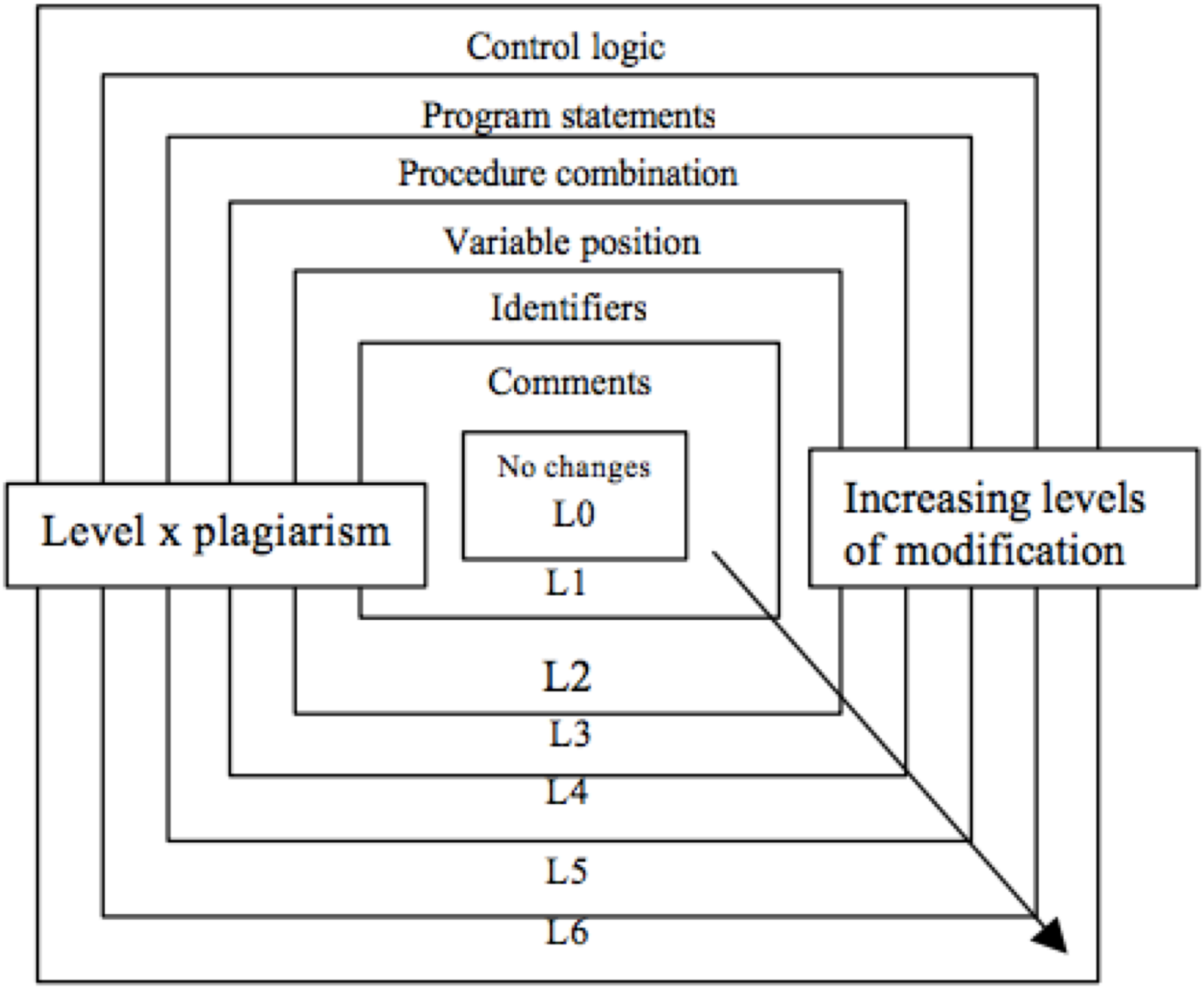
Learning Analytics

Dr. Bowen Hui
Computer Science
University of British Columbia Okanagan



Reference: Faidhi & Robinson (1987). An empirical approach for detecting program similarity and plagiarism within a university programming environment, Computing in Education, Vol. 11, pp(11-19).

Plagiarism Disguises

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion

Plagiarism Disguises

- Format alteration
  - Insert/remove blanks
  - Insert/remove comments
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion

Ex: Method to Sum Up Array Elements

```
int sum( int array[], int count )
{
   int i, sum;
   sum = 0;
   for( i=0; i<count; i++ )
      sum = sum + array[i];
   return sum;
}
```

Ex: Method to Sum Up Array Elements

```
int sum( int array[], int count )
{
   // var declarations
   int i, sum;

   // tally up each array element
   sum = 0;
   for( i=0; i<count; i++ )
   {
      sum = sum + array[i];
   }

   // return total
   return sum;
}
```

Added comments and {,}

Plagiarism Disguises

- Format alteration
- Identifier Renaming
  - Change identifier names without violating correctness
  - How to match identifiers in two programs?
  - Potentially change data type, split/merge variables
- Statement Reordering
- Control Replacement
- Code Insertion

Ex: Method to Sum Up Array Elements

```
int sum( int array[] )
{
    // var declarations
    int index, sum;
    int len = array.length;

    // tally up each array element
    sum = 0;
    for( index=0; index<len; index++ )
    {
        sum = sum + array[index];
    }

    // return total
    return sum;
}
```

Rename i and count

Refactored len as a local var

Plagiarism Disguises

- Format alteration
- Identifier Renaming
- Statement Reordering
  - Reordering statements without causing errors
  - Common: declaration statements split and moved all over the code
  - Chunks of code can often be reordered
- Control Replacement
- Code Insertion

