CE 815 – Secure Software Systems

ML-Based Vulnerability Detection Methods (Hoppity)

Mohammad Haddadian/Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide. Thanks to Mohammad Haddadian for the help on the slides.

Introduction

- Vulnerability detection as first step
- Then, Vulnerability repair

- Compiler VS Interpreter
- Vulnerability VS Bug
- Security VS non-Security

HOPPITY: Learning Graph Transformations to Detect and Fix Bugs in Programs, ICLR 2020.

Problem

Source-code analysis is:

- Undecidable
- Noisy
- Rules are hand written
- Tailored to specific code bases / bug patterns

Javascript Challenges

- Incorrect operators
- Incorrect identifiers
- Accessing undefined properties
- Mishandling variable scopes
- Type incompatibilities

Example

```
function clearEmployeeListOnLinkClick(){
  document.querySelector("a").addEventListener("click",
    function(event){
      document.querySelector("ul").InnerHTML = "";
    }
  );
}
```

(a) `InnerHTML` should have been `innerHTML`.

```
if (matches) {
  return {
    episode: Number(matches.groups.episode),
    hosts: matches.groups.hosts.split(/([,&]+|\sand\s)/).
              map(el => S(el).trim().s)
  };
}
```

(b) Highlighted parentheses should have been removed.

```
module.exports = function (grunt) {
  grunt.initConfig({
    execute: {...}, copy: {...}, checktextdomain: {...}
    wp_readme_to_markdown: {...}, makepot: {...}})
  ...
  grunt.registerTask('default',['wp_readme_to_markdown'
    ,'makepot','execute','checktextdomain'])
};
```

(c) `copy` function should have also been included in the highlighted list.

```
export default {
  computed: {
    level () {
      return dictMap.skillLevel[
        parseInt((this.value === 0 ? 1 : this.value)/20)];
    }
  },...
}
```

(d) `parseInt` should have been removed because === implies `this.value` is an integer.

Solution

Leverage large amounts of Javascript fixes on Github to locate and repair bugs

Steps

- Represent source code
- Represent fixes
- Learning

Model

- Problem of detecting and repairing bugs in programs is a structured prediction problem on a graph-based representation of programs.

$P(g_1, g_2, \ldots, g\_T \mid G_0, \theta)$, where θ is the model parameter

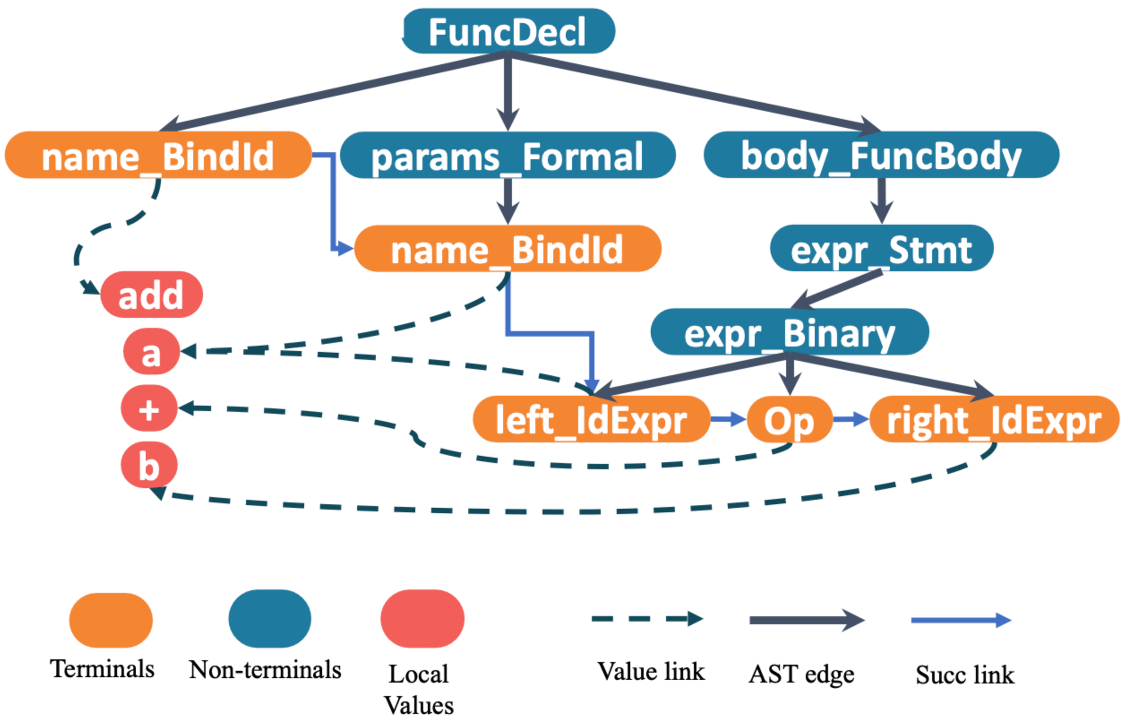
$$p(g_{fix}|g_{bug};\theta) = p(g_1|g_{bug};\theta)p(g_2|g_1;\theta)\ldots p(g_{fix}|g_{T-1};\theta)$$

