RESEARCH ON CODE CLONE

Preliminary work on AST and CFG

Dinglong Li	PB13011077
Xiao Luo	PB13207088
Yanzhao Wu	PB13011059

OUTLINE

- Background
- Current Research
- Our Ideas
- Demo
- Future Work

Background

CODE CLONE

is a computer programming term for a sequence of source code that occurs more than once.

The automated process of finding clones in source code is called

CLONE DETECTION

Background

Code Plagiarisms

is a persistent problem in many university courses, particularly those with programming assignments.[1]

[1]: Verco, K. L., & Wise, M. J. (1996). Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. ACSE, 96.

Background

Copy & Paste Codes

Recent studies have shown that large software suites contain significant amounts of replicated code. It is assumed that some of this replication is due to copy-and-paste activity and that a significant proportion of bugs in operating systems are due to copy- paste errors. [1]

[1]: Li, Z., Lu, S., Myagmar, S., & Zhou, Y. (2006). CP-Miner: Finding copy-paste and related bugs in large-scale software code. Software Engineering, IEEE Transactions on, 32(3), 176-192.

Background

Similarity and Differences

Code Plagiarisms

Copy & Paste Codes

Current Research

Widely Accepted Standard

Lv.1 Almost the same code(except for some characters)

Lv.2 Different variable names

Lv.3 Switches of lines and some structure changes

Lv.4 Codes perform same function

Current Research

- String Based
- Token Based
- Token-Tree Based
- Data Mining

Current Research

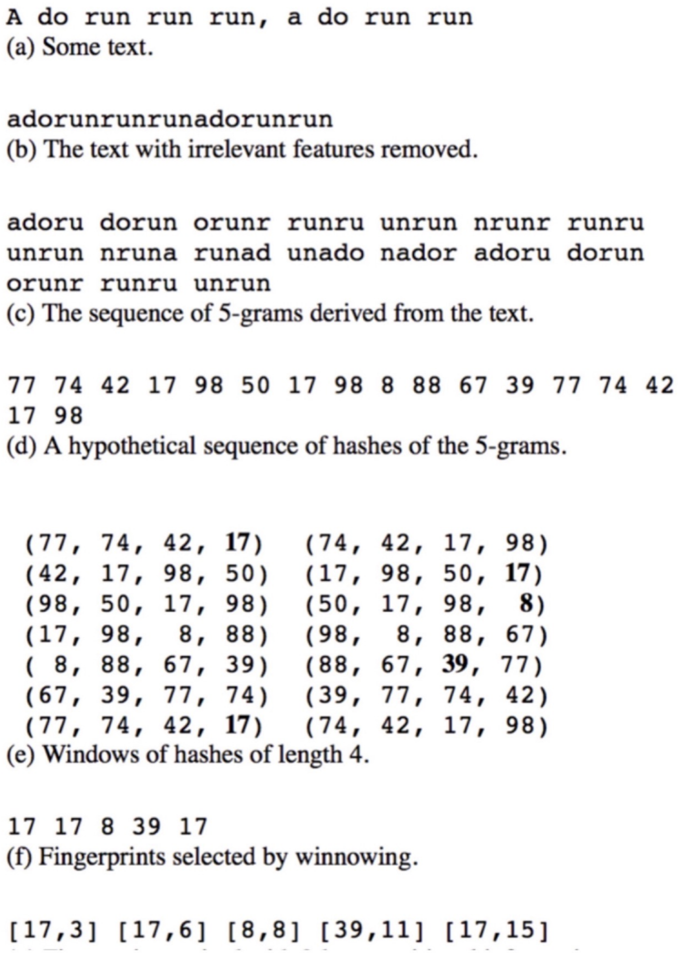
String based(moss)

K-gram

Finger print

Winnowing

$$d = \frac{2}{w+1}$$



```
A do run run run, a do run run
```

(a) Some text.

```
adorunrunrunadorunrun
```

(b) The text with irrelevant features removed.

```
adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun
```

(c) The sequence of 5-grams derived from the text.

```
77 74 42 17 98 50 17 98 8 88 67 39 77 74 42
17 98
```