Ex: Method to Sum Up Array Elements

```
int sum( int array[] )
{
    // var declarations
    int index = 0;
    int len = array.length;

    // tally up each array element
    int sum = 0;
    for( index=0; index<len; index++ )
    {
        sum = sum + array[index];
    }

    // return total
    return sum;
}
```

Split var declarations

Plagiarism Disguises

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
  - Exchanging for loop with while loop
  - Reversing logical conditions
    e.g. if(a) then X else Y ⇔ if(!a) then Y else X
- Code Insertion

Ex: Method to Sum Up Array Elements

```
int sum( int array[] )
{
    // var declarations
    int index = 0;
    int len = array.length;

    // tally up each array element
    int sum = 0;
    while( index<len )
    {
        sum = add( sum, array[index] );
        index++;
    }

    // return total
    return sum;
}
```

Replaced for loop

Created add method

Plagiarism Disguises

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion
  - Inject inconsequential code

Ex: Method to Sum Up Array Elements

```
int sum( int array[] )
{
    // var declarations
    int index = 0;
    int len = array.length;

    // tally up each array element
    int sum = 0;
    while( index<len )
    {
        sum = add( sum, array[index] );
        index++;
        System.out.println( "sum = " + sum );
    }

    // return total
    return sum;
}
```

Added println statement

Exercise: Insert Disguises

- Consider the following code:

```
public class Factorial
{
        public static void main(String[] args)
        {       final int NUM_FACTS = 100;
                for(int i = 0; i < NUM_FACTS; i++)
                        System.out.println( i + "! is " + factorial(i));
        }

        public static int factorial(int n)
        {       int result = 1;
                for(int i = 2; i <= n; i++)
                        result *= i;
                return result;
        }
}
```

- Add each of the following plagiarism disguises:
  - Identifier renaming
  - Statement reordering
  - Control replacement
  - Code insertion

How to Detect Disguises?

- Format alteration
  - Insert/remove blanks
  - Insert/remove comments
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion

How to Detect Disguises?

- Format alteration
  - Insert/remove blanks
  - Insert/remove comments
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion

- Tokenize code
- Strip comments

How to Detect Disguises?

- Format alteration
- Identifier Renaming
  - Change identifier names without violating correctness
  - How to match identifiers in two programs?
  - Potentially change data type, split/merge variables
- Statement Reordering
- Control Replacement
- Code Insertion

How to Detect Disguises?

- Format alteration
- Identifier Renaming
  - Change identifier names without violating correctness
    - Easy if program structure is intact
  - How to match identifiers in two programs?
  - Potentially change data type, split/merge variables
- Statement Reordering
- Control Replacement
- Code Insertion

- Create variable space
- Keep track of variables and modifications
- Compare “distance”

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
  - Reordering statements without causing errors
  - Common: declaration statements split and moved all over the code
  - Chunks of code can often be reordered
- Control Replacement
- Code Insertion

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
  - Reordering statements without causing errors
  - Common: declaration statements split and moved all over the code
  - Chunks of code can often be reordered
- Control Replacement
- Code Insertion

- Determine type of statement
- Compare changes in statement dependencies

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
  - Exchanging for loop with while loop
  - Reversing logical conditions
    e.g. if(a) then X else Y ⇔ if(!a) then Y else X
- Code Insertion

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
  - Exchanging for loop with while loop
  - Reversing logical conditions
    e.g. if(a) then X else Y ⇔ if(!a) then Y else X
- Code Insertion

- Loop label and associated syntax changes
- Boolean logic evaluation

How to Detect Disguises?

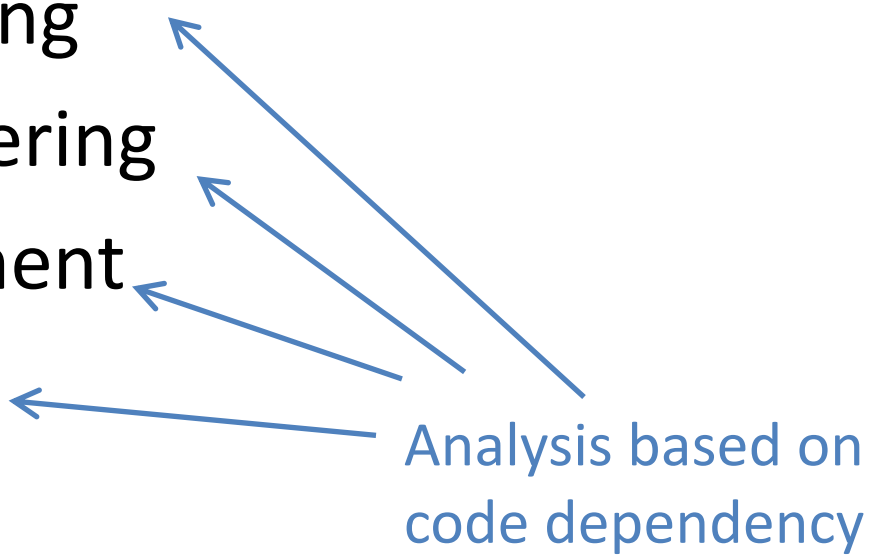
- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion
  - Inject inconsequential code

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion
  - Inject inconsequential code
    - Statements with no dependency on code logic

How to Detect Disguises?

