A Tree Kernel Based Approach for Clone Detection

Anna Corazza[1], Sergio Di Martino[1],
Valerio Maggio[1], Giuseppe Scanniello[2]

1) University of Naples Federico II
2) University of Basilicata

Outline

- Background
  - Clone detection definition
  - State of the Art Techniques Taxonomy
- Our Abstract Syntax Tree based Proposal
  - A *Tree Kernel* based approach for clone detection
- A preliminary evaluation

Code Clones

- Two code fragments form a **clone** if they are similar enough according to a given measure of similarity (*I.D. Baxter, 1998*)

3. R. Tiarks, R. Koschke, and R. Falke,
An assessment of type-3 clones as detected by state-of-the-art tools

Code Clones

- Two code fragments form a **clone** if they are similar enough according to a given measure of similarity (*I.D. Baxter, 1998*)
- Similarity based on **Program Text** or on "**Semantics**"

3. R. Tiarks, R. Koschke, and R. Falke,
An assessment of type-3 clones as detected by state-of-the-art tools

Code Clones

- Two code fragments form a **clone** if they are similar enough according to a given measure of similarity (*I.D. Baxter, 1998*)
- Similarity based on **Program Text** or on "**Semantics**"
- **Program Text** can be further distinguished by their degree of similarity[1]
  - Type 1 Clone: **Exact Copy**
  - Type 2 Clone: **Parameter Substituted Clone**
  - Type 3 Clone: **Modified/Structure Substituted Clone**

1. R. Tiarks, R. Koschke, and R. Falke,
An assessment of type-3 clones as detected by state-of-the-art tools

State of the Art Techniques

- Classified in terms of Program Text representation[2]
  - String, token, syntax tree, control structures, metric vectors
- String/Token based Techniques
- Abstract Syntax Tree (AST) Techniques
- ...

2. Roy, Cordy, Koschke Comparison and Evaluation of Clone Detection Tools and Technique 2009

State of the Art Techniques

- String/Token based Techniques
- Abstract Syntax Tree (AST) Techniques
- ...
- Combined Techniques (a.k.a. Hybrid)
  - Combine different representations
  - Combine different techniques
  - Combine different sources of information
    - Tree Kernel based approach (Our approach :)

The Proposed Approach

The Goal

- Define an AST based technique able to detect up to Type 3 Clones

The Goal

- Define an AST based technique able to detect up to Type 3 Clones
- The Key Ideas:
  - Improve the amount of information carried by ASTs by adding (also) lexical information
  - Define a proper measure to compute similarities among (sub)trees, exploiting such information

The Goal

- Define an AST based technique able to detect up to Type 3 Clones
- The Key Ideas:
  - Improve the amount of information carried by ASTs by adding (also) lexical information
  - Define a proper measure to compute similarities among (sub)trees, exploiting such information
- As a measure we propose the use of a **(Tree) Kernel Function**

Kernels for Structured Data

- Kernels are a class of functions with many appealing features:
  - Are based on the idea that a complex object can be described in terms of its constituent parts
  - Can be easily tailored to a specific domain
- There exist different classes of Kernels:
  - String Kernels
  - Graph Kernels
  - …
  - **Tree Kernels**
    - Applied to NLP Parse Trees (Collins and Duffy 2004)

Defining a new Tree Kernel

- The definition of a new Tree Kernel requires the specification of:

  (1) A set of **features** to annotate nodes of compared trees

Defining a new Tree Kernel

- The definition of a new Tree Kernel requires the specification of:
  - (1) A set of **features** to annotate nodes of compared trees
  - (2) A (**primitive**) **Kernel Function** to measure the similarity of each pair of nodes

Defining a new Tree Kernel

- The definition of a new Tree Kernel requires the specification of:
  - (1) A set of **features** to annotate nodes of compared trees
  - (2) A (**primitive**) **Kernel Function** to measure the similarity of each pair of nodes
  - (3) A proper **Kernel Function** to compare subparts of trees

(1) The defined features

- We annotate each node of AST by **4 features**:

(1) The defined features

- We annotate each node of AST by **4 features**:
  - `Instruction Class`
    - i.e. `LOOP, CONDITIONAL CONTROL, CONTROL FLOW CONTROL,...`

(1) The defined features

- We annotate each node of AST by **4 features**:
  - `Instruction Class`
    - i.e. `LOOP, CONDITIONAL CONTROL, CONTROL FLOW CONTROL,...`
  - `Instruction`
    - i.e. `FOR, WHILE, IF, RETURN, CONTINUE,...`

(1) The defined features

- We annotate each node of AST by **4 features**:
  - Instruction Class
    - i.e. LOOP, CONDITIONAL CONTROL, CONTROL FLOW CONTROL,...
  - Instruction
    - i.e. FOR, WHILE, IF, RETURN, CONTINUE,...
  - Context
    - Instruction class of statement in which node is enclosed

(1) The defined features

- We annotate each node of AST by **4 features**:
  - Instruction Class
    - i.e. LOOP, CONDITIONAL CONTROL, CONTROL FLOW CONTROL,...
  - Instruction
    - i.e. FOR, WHILE, IF, RETURN, CONTINUE,...
  - Context
    - Instruction class of statement in which node is enclosed
  - Lexemes
    - Lexical information within the code