(d) A hypothetical sequence of hashes of the 5-grams.

```
(77, 74, 42, 17)  (74, 42, 17, 98)
(42, 17, 98, 50)  (17, 98, 50, 17)
(98, 50, 17, 98)  (50, 17, 98,  8)
(17, 98,  8, 88)  (98,  8, 88, 67)
( 8, 88, 67, 39)  (88, 67, 39, 77)
(67, 39, 77, 74)  (39, 77, 74, 42)
(77, 74, 42, 17)  (74, 42, 17, 98)
```

(e) Windows of hashes of length 4.

```
17 17 8 39 17
```

(f) Fingerprints selected by winnowing.

```
[17,3] [17,6] [8,8] [39,11] [17,15]
```

[1] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S., & Bier, L. (1998, November). Clone detection using abstract syntax trees. In Software Maintenance, 1998. Proceedings., International Conference on (pp. 368-377). IEEE.

Current Research

Token based(JPlag、CC-Finder)

Convert code stream to token stream

JPlag[1] LCS

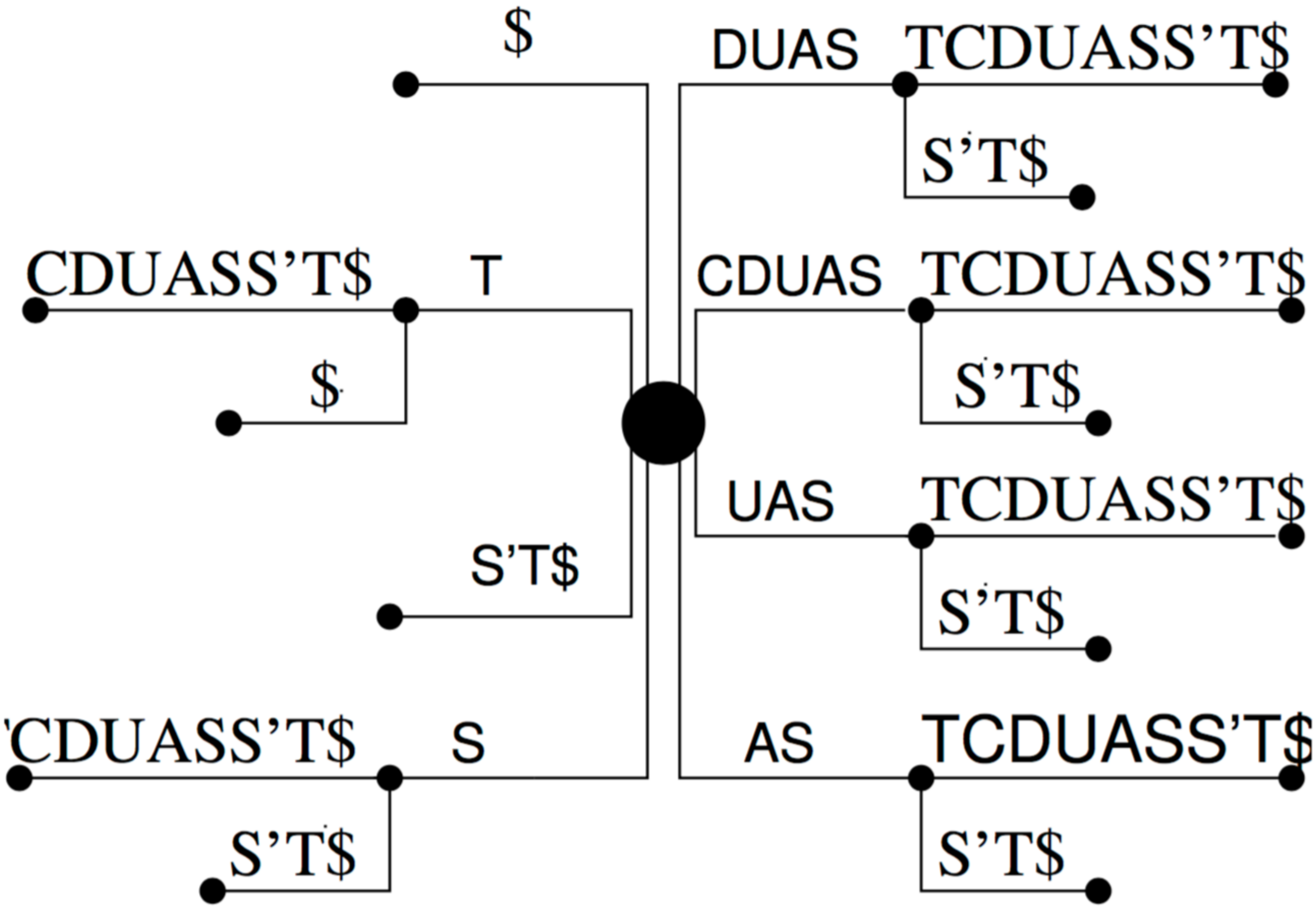
$$sim(A,B) = \frac{2 * coverage(tiles)}{(|A| + |B|)}$$