- Format alteration
- Identifier Renaming
- Statement Reordering
- Control Replacement
- Code Insertion



Program Dependency Graph (PDG)

- A program dependency graph for a procedure P is a 4-tuple G=(V,E,μ, δ) where:
  - V is the set of vertices in P
  - μ: V $\rightarrow$ S is a function assigning types to vertices
  - E $\subseteq$ V $\times$ V is the set of dependency edges
  - δ: E $\rightarrow$ T is a function assigning types to edges
  - |G| = |V|

PDG Vertices

- Represent statements
- Each vertex has one and only one type

Type	Description
call-site	Call to procedures.
control	If, switch, while, do-while, or for.
declaration	Declaration for a variable or formal parameter.
assignment	Assignment expression.
increment	++ or −− expression
return	Function return expression.
expression	General expression except the above three, like one with ? operator
jump	Goto, break, or continue
label	Program labels
switch-case	Case or Default

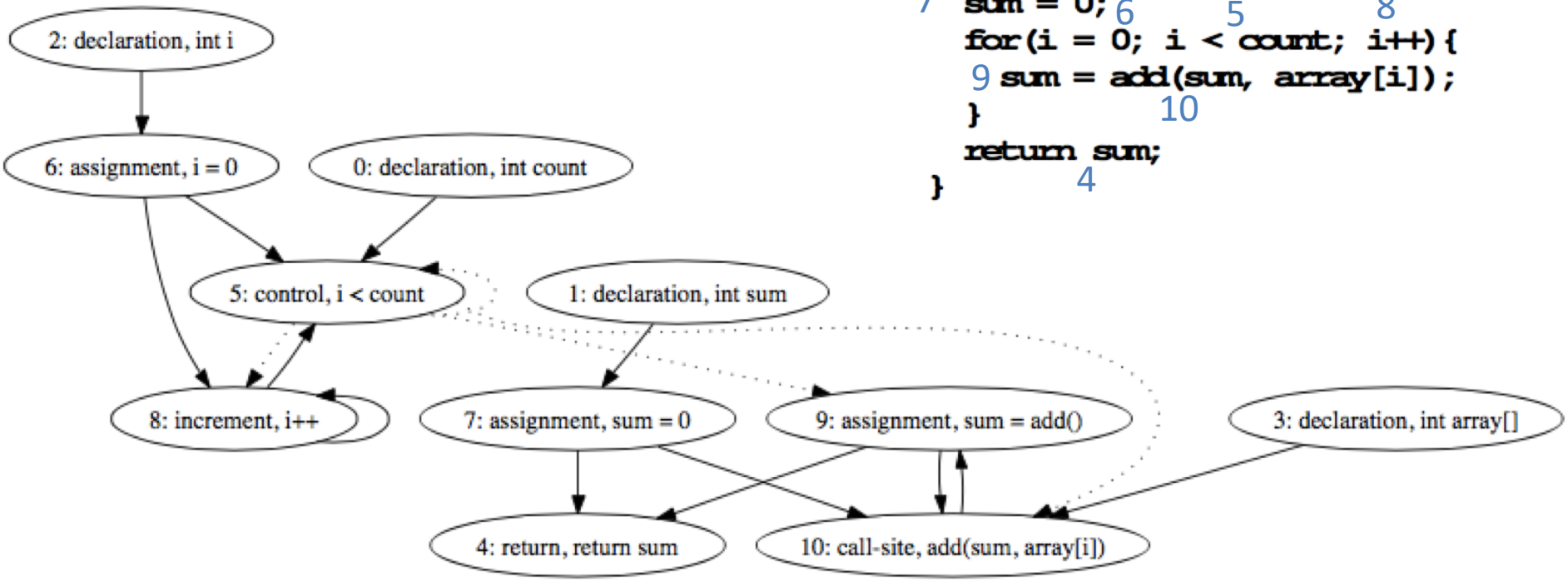
See GPLAG paper for details

PDG Edges

- Model dependencies between vertices
- Control dependencies: connects a control vertex to another vertex whose statement will be executed if the condition is evaluated to true

Example of Control Dependencies

```
int sum(int array[], int count)
{
  int i, sum;
  sum = 0;
  for(i = 0; i < count; i++){
  sum = add(sum, array[i]);
  }
  return sum;
}
```



