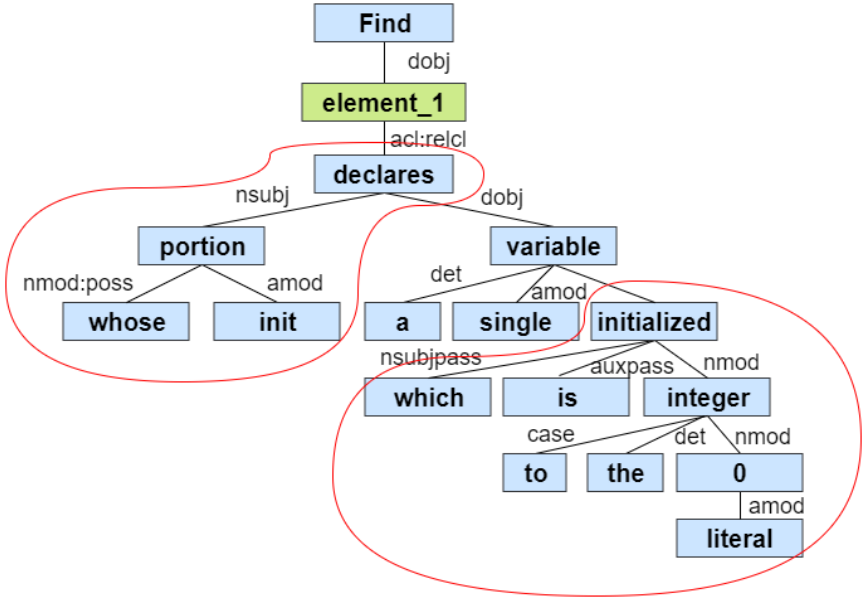


Dependency Tree

Tree Reordering and Pruning

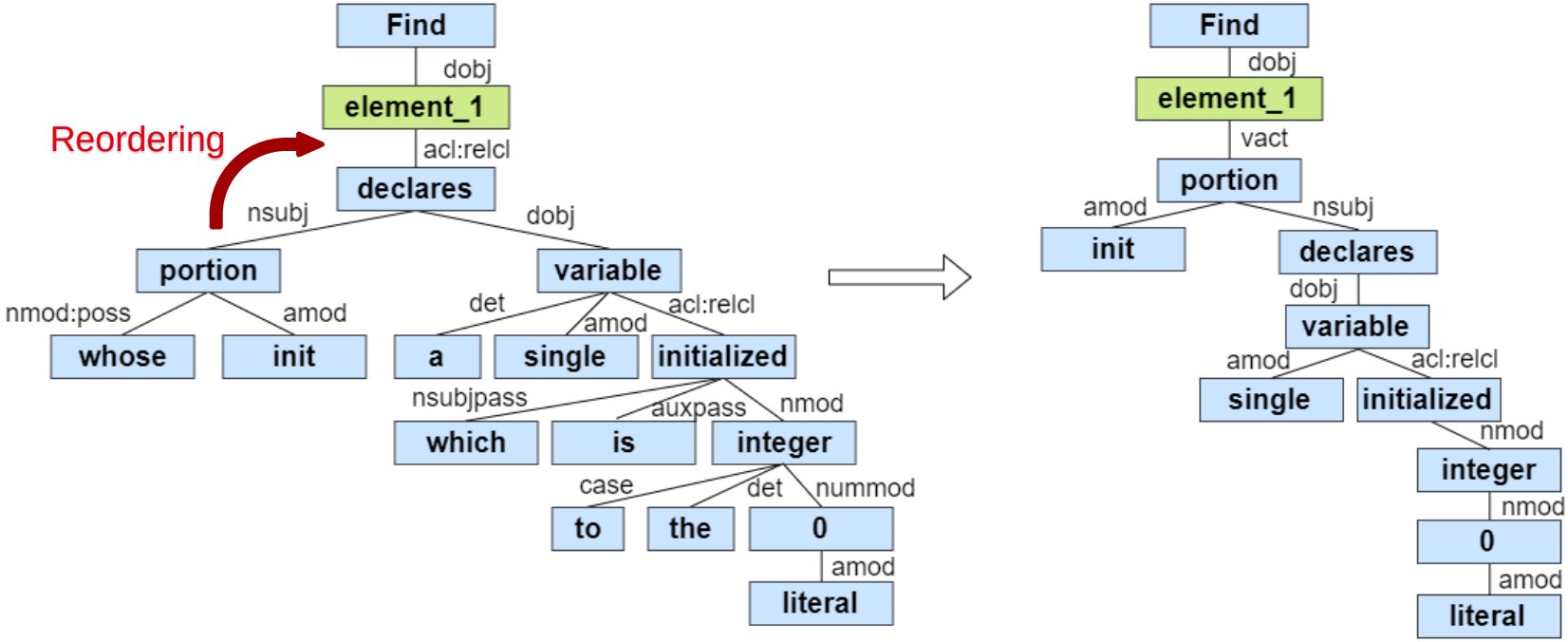
Relative Clause

- The NLP engines always link the dependency between the verb and the noun that clause modifies.
- The subject and object of clause will be the children nodes of that verb