$$coverage(tiles) = \sum_{match(a,b,length) \in tiles} length$$

[1] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S., & Bier, L. (1998, November). Clone detection using abstract syntax trees. In Software Maintenance, 1998. Proceedings., International Conference on (pp. 368-377). IEEE.

[2] Koschke, R., Falke, R., & Frenzel, P. (2006, October). Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006. WCRE'06. 13th Working Conference on (pp. 253-262). IEEE.

[3] Koschke, R., Falke, R., & Frenzel, P. (2006, October). Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006. WCRE'06. 13th Working Conference on (pp. 253-262). IEEE.


$
DUAS
TCDUASS'T$
S'T$
CDUASS'T$
T
CDUAS
TCDUASS'T$
$
S'T$
UAS
TCDUASS'T$
S'T$
S'T$
'CDUASS'T$
S
AS
TCDUASS'T$
S'T$
S'T$

Current Research

Token based(JPlag、CC-Finder)

Convert code stream to token stream

CC-Finder[2][3]: suffix tree

[1] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S., & Bier, L. (1998, November). Clone detection using abstract syntax trees. In Software Maintenance, 1998. Proceedings., International Conference on (pp. 368-377). IEEE.

[2] Koschke, R., Falke, R., & Frenzel, P. (2006, October). Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006. WCRE'06. 13th Working Conference on (pp. 253-262). IEEE.

[3] Koschke, R., Falke, R., & Frenzel, P. (2006, October). Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006. WCRE'06. 13th Working Conference on (pp. 253-262). IEEE.

Data Mining(CP-Miner)

Frequent subsequence mining[1]
Token stream
DataBase

[1]: Li, Z., Lu, S., Myagmar, S., & Zhou, Y. (2006). CP-Miner: Finding copy-paste and related bugs in large-scale software code. Software Engineering, IEEE Transactions on, 32(3), 176-192.

Current Research

Baxter

Hash

Equivalence Class

$$Similarity = \frac{2S}{2S + L + R}$$

[1] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S., & Bier, L. (1998, November). Clone detection using abstract syntax trees. In Software Maintenance, 1998. Proceedings., International Conference on (pp. 368-377). IEEE.

DECKARD

Characteristic Vectors

Vector Clustering

[1]: Jiang, L., Misherghi, G., Su, Z., & Glondu, S. (2007, May). Deckard: Scalable and accurate tree-based detection of code clones. In Proceedings of the 29th international conference on Software Engineering (pp. 96-105). IEEE Computer Society.