Example taken from GPLAG paper

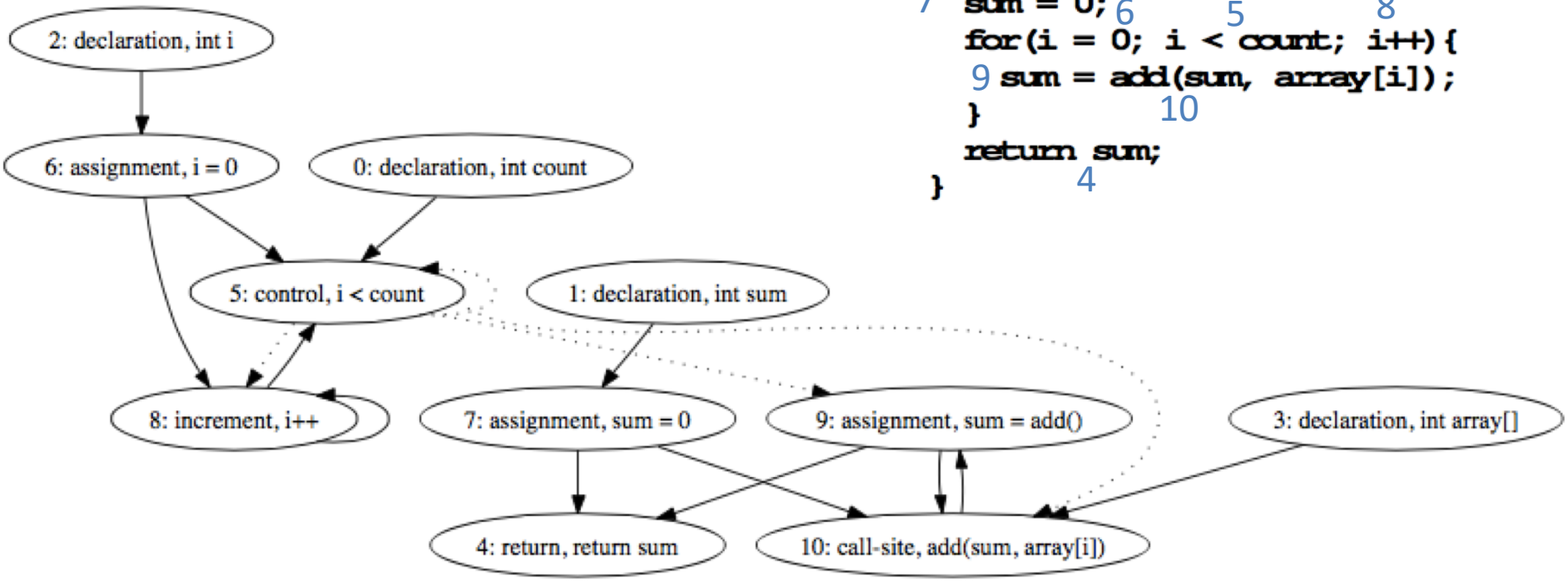
Control dependencies - - - - - -

PDG Edges

- Model dependencies between vertices
- Control dependencies: connects a control vertex to another vertex whose statement will be executed if the condition is evaluated to true
- Data dependencies: connects vertices v_1 and v_2 if there is some variable `var` such that:
  - v_1 may be assigned to var
  - v_2 may use value in var
  - There is an execution path from v_1 to v_2 in the code where there is no assignment to `var`

Example of Data Dependencies

```
int sum(int array[], int count)
{
  int i, sum;
  sum = 0;
  for(i = 0; i < count; i++){
  sum = add(sum, array[i]);
  }
  return sum;
}
```



Example taken from GPLAG paper

Control dependencies

Data dependencies

PDG History

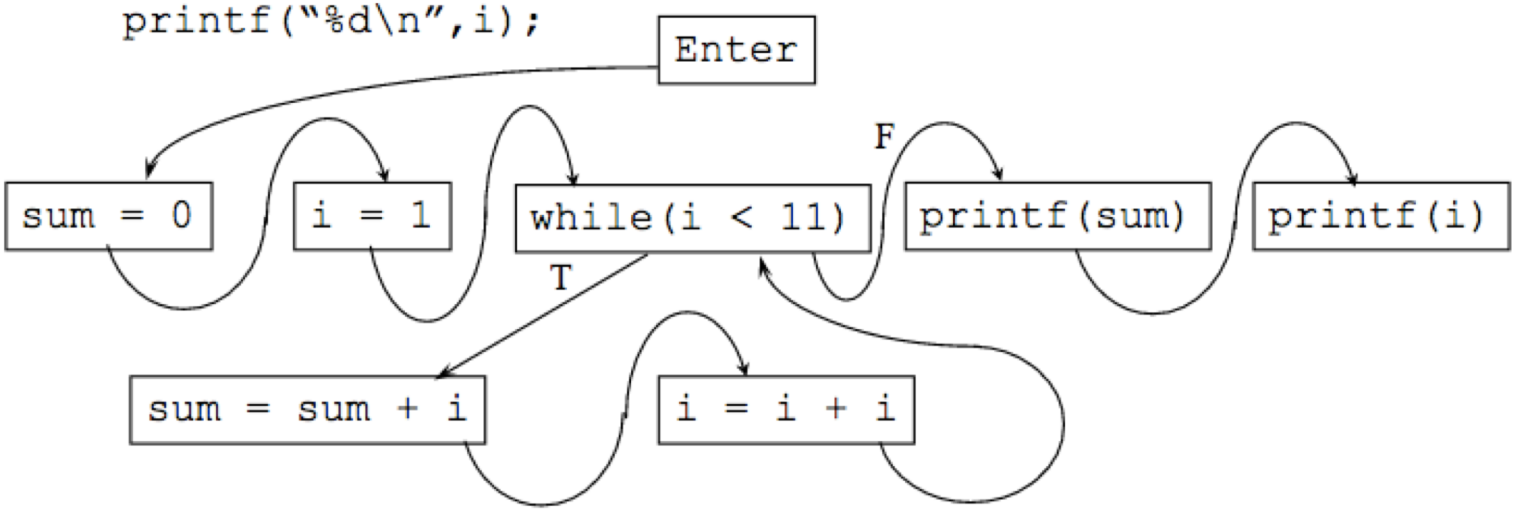
- Original use in code optimization
  - Compilers
  - Parallel processing
  - Software maintenance, optimization, refactoring

- 2006: GPLAG algorithm
  - Efficiently find two pieces of code to check
  - Create PDG for each piece of code
  - Compare “distance” between two PDGs

Example

```
int main() {
        int sum = 0;
        int i = 1;
        while (i < 11) {
                sum = sum + i;
                i = i + 1;
        }
        printf("%d\n",sum);
        printf("%d\n",i);
}
```