Goal

- 1- function add (a) {a + b;} Buggy
- 2- function add (a, b) {a + b;} Step 1
- 3- function add (a, b) { return a + b;} Step 2

Source code representation

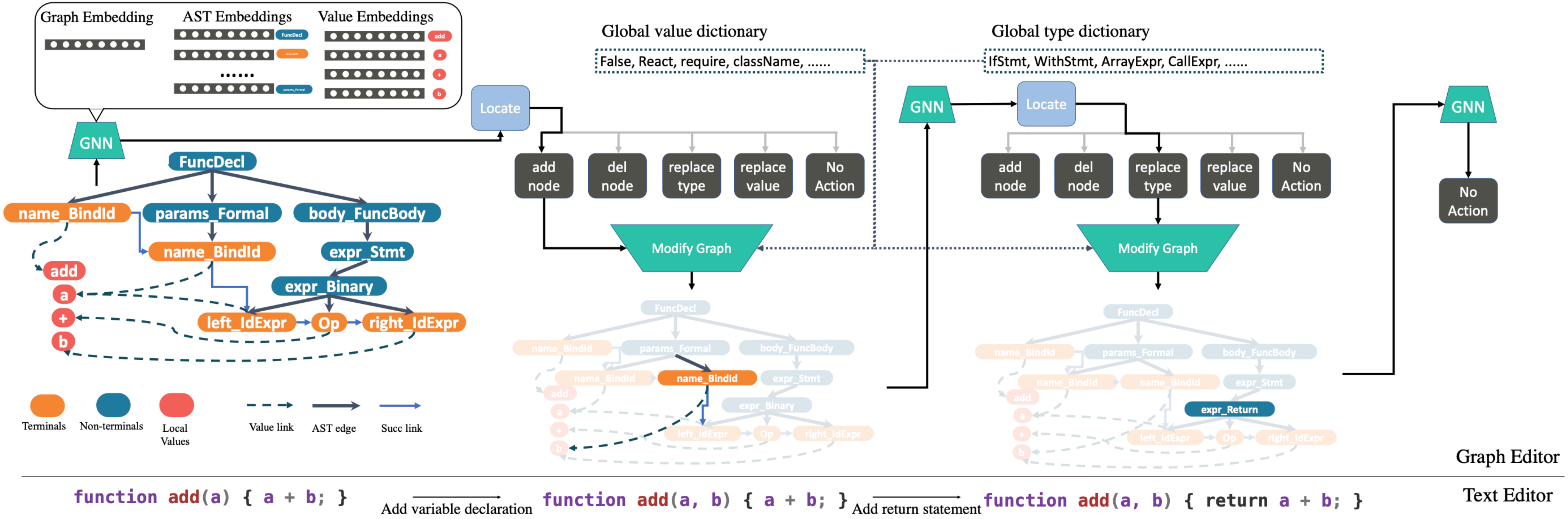
- AST
- ValueLink

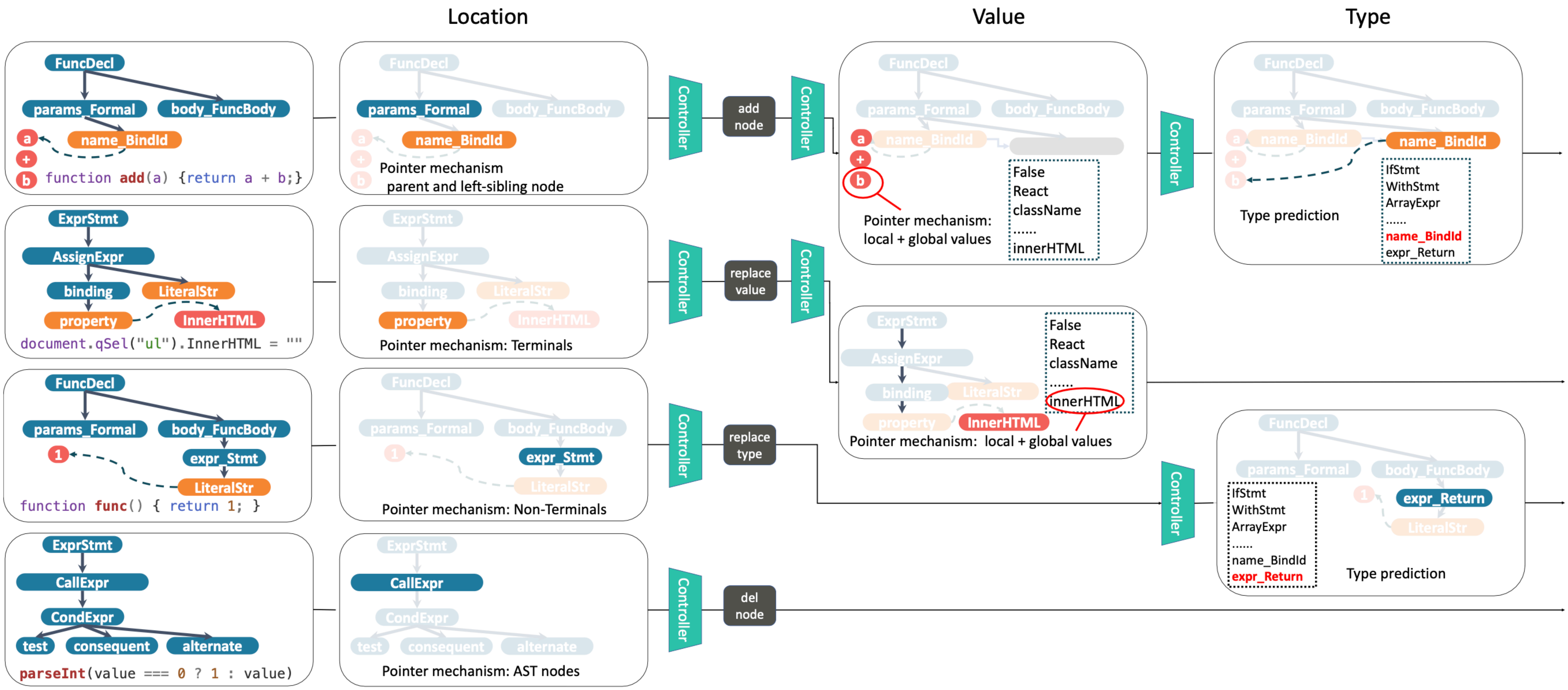



```
function add(a) { a + b; }
```

Fix representation

- Graph Edits



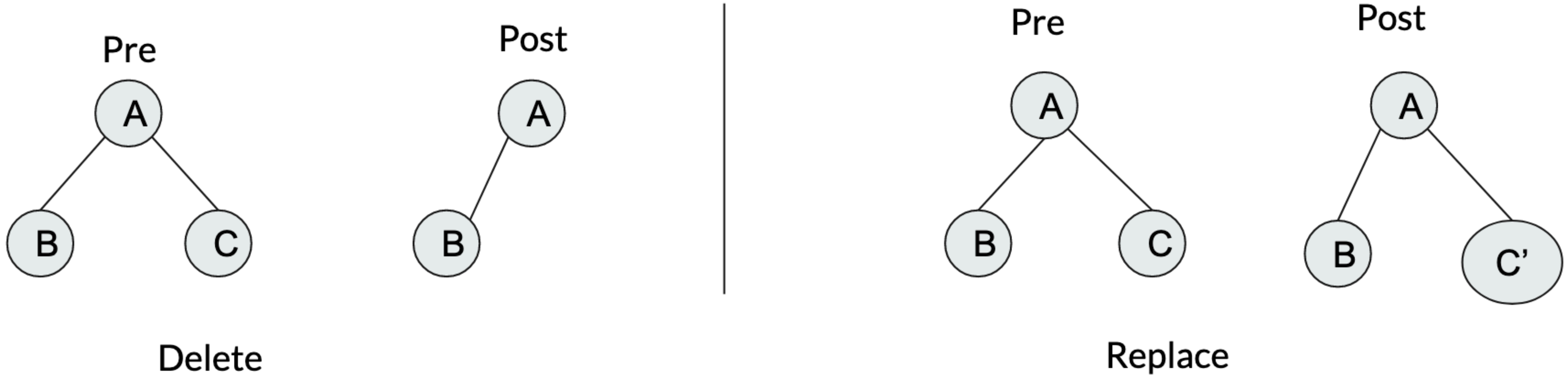


Location
Value
Type
FuncDecl
params_Formal
body_FuncBody
name_BindId
function add(a) {return a + b;}
Pointer mechanism
parent and left-sibling node
Controller
add node
Pointer mechanism:
local + global values
False
React
className
......
innerHTML
Type prediction
IfStmt
WithStmt
ArrayExpr
name_BindId
expr_Return
ExprStmt
AssignExpr
binding
LiteralStr
property
InnerHTML
document.qSel("ul").InnerHTML = ""
Pointer mechanism: Terminals
replace value
Pointer mechanism: local + global values
expr_Stmt
function func() { return 1; }
Pointer mechanism: Non-Terminals
replace type
CallExpr
CondExpr
test
consequent
alternate
parseInt(value === 0 ? 1 : value)
Pointer mechanism: AST nodes
del node

[HOPPITY]

Low level primitives

- Location
- Value
- Type



Low level primitives: Value

Choose from either the values appearing in the current file (local value table), or a collection of global values that are common for the specific language

Let D_{val} be the global dictionary of commonly used leaf-node values in the language, where each item $i \in D$ is associated with a vector representation:

$\vec{i_v} \in R^d$

Low level primitives: Type

- As the total possible number of types is finite and fixed for a given language, the type prediction is simply a multi-class classification problem.
- But utilize the AST grammar checker with contextual information to prune the output space.

Graph edit operators

add node

del node

replace type

replace value

No Action

Anatomy of a graph edit

“replace_val”

1. Predict Location
2. Predict Value

“replace_type”