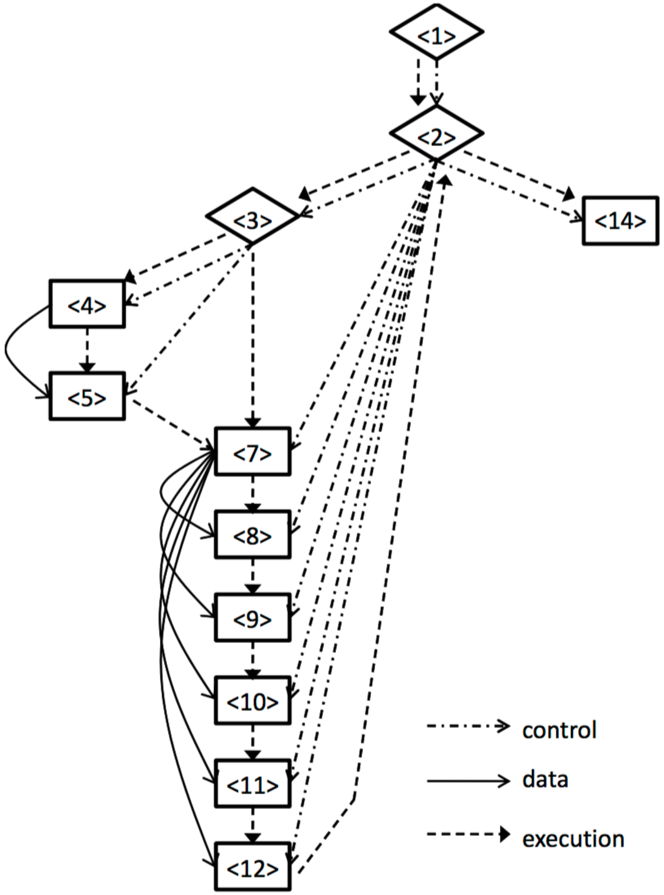
Current Research

PDG

Subgraph isomorphism

NPC

Heuristic



[1]Higo, Y., & Kusumoto, S. (2011, March). Code clone detection on specialized PDGs with heuristics. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on (pp. 75-84). IEEE.