Control flow graph
Not a PDG

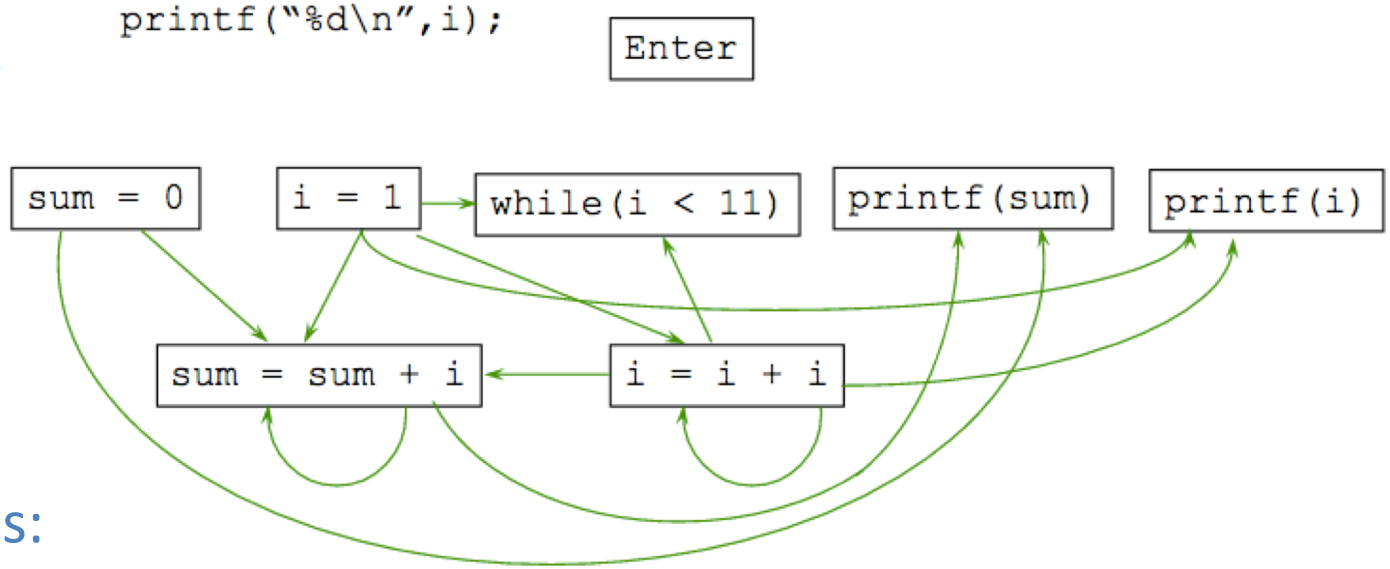


Example (cont.)

```
int main() {
        int sum = 0;
        int i = 1;
        while (i < 11) {
                sum = sum + i;
                i = i + 1;
        }
        printf("%d\n",sum);
        printf("%d\n",i);
}
```

Data dependence

p → q Value of variable assigned at p may be used at q.



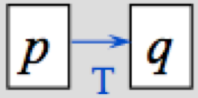
Variations:

- No vertices for declaration
- Extra “enter” vertex

Example (cont.)

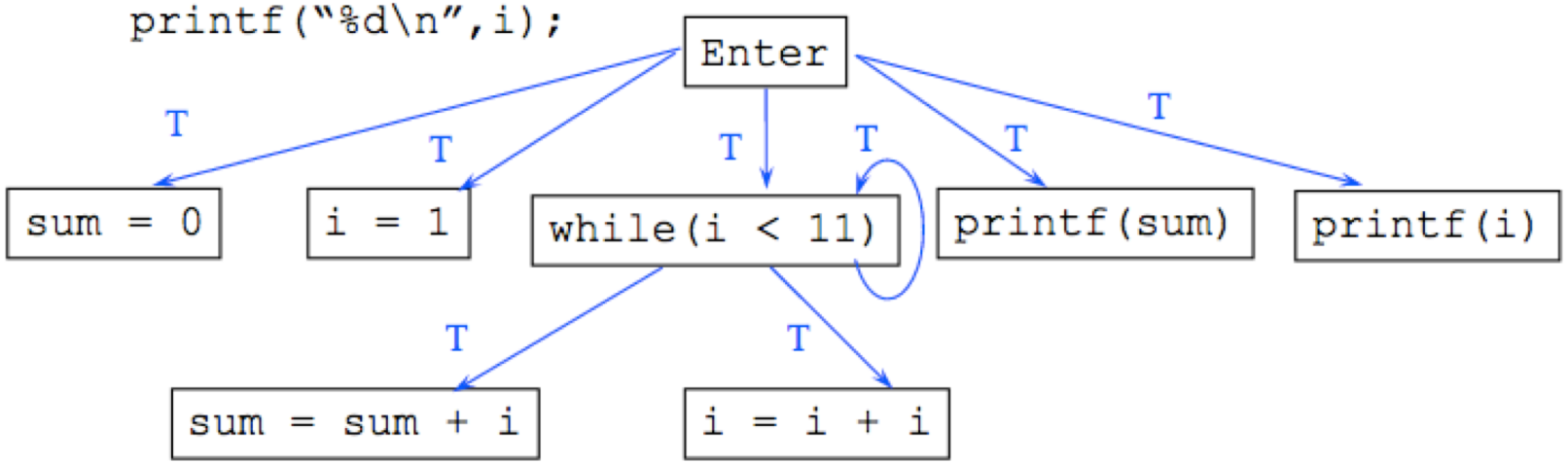
```
int main() {
        int sum = 0;
        int i = 1;
        while (i < 11) {
                sum = sum + i;
                i = i + 1;
        }
        printf("%d\n",sum);
        printf("%d\n",i);
}
```

Control dependence



q is reached from *p* if condition *p* is true (T), not otherwise.

Similar for false (F).