1. Predict Location
2. Predict Type

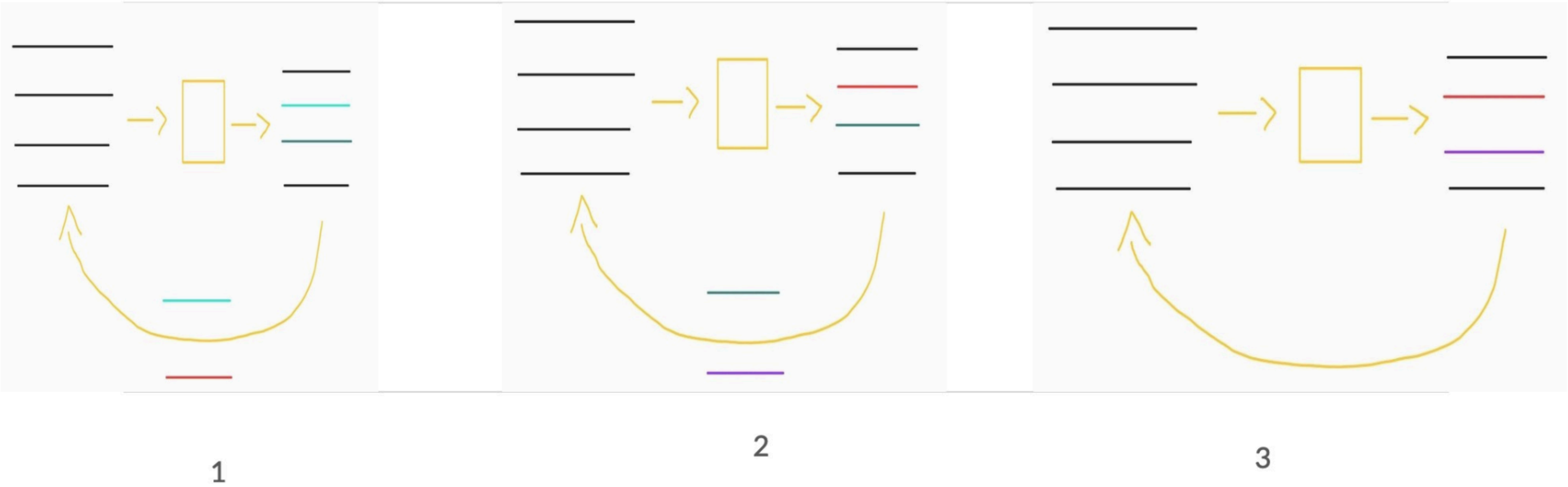