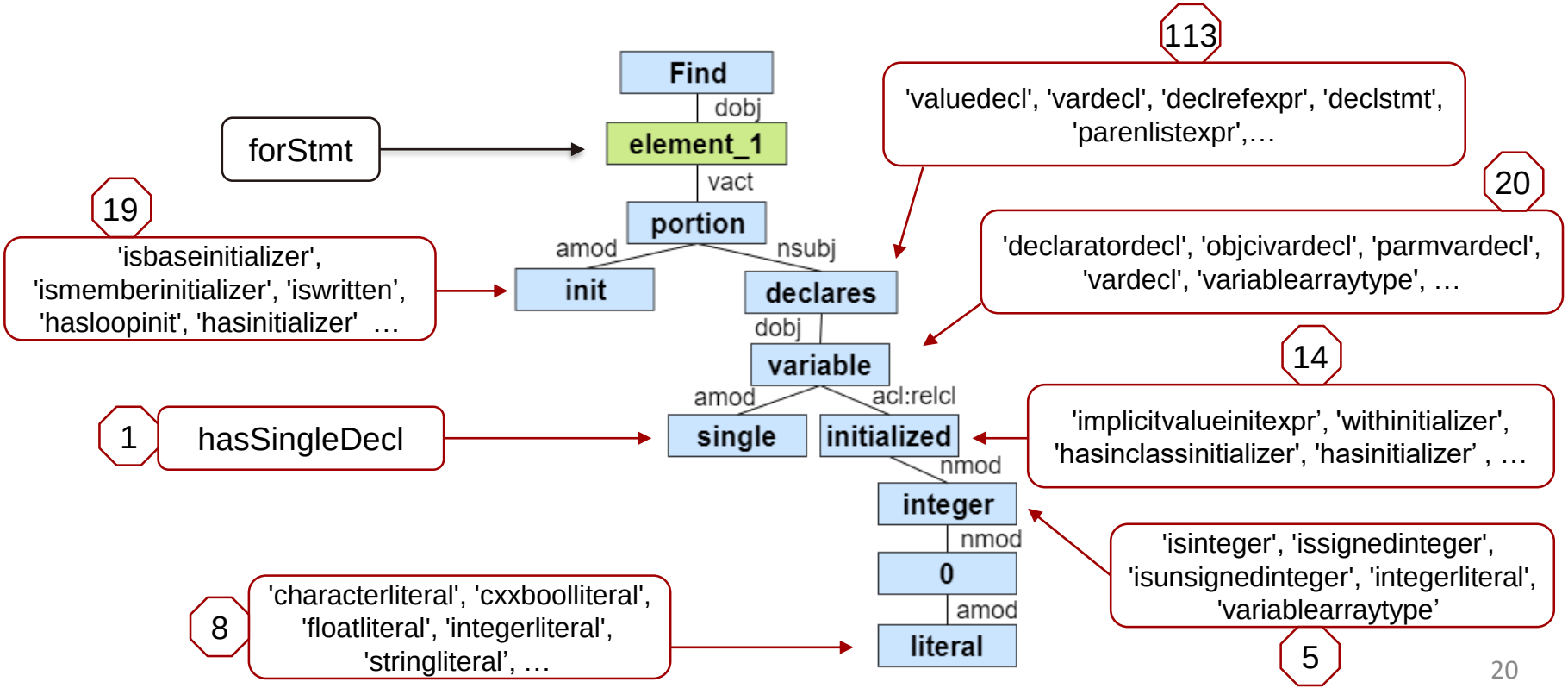


Dependency Tree

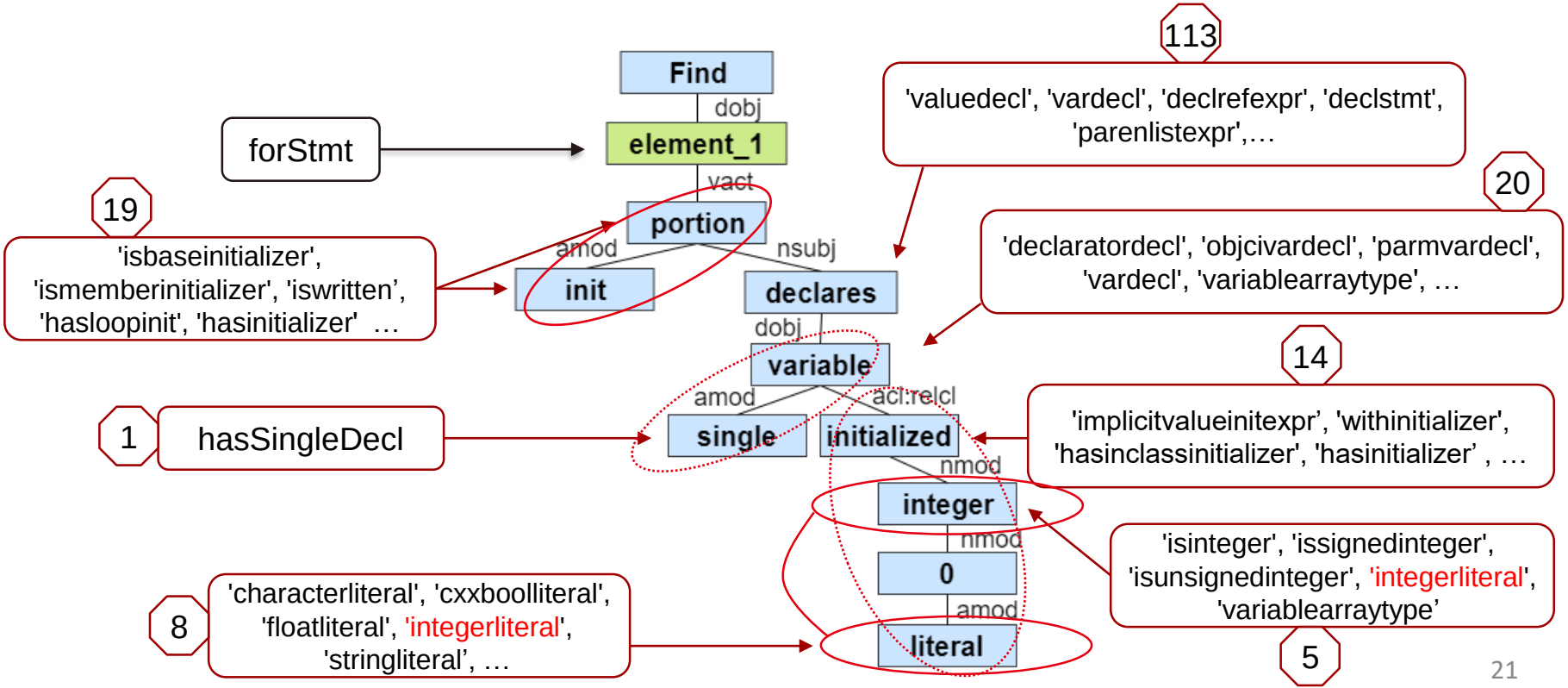
Tree Reordering and Pruning



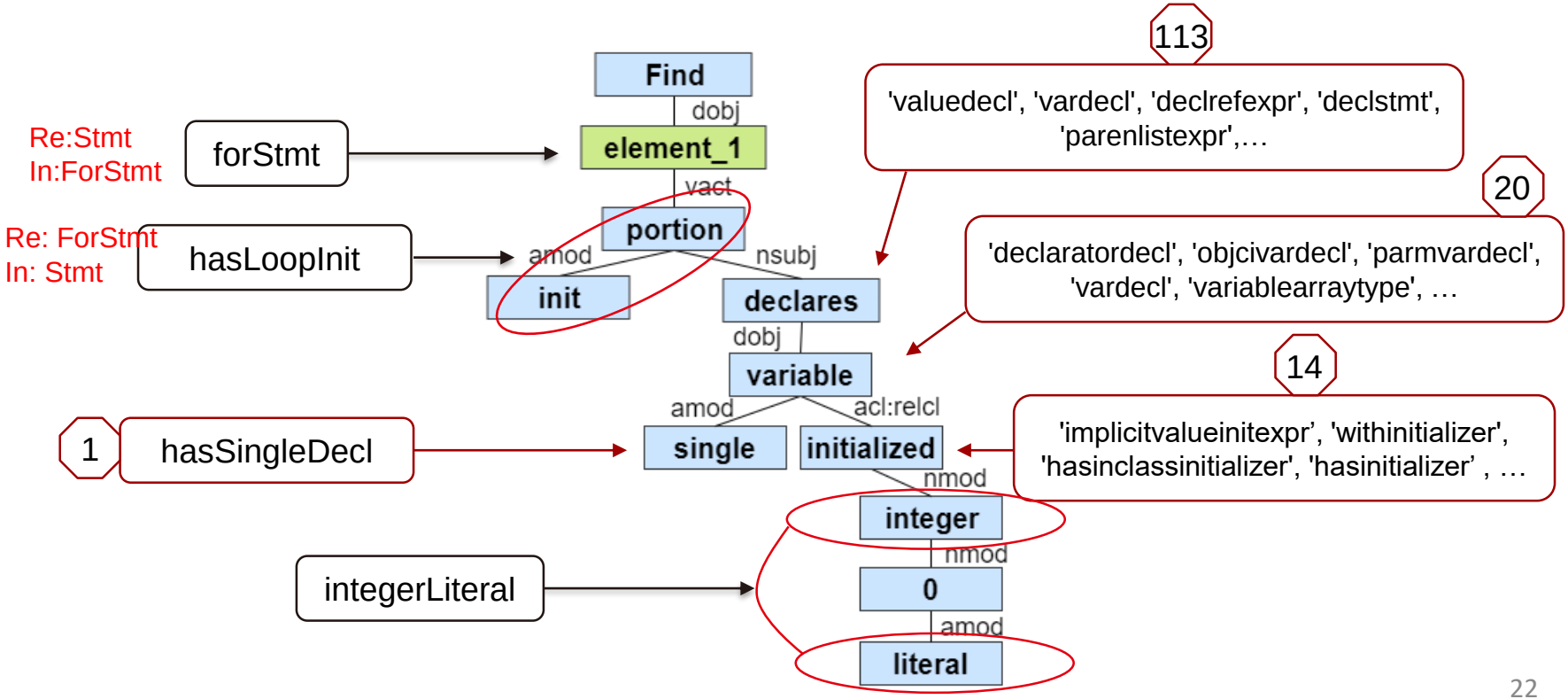
Individual Translation



Longest-match scheme



Order-based Type-driven Focusing



Scope-based prioritization

Scope

- The number of non-trivial words that the matcher description contains.