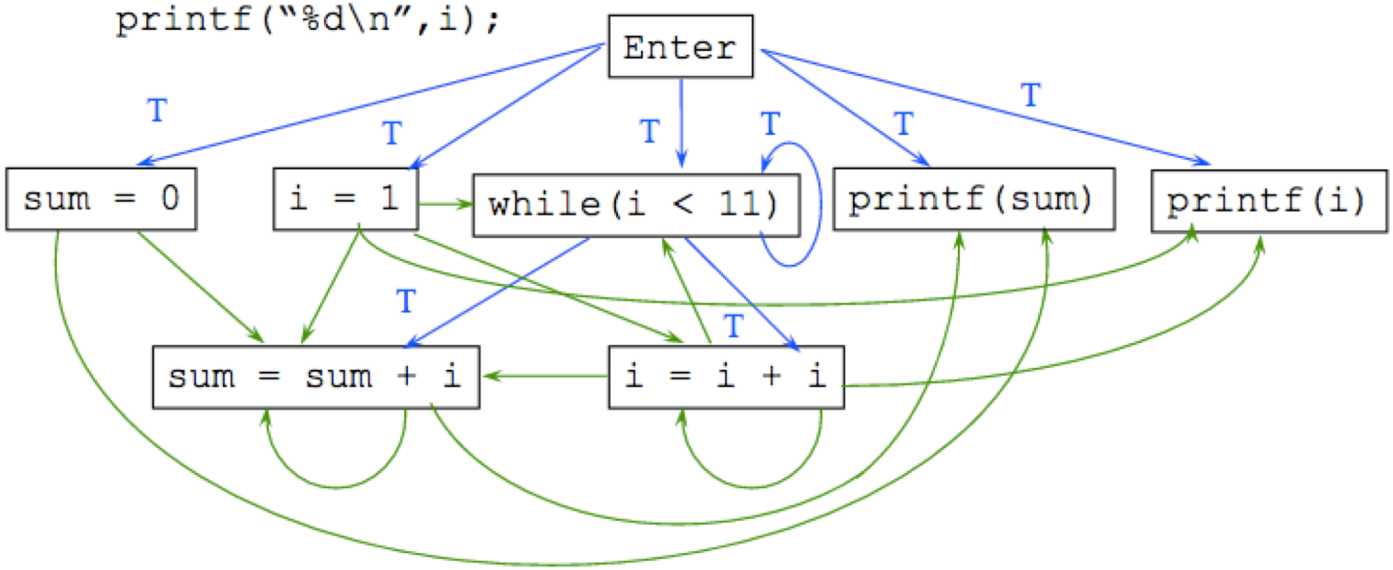


Example (cont.)

```
int main() {
        int sum = 0;
        int i = 1;
        while (i < 11) {
                sum = sum + i;
                i = i + 1;
        }
        printf("%d\n",sum);
        printf("%d\n",i);
}
```

Control dependence

Data **dependence**

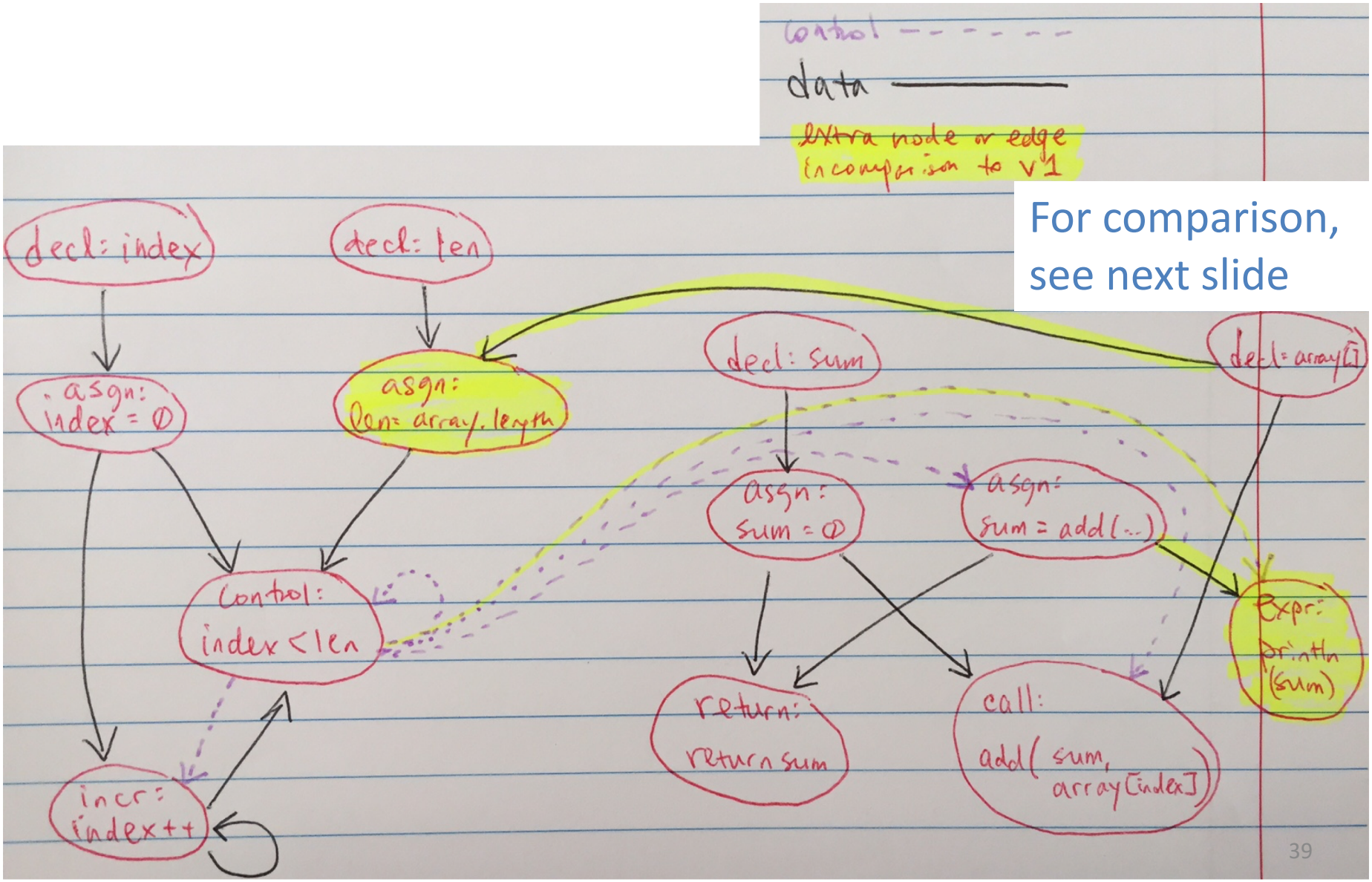


Exercise

Given the following code, build its PDG (follow table of vertex types)