“remove”

1. Predict Location

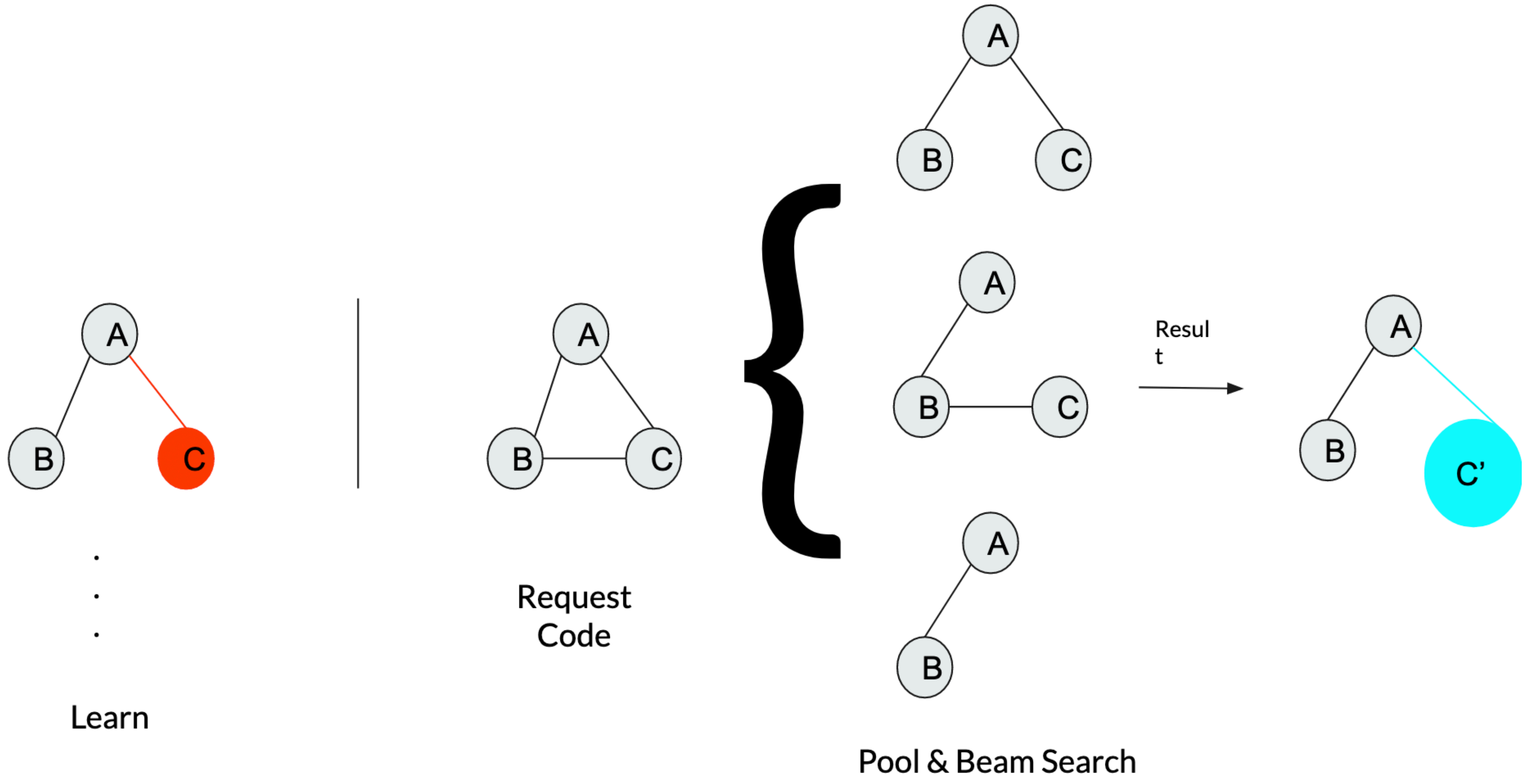
“add”