Scope = 3

varDecl: Matches variable declarations

Scope = 4

parmVarDecl: Matches parameter variable declarations

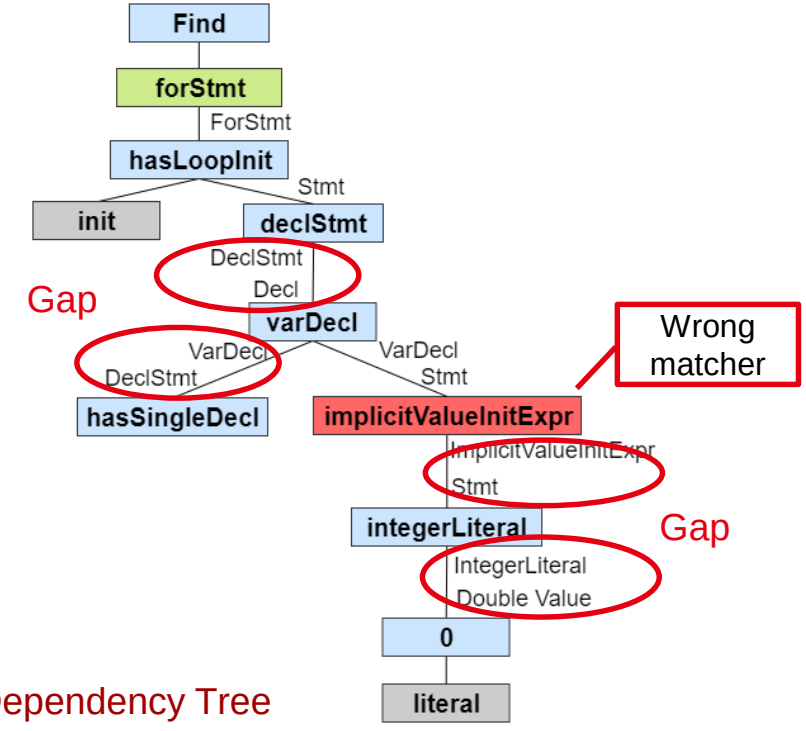
Scope-based prioritization

- TEESY computes the extra scope of every promising matcher.
- The smallest extra scope will be the most promising matchers.

The synthesis result

The correct expression

```
forStmt(
    hasLoopInit(
        declStmt(
            hasSingleDecl(
                varDecl(
                    hasInitializer(
                        integerLiteral(
                            equals(0))))))));
```



Dependency Tree

Combined Gap Filling

- Some verbs represent general relations
- But do not match with any matcher directly.

- A set of rules are created for those verbs

Rule-directed

Gap Filling

Find all the functions that **use** a particular global variable named “PI”.

Query with special verb “use”

```
decl – use --decl
```

```
decl --
 hasDecendant(
   declRefExpr(
     to)))
-- decl
```

Rules for “use”

Combined Gap Filling

- The adjective modifier could modify a node either as a parent node or as a child node.
  - Use type consistency to check all the matchers.
  - Reorder the adjective modifier if necessary.