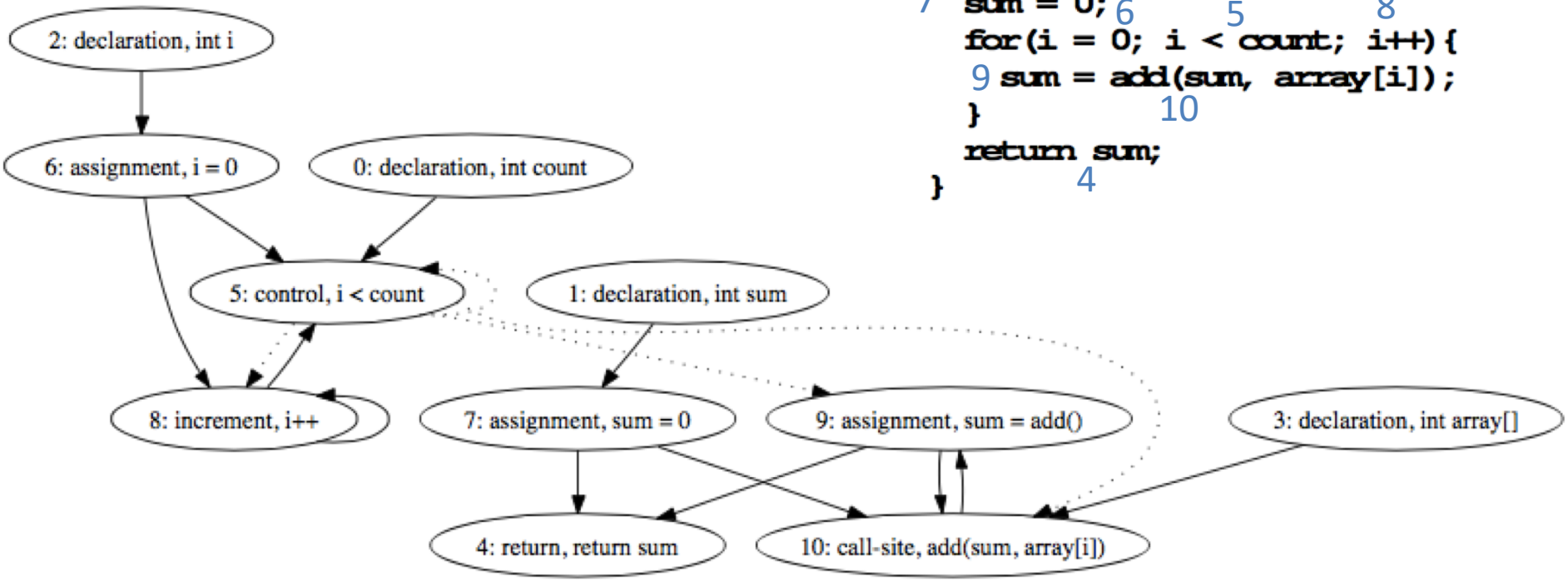
```
int sum( int array[] )
{
    // var declarations
    int index = 0;
    int len = array.length;

    // tally up each array element
    int sum = 0;
    while( index<len )
    {
        sum = add( sum, array[index] );
        index++;
        System.out.println( “sum = “ + sum );
    }

    // return total
    return sum;
}
```

Exercise Solution



PDG of Original Code

```
int sum(int array[], int count)
{
  int i, sum;
  sum = 0;
  for(i = 0; i < count; i++){
  sum = add(sum, array[i]);
  }
  return sum;
}
```



Example taken from GPLAG paper

Control dependencies

Data dependencies

Problem Formulation

	Original	**Suspect**
Program Source	*P*	*P'*
Number of Procedures	*n*	*m*
Converted PDG	*G*	*G'*
Size	$\|G\| = n$	$\|G\| = m$

- Subtasks:
  - Given $g \in G$ and $g' \in G'$, decide if g' is plagiarized from g
  - How to efficiently locate code pairs without $n \times m$ comparisons?

Main Claims

- Restricted to 5 disguises (see above)

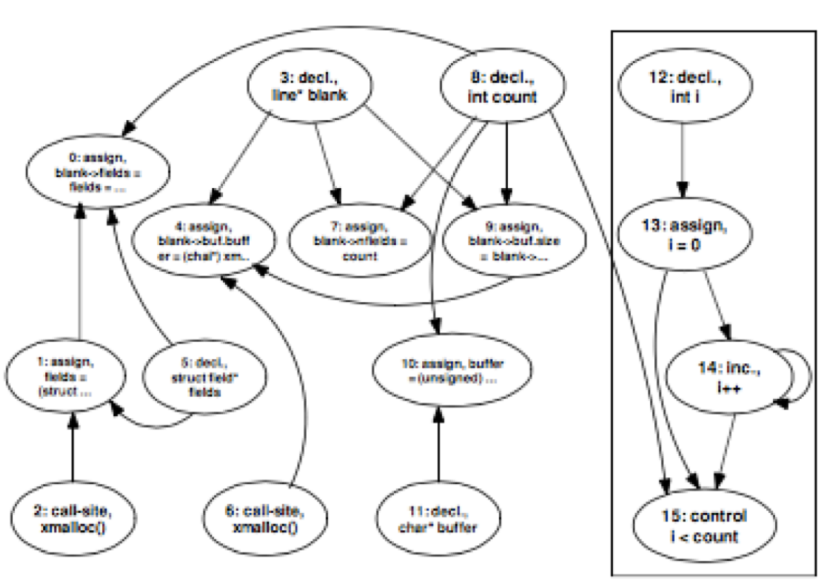
1. If *g* is subgraph isomorphic to *g'*, then the corresponding procedure of *g'* is considered as plagiarized from *g*