Current Research

String-based: Lv1 Effective

Token-based: Lv2 Large-Scale

Token-Tree based: Lv3

Data Mining: new tendency

Current Research

CRITERION

ACCURACY

EFFECTIVE

DETECT LEVEL

Our Ideas

our ideas

AST

HASH

MATCHING

SUM-WEIGHTED SIMILARITY

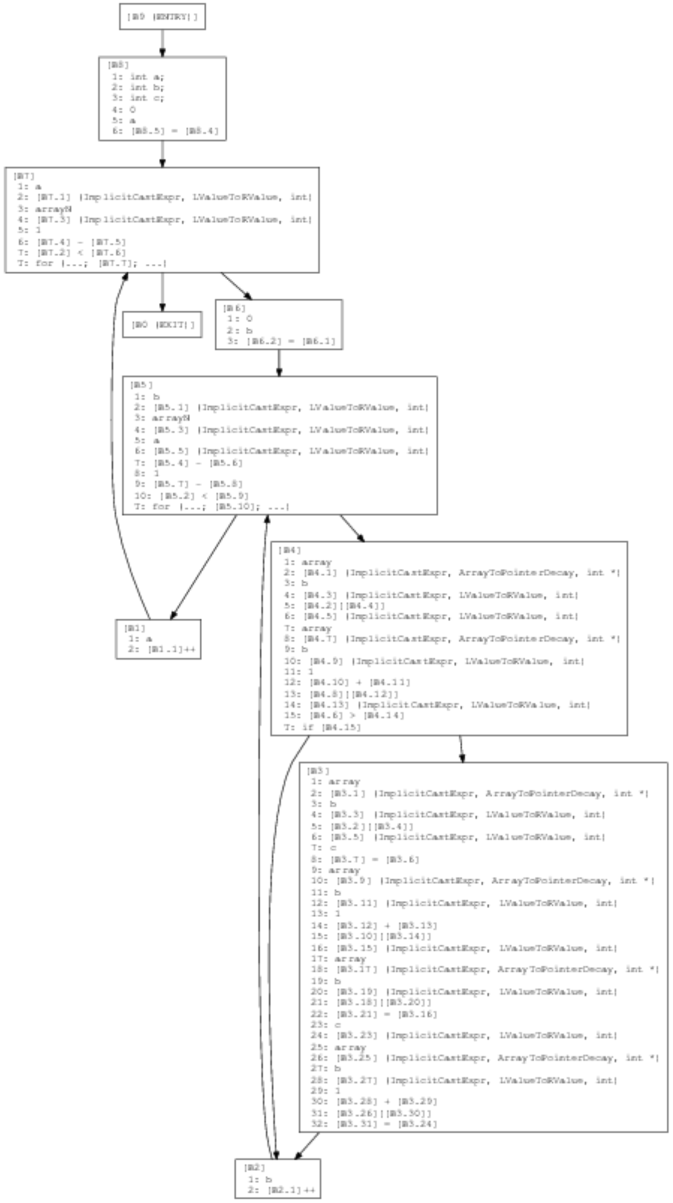
our ideas

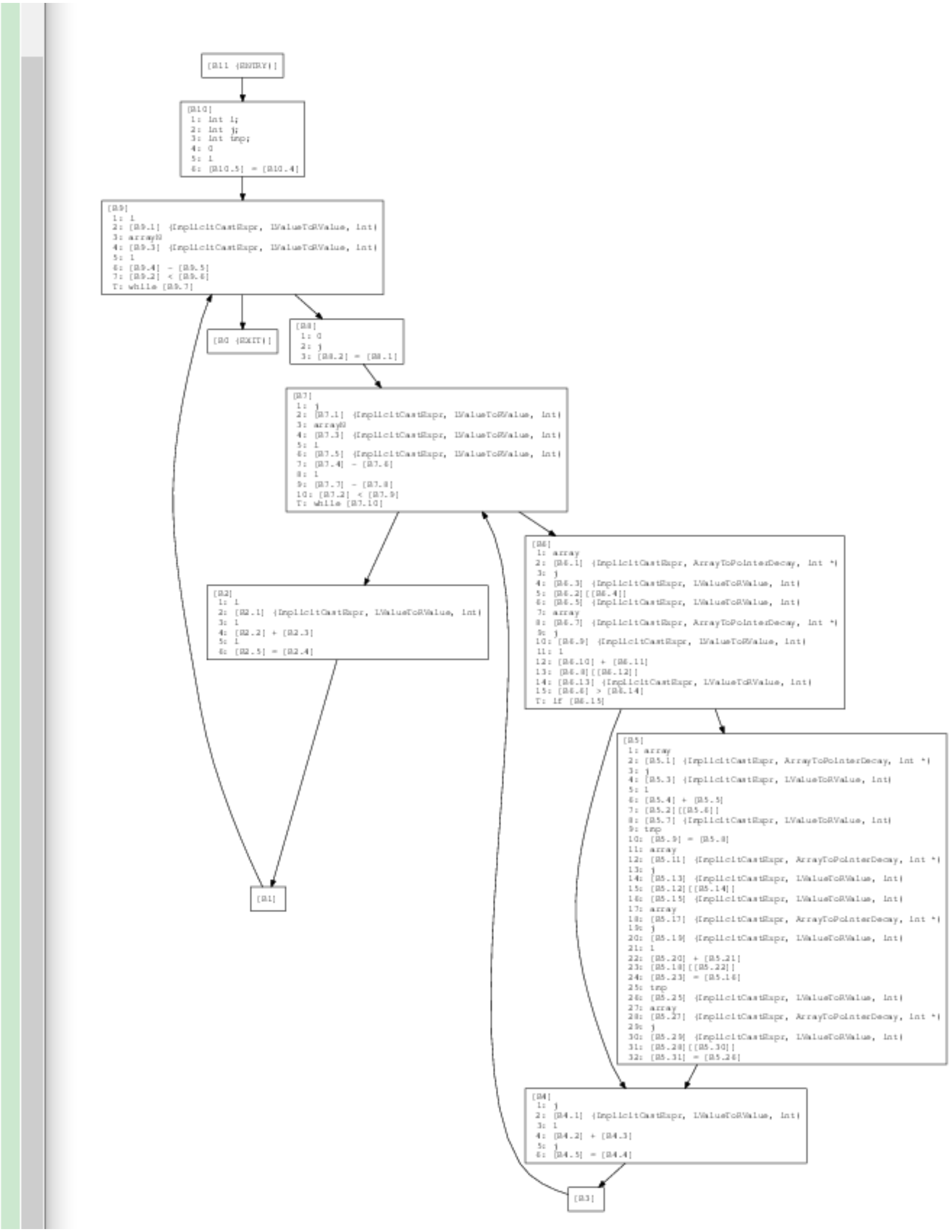
CFG

SIMILARITY

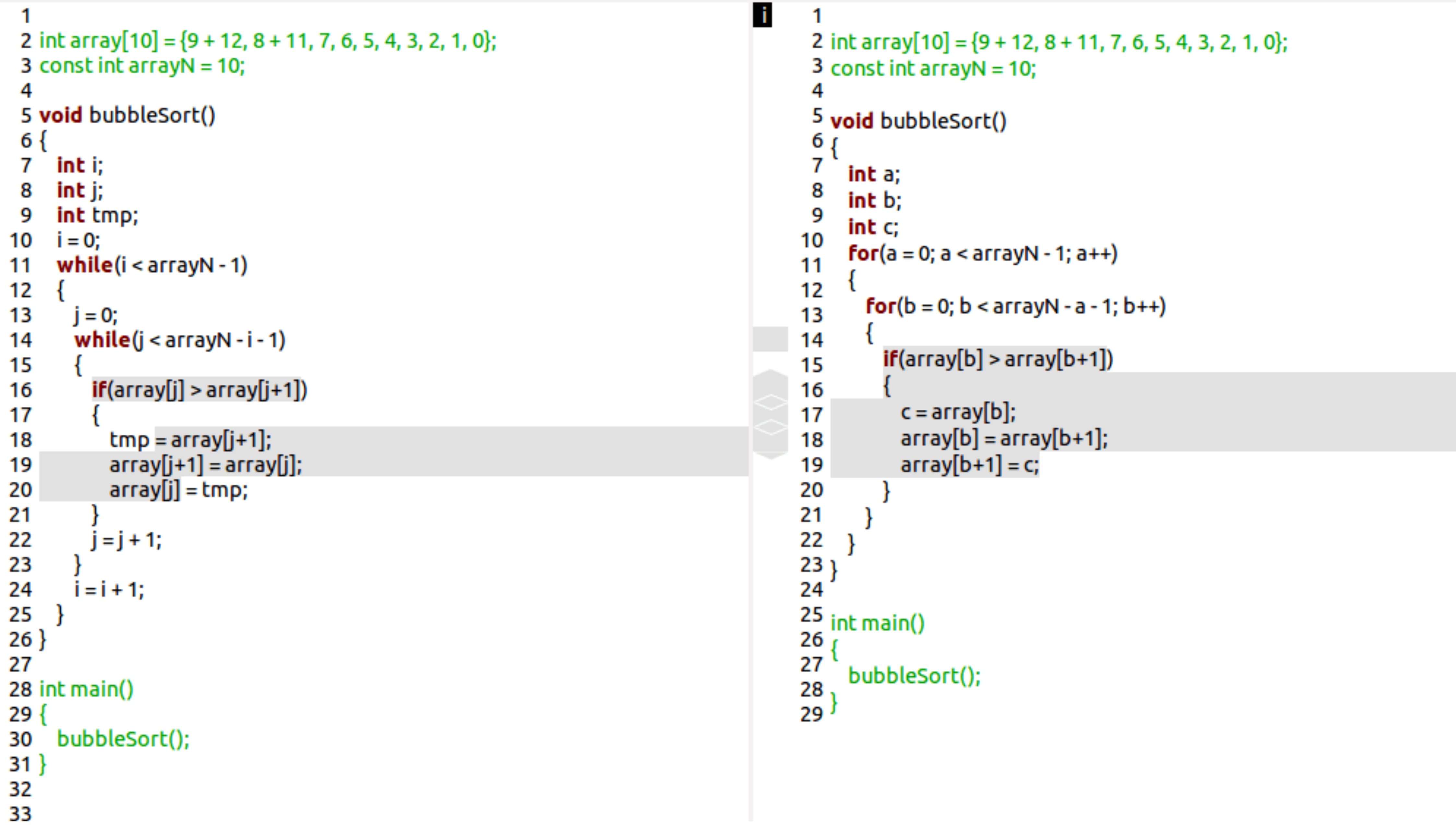
CONTROL FLOW

FEATURE EXTRACTION


[B9 (ENTRY)]
[B8]
1: int a;
2: int b;
3: int c;
4: 0
5: a
6: [B8.5] = [B8.4]
[B7]
1: a
2: [B7.1] (ImplicitCastExpr, LValueToRValue, int)
3: arrayN
4: [B7.3] (ImplicitCastExpr, LValueToRValue, int)
5: 1
6: [B7.4] - [B7.5]
7: [B7.2] < [B7.6]
T: for (...; [B7.7]; ...)
[B0 (EXIT)]
[B6]
1: 0
2: b
3: [B6.2] = [B6.1]
[B5]
1: b
2: [B5.1] (ImplicitCastExpr, LValueToRValue, int)