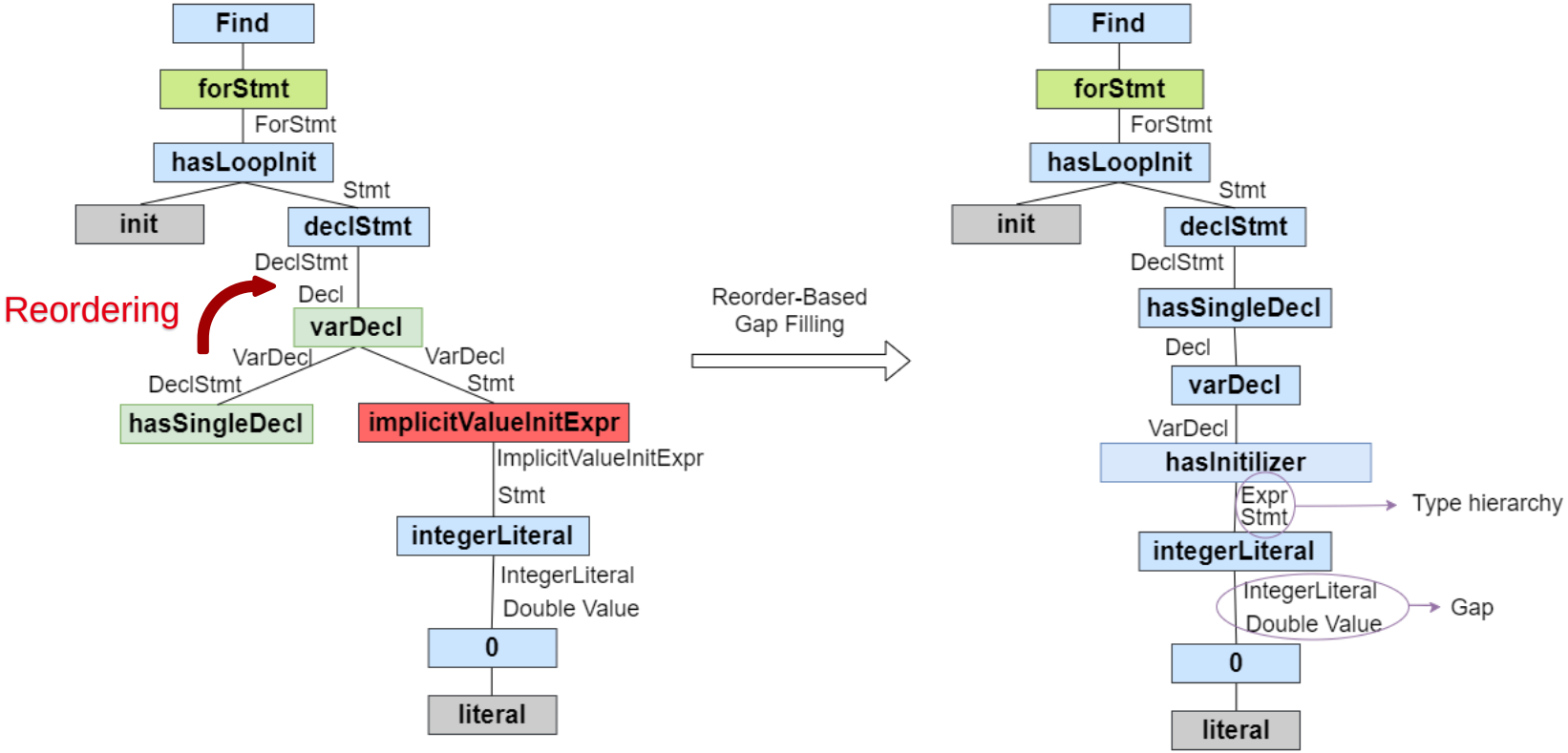
- Verbs
  - Rules

Rule-directed

Reorder-based

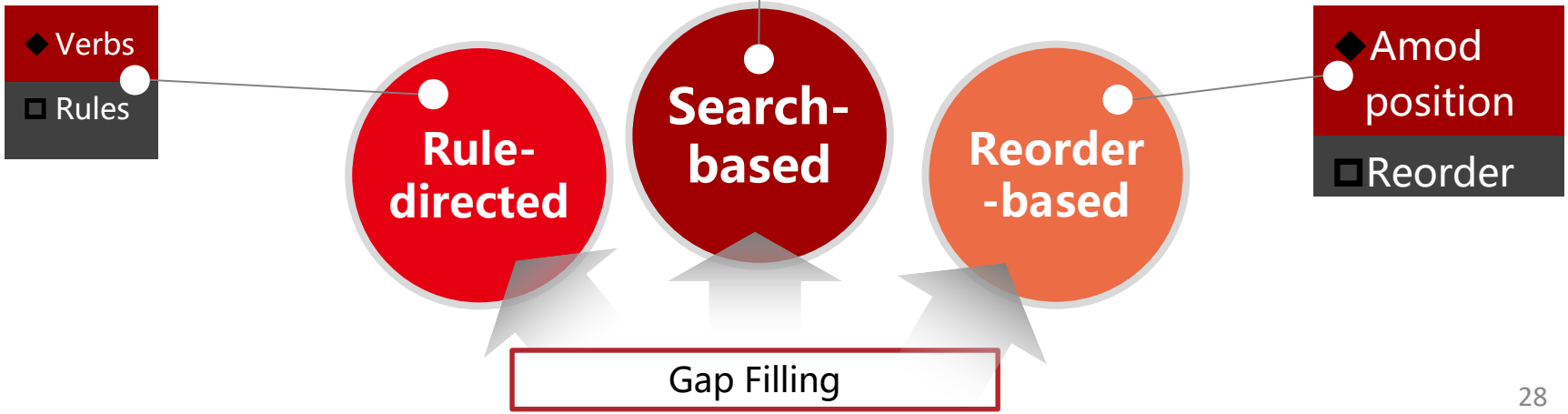
Gap Filling

Synthesis result at this stage

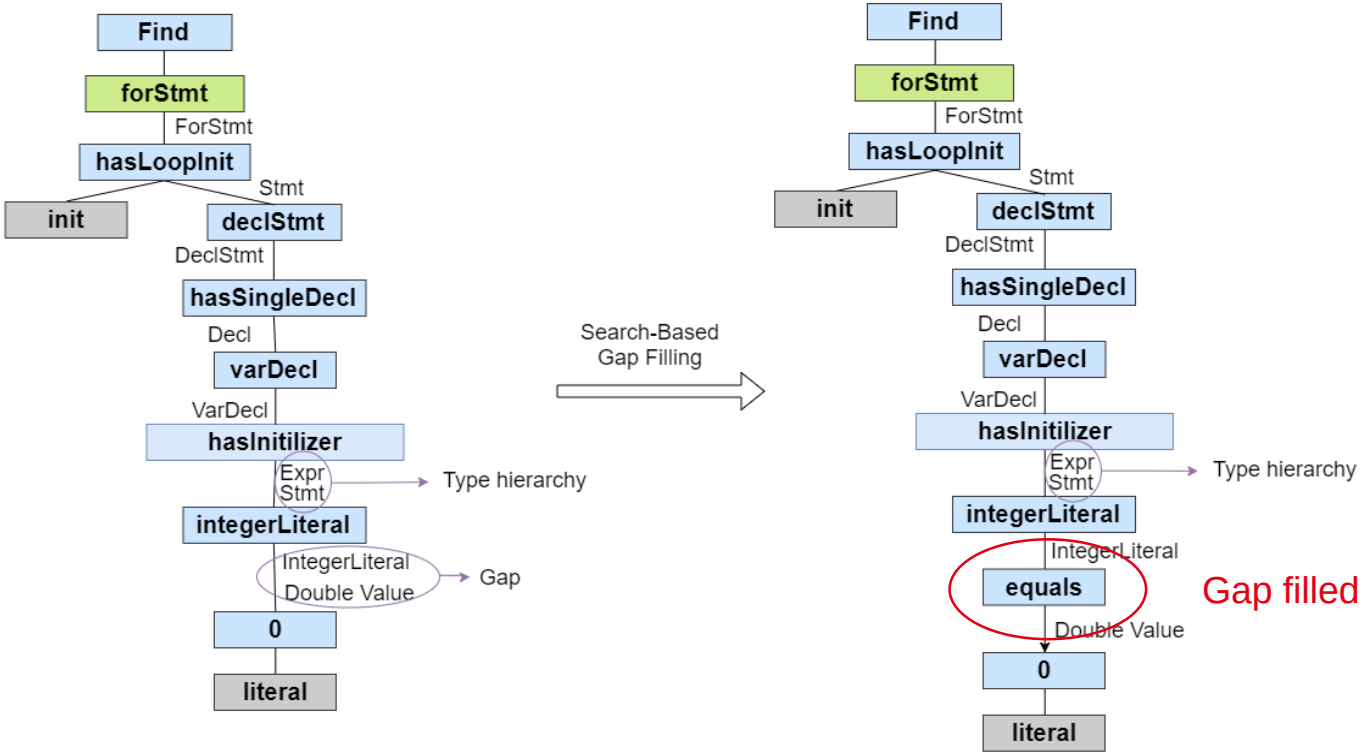


Combined Gap Filling