Recall Disguises

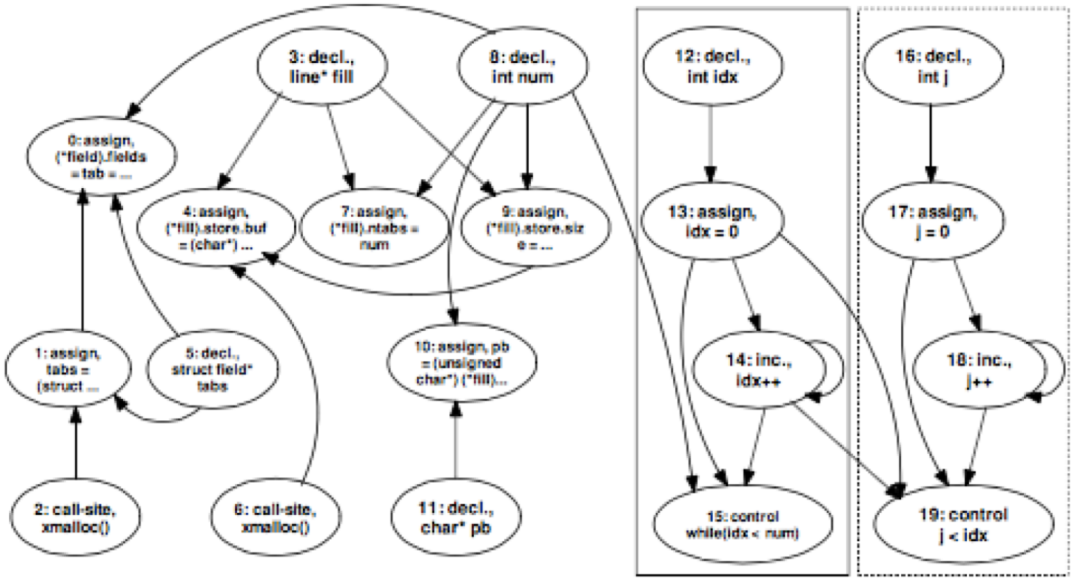
- PDGs generally immune to the following:
  - Format alteration
  - Identifier Renaming
  - Statement Reordering
  - Control Replacement

- Assuming correctness is preserved, PDG of plagiarized code is “bigger”
  - Code Insertion

An inserted extra loop (within loop) that essentially does nothing



(a) PDG of the Original Code



(b) PDG of the Plagiarized Code

Example taken from GPLAG paper

loop

loop

Left graph (g) is subgraph isomorphic to right graph (g')

Beyond 5 Disguises

- Detect cheats resulting in “similar enough” PDGs
  - Example of having two variables merged into one: Simple code change that modifies vertices in PDG
- Set threshold γ which indicates proportion of overlap
  - Suggested use of 0.9
  - More than 10% differences in PDGs is like rewriting code

Main Claims

- Restricted to 5 disguises (see above)

1. If *g* is subgraph isomorphic to *g'*,
   then the corresponding procedure of *g'* is considered as plagiarized from *g*

- Beyond 5 disguises

2. If *g* is γ-isomorphic to *g'*,
   then the corresponding procedure of *g'* is considered as plagiarized from *g*
   Note: $0 < \gamma \leq 1$

Graph Terminology

- Given two graphs, check isomorphism
- Define graph isomorphism
  A bijective function $f: V \to V'$ is a graph morphism from a graph $G = (V, E, \mu, \delta)$to a graph $G' = (V', E', \mu', \delta')$ if:
  - $\mu(v) = \mu'(f(v))$
  - $\forall e = (v_1, v_2) \in E,$
    $\exists e' = \big(f(v_1), f(v_2)\big) \in E'$ such that $\delta(e) = \delta(e')$
  - $\forall e' = (v'_1, v'_2) \in E',$
    $\exists e = \big(f^{-1}(v'_1), f^{-1}(v'_2)\big) \in E$ such that $\delta(e') = \delta(e)$

One-to-one correspondence

Graph Terminology

- Given two graphs, check isomorphism
- Define graph isomorphism
- Define subgraph isomorphism
  An injective function $f: V \rightarrow V'$ is a subgraph $S \subset G'$ such that f is a graph isomorphism from G to S

One-to-one mapping that
preserves distinctness of elements in domain

Graph Terminology

- Given two graphs, check isomorphism
- Define graph isomorphism
- Define subgraph isomorphism
- Define γ-Isomorphic
  A graph *G* is γ-isomorphic to *G'* if there exists a subgraph $S \subset G$ such that *S* is subgraph isomorphic to *G'* and $|S| \geq \gamma|G|$ where $\gamma \in (0,1]$

Similar to computing "distance" between two graphs

Overall GPLAG Algorithm

- Inputs:
  P, *P'* (and some parameters)
- Output:
  F, the set of PDG pairs considered to be involved in plagiarism (for human consideration)

Overall GPLAG Algorithm

- Inputs:
  P, *P'* (and some parameters)
- Output:
  F, the set of PDG pairs considered to be involved in plagiarism (for human consideration)
- Steps:
  - Construct *G* and *G'*
  - Efficiently identify *g* and *g'* pairs to compare
  - If *g'* is γ-isomorphic to *g*
    - Add to suspect set for output: $F = F \cup (g, g')$
  - Return *F*

We skipped this

Key Ideas

- Detecting source code plagiarism is much harder than detecting plagiarism in natural language
  - Lack idiosyncrasies
  - Trivial changes can modify code logic and flow
- Representation:
  - Models source code as program dependency graph (ignores superficial code variants)
- Algorithm:
  - GPLAG: Uses graph isomorphism to detect plagiarism