3: arrayN
4: [B5.3] (ImplicitCastExpr, LValueToRValue, int)
5: a
6: [B5.5] (ImplicitCastExpr, LValueToRValue, int)
7: [B5.4] - [B5.6]
8: 1
9: [B5.7] - [B5.8]
10: [B5.2] < [B5.9]
T: for (...; [B5.10]; ...)
[B1]
1: a
2: [B1.1]++
[B4]
1: array
2: [B4.1] (ImplicitCastExpr, ArrayToPointerDecay, int *)
3: b
4: [B4.3] (ImplicitCastExpr, LValueToRValue, int)
5: [B4.2][[B4.4]]
6: [B4.5] (ImplicitCastExpr, LValueToRValue, int)
7: array
8: [B4.7] (ImplicitCastExpr, ArrayToPointerDecay, int *)
9: b
10: [B4.9] (ImplicitCastExpr, LValueToRValue, int)
11: 1
12: [B4.10] + [B4.11]
13: [B4.8][[B4.12]]
14: [B4.13] (ImplicitCastExpr, LValueToRValue, int)
15: [B4.6] > [B4.14]
T: if [B4.15]
[B3]
1: array
2: [B3.1] (ImplicitCastExpr, ArrayToPointerDecay, int *)
3: b
4: [B3.3] (ImplicitCastExpr, LValueToRValue, int)
5: [B3.2][[B3.4]]
6: [B3.5] (ImplicitCastExpr, LValueToRValue, int)
7: c
8: [B3.7] = [B3.6]
9: array
10: [B3.9] (ImplicitCastExpr, ArrayToPointerDecay, int *)
11: b
12: [B3.11] (ImplicitCastExpr, LValueToRValue, int)
13: 1
14: [B3.12] + [B3.13]
15: [B3.10][[B3.14]]
16: [B3.15] (ImplicitCastExpr, LValueToRValue, int)
17: array
18: [B3.17] (ImplicitCastExpr, ArrayToPointerDecay, int *)
19: b
20: [B3.19] (ImplicitCastExpr, LValueToRValue, int)
21: [B3.18][[B3.20]]
22: [B3.21] = [B3.16]
23: c
24: [B3.23] (ImplicitCastExpr, LValueToRValue, int)
25: array
26: [B3.25] (ImplicitCastExpr, ArrayToPointerDecay, int *)
27: b
28: [B3.27] (ImplicitCastExpr, LValueToRValue, int)
29: 1
30: [B3.28] + [B3.29]
31: [B3.26][[B3.30]]
32: [B3.31] = [B3.24]
[B2]
1: b
2: [B2.1]++