- The gap that need a connecting matcher
  - Use the breadth-first search on the matcher graph
  - Start from the expected argument type
  - Stop if it reaches the return type or upper limit of hops



Synthesis result at this stage



Evaluation

- ASTMatcher expression accuracy (Overall Accuracy)
- Individual matcher accuracy (Inner Accuracy)

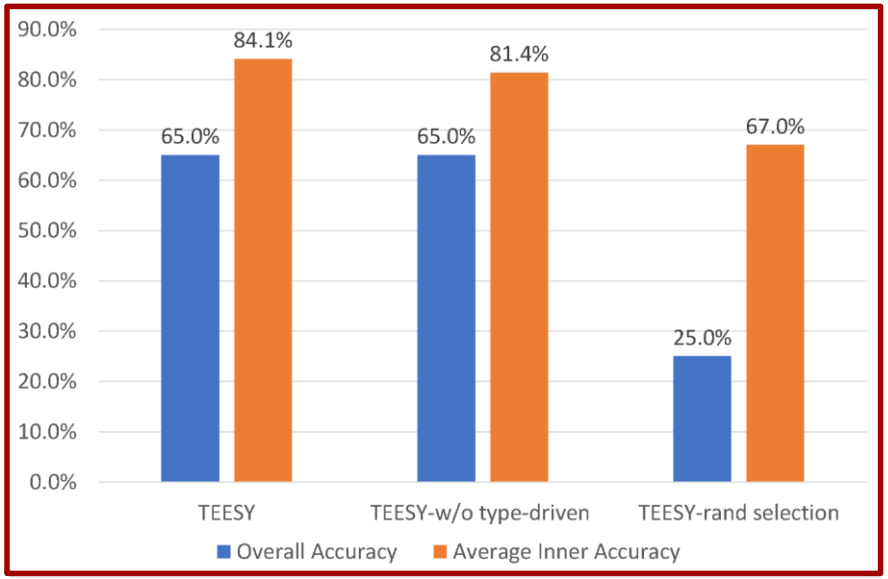
- TEESY – w/o type-driven
- TEESY – random decision
- TEESY step by step result