1. Predict Location
2. Predict Value
3. Predict Type
4. Predict Child Number

Graph transformation



Inference



A
B
C
.
.
.
Learn
A
B
C
Request
Code
A
B
C
A
B
C
A
B
Pool & Beam Search
Resul
t
A
B
C'

Dataset

- OneDiff (just one change)
- ZeroOneDiff (zero or one edit)
- ZeroOneTwoDiff (zero, one or two edits)

	ADD	REP_TYPE	REP_VAL	DEL	total
train	6,473	1,864	251,097	31,281	290,715
validate	790	245	31,357	3,957	36,349
test	796	233	31,387	3,945	36,361

Table 1: Statistic of `OneDiff` dataset. See appendix for more information of other dataset.

Evaluation

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
TOTAL	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
`ADD`	52.9	39.2	69.6	51.4	70.6	65.7	55.1	76.8	68.5
`REP_VAL`	23.4	11.9	33.3	18.5	31.7	53.0	28.8	-	-
`REP_TYPE`	71.7	52.4	73.0	52.8	79.4	-	-	74.7	61.0
`DEL`	39.6	24.8	44.0	27.5	45.8	-	-	-	-
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 2: Evaluation of model on the `OneDiff` dataset: accuracy (%).

Evaluation (cont.)

Type	GGNN-Rep	GGNN-Cls	HOPPITY
Top-1	53.2%	**99.6%**	90.0%
Top-3	85.8%	**99.6%**	94.8%

Table 3: `REP_TYPE` accuracies with location+op.

Value	GGNN-Rep	GGNN-RNN	HOPPITY
Top-1	63.8%	60.3%	**69.1%**
Top-3	67.6%	63.6%	**73.4%**

Table 4: `REP_VAL` accuracies with location+op.

	Top-1	Top-3
HOPPITY	**67.7%**	**73.3%**
SequenceR	64.2%	68.6%

Table 5: Overall OneDiff accuracy with location.

Bug Type	Amount	TAJS	HOPPITY
Undefined Property	7	0	1
Functional Bug	11	0	3
Refactoring	12	0	1
Total	30	0	5

Table 6: Comparison with TAJS.

Acknowledgments

- [HOPPITY] HOPPITY: Learning Graph Transformations to Detect and Fix Bugs in Programs, ICLR 2020.