[B11 (ENTRY)]
[B10]
1: int i;
2: int j;
3: int tmp;
4: 0
5: i
6: [B10.5] = [B10.4]
[B9]
1: i
2: [B9.1] (ImplicitCastExpr, LValueToRValue, int)
3: arrayN
4: [B9.3] (ImplicitCastExpr, LValueToRValue, int)
5: 1
6: [B9.4] - [B9.5]
7: [B9.2] < [B9.6]
T: while [B9.7]
[B0 (EXIT)]
[B8]
1: 0
2: j
3: [B8.2] = [B8.1]
[B7]
1: j
2: [B7.1] (ImplicitCastExpr, LValueToRValue, int)
3: arrayN
4: [B7.3] (ImplicitCastExpr, LValueToRValue, int)
5: i
6: [B7.5] (ImplicitCastExpr, LValueToRValue, int)
7: [B7.4] - [B7.6]
8: 1
9: [B7.7] - [B7.8]
10: [B7.2] < [B7.9]
T: while [B7.10]
[B2]
1: i
2: [B2.1] (ImplicitCastExpr, LValueToRValue, int)
3: 1
4: [B2.2] + [B2.3]
5: i
6: [B2.5] = [B2.4]
[B6]
1: array
2: [B6.1] (ImplicitCastExpr, ArrayToPointerDecay, int *)
3: j
4: [B6.3] (ImplicitCastExpr, LValueToRValue, int)
5: [B6.2][[B6.4]]
6: [B6.5] (ImplicitCastExpr, LValueToRValue, int)
7: array
8: [B6.7] (ImplicitCastExpr, ArrayToPointerDecay, int *)
9: j
10: [B6.9] (ImplicitCastExpr, LValueToRValue, int)
11: 1
12: [B6.10] + [B6.11]
13: [B6.8][[B6.12]]
14: [B6.13] (ImplicitCastExpr, LValueToRValue, int)
15: [B6.6] > [B6.14]
T: if [B6.15]
[B5]
1: array
2: [B5.1] (ImplicitCastExpr, ArrayToPointerDecay, int *)
3: j
4: [B5.3] (ImplicitCastExpr, LValueToRValue, int)
5: 1
6: [B5.4] + [B5.5]
7: [B5.2][[B5.6]]
8: [B5.7] (ImplicitCastExpr, LValueToRValue, int)
9: tmp
10: [B5.9] = [B5.8]
11: array
12: [B5.11] (ImplicitCastExpr, ArrayToPointerDecay, int *)
13: j
14: [B5.13] (ImplicitCastExpr, LValueToRValue, int)
15: [B5.12][[B5.14]]
16: [B5.15] (ImplicitCastExpr, LValueToRValue, int)
17: array
18: [B5.17] (ImplicitCastExpr, ArrayToPointerDecay, int *)
19: j
20: [B5.19] (ImplicitCastExpr, LValueToRValue, int)
21: 1
22: [B5.20] + [B5.21]
23: [B5.18][[B5.22]]
24: [B5.23] = [B5.16]
25: tmp
26: [B5.25] (ImplicitCastExpr, LValueToRValue, int)
27: array
28: [B5.27] (ImplicitCastExpr, ArrayToPointerDecay, int *)
29: j
30: [B5.29] (ImplicitCastExpr, LValueToRValue, int)
31: [B5.28][[B5.30]]
32: [B5.31] = [B5.26]
[B4]
1: j
2: [B4.1] (ImplicitCastExpr, LValueToRValue, int)
3: 1
4: [B4.2] + [B4.3]
5: j
6: [B4.5] = [B4.4]
[B3]
[B1]

Demo



```

int array[10] = {9 + 12, 8 + 11, 7, 6, 5, 4, 3, 2, 1, 0};
const int arrayN = 10;

void bubbleSort()
{
  int i;
  int j;
  int tmp;
  i = 0;
  while(i < arrayN - 1)
  {
    j = 0;
    while(j < arrayN - i - 1)
    {
      if(array[j] > array[j+1])
      {
        tmp = array[j+1];
        array[j+1] = array[j];
        array[j] = tmp;
      }
      j = j + 1;
    }
    i = i + 1;
  }
}

int main()
{
  bubbleSort();
}
```

```

int array[10] = {9 + 12, 8 + 11, 7, 6, 5, 4, 3, 2, 1, 0};
const int arrayN = 10;

void bubbleSort()
{
  int a;
  int b;
  int c;
  for(a = 0; a < arrayN - 1; a++)
  {
    for(b = 0; b < arrayN - a - 1; b++)
    {
      if(array[b] > array[b+1])
      {
        c = array[b];
        array[b] = array[b+1];
        array[b+1] = c;
      }
    }
  }
}

int main()
{
  bubbleSort();
}
```

RESULT

Better Accuracy than token based methods

Potentially outperform CC-Finder

RESULT

Sample : 30 X 30 third-part codes

Competitors: JPlag Moss

Outcomes : Almost better than the competitors

Failed Case: Considerable change on AST

FUTURE WORK

FILTERING BASED CONTROL FLOW

FRIENDLY INTERFACE

SIMILARITY ENHANCEMENT

Long way to go…

QUESTIONS

THANK YOU