- Parse the English query to the dependency tree.
- Synthesis a complete matcher expression from dependency tree.

ASTMatcher Test Examples

No.	English description	Correct Matcher expression
1	Find for statement whose init portion declares a single variable which is initialized to the integer literal 0.	forStmt(hasLoopInit(declStmt(hasSingleDecl(varDecl(hasInitializer(integerLiteral(equals(0)))))));
2	Find all call expressions of a c++ class function named “func”.	cxxMemberCallExpr(callee(cxxMethodDecl(hasName(“func”))))
3	Find all the functions that use a particular global variable named “PI”.	functionDecl(hasDescendant(declRefExpr(to(varDecl(hasGlobalStorage(), hasName(“PI”))))))

Evaluation: Accuracy and Time



- **19.23 seconds**

The average time used to parse the English query.

- **0.19 seconds**

The average time used to synthesis the ASTMatcher expression.

Evaluation: Benefits of Individual Step

Id	Ground truth	Original tree	The number of correct matchers matched after each step						
			Word trans	Longest match	Order-based	Scope-based	Rule - directed	Reorder-based	Search-based
1	8	0	1	1	2	5	8	8	8
2	9	0	2	2	3	6	6	8	9
3	7	0	1	2	3	4	7	7	7
4	3	0	0	0	0	2	2	3	3
5	5	0	1	1	3	4	4	5	5
6	9	0	2	2	3	3	9	9	9
7	5	0	1	1	1	2	4	5	5
8	5	0	1	1	2	4	5	5	5
9	3	0	0	0	0	2	2	2	3
10	5	0	0	0	0	4	4	4	5

Special terms

Identify the most promising matcher

Rules applied

Evaluation: Error Analysis

- Consists with parent and child nodes.
- Has smaller scope

English description	Matcher expression	Error Expression	Reason
Find if statement whose condition is smaller than 10.	ifStmt(**hasCondition(** binaryOperator(hasOperatorName("<"), hasRHS(integerLiteral(equals(10))))))	ifStmt(**hasConditionVariableStatement(** binaryOperator(hasOperatorName("<"), hasRHS(integerLiteral(equals(10))))))	Scope-based prioritization

Evaluation: Error Analysis

- The search result
- No keywords assist

English description	Matcher expression	Error Expression	Reason
Find all c++ call expression of the c++ method named string_1	cxxMemberCallExpr(**callee(** cxxMethodDecl(hasName(string_1)	cxxMemberCallExpr(**thisPointerType(** cxxMethodDecl(hasName(string1)	Search-based gap filling

Evaluation: Error Analysis

- The dependency parser gives the wrong order

English description	Matcher expression	Error Expression	Reason
Find call expression which calls the function named string_1.	callExpr(callee(functionDecl(hasName(string1))))	callExpr(callee(**typedefNameDecl(** **functionType**(string1))))	Dependency parsing
Find call expression which call the function whose name is string_1.		(correct)	--

Discussions

Engineering considration

- Provide information feedback for error correction.
- Show other matchers to provide more choice.

NLP techniques

- StanfordCoreNLP is not the most cutting edge, but mature.
- As the techniques become more mature, the accuracy of TEESY could get further improved.

Conclusion

- TEESY is a Training-Free Natural Language-Based Synthesizer for ASTMatcher.
- TEESY uses dependency tree-based co-evolvement:
  Leverages deep semantic from NL dependency analysis, and synergizes NL processing (NLP) and techniques on API data types and domain knowledge.
- TEESY is relative to a broad range of users for the popularity of Clang/LLVM.
- May provide some insights in improving the productivity of other program analysis tools.