Context Feature

- Rationale: two nodes are more similar if they appear in the same `Instruction` class

```
for (int i=0; i<10; i++)
    x += i+2;
```

```
while (i<10)
    x += i+2;
```

```
if (i<10)
    x += i+2;
```

Context Feature

- Rationale: two nodes are more similar if they appear in the same `Instruction` class

```
for (int i=0; i<10; i++)
    x += i+2;
```

```
while (i<10)
    x += i+2;
```

```
if (i<10)
    x += i+2;
```

Context Feature

- Rationale: two nodes are more similar if they appear in the same `Instruction` class

```
for (int i=0; i<10; i++)
    x += i+2;
```

```
while (i<10)
    x += i+2;
```

```
if (i<10)
    x += i+2;
```

Context Feature

- Rationale: two nodes are more similar if they appear in the same `Instruction` class

```
for (int i=0; i<10; i++)
    x += i+2;
```

```
while (i<10)
    x += i+2;
```

```
if (i<10)
    x += i+2;
```

Context Feature

- Rationale: two nodes are more similar if they appear in the same `Instruction` class

```
for (int i=0; i<10; i++)
    x += i+2;
```

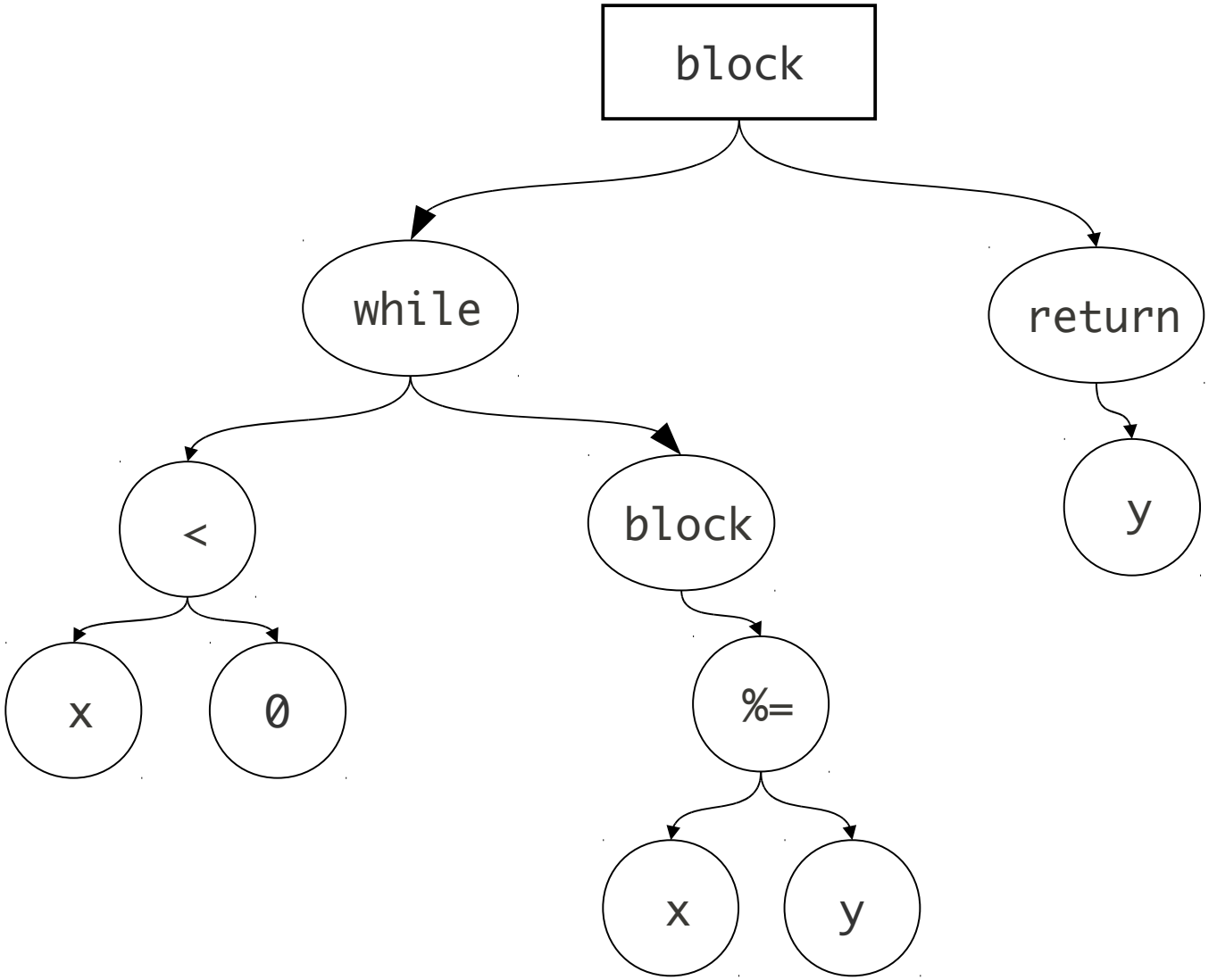
```
while (i<10)
    x += i+2;
```

```
if (i<10)
    x += i+2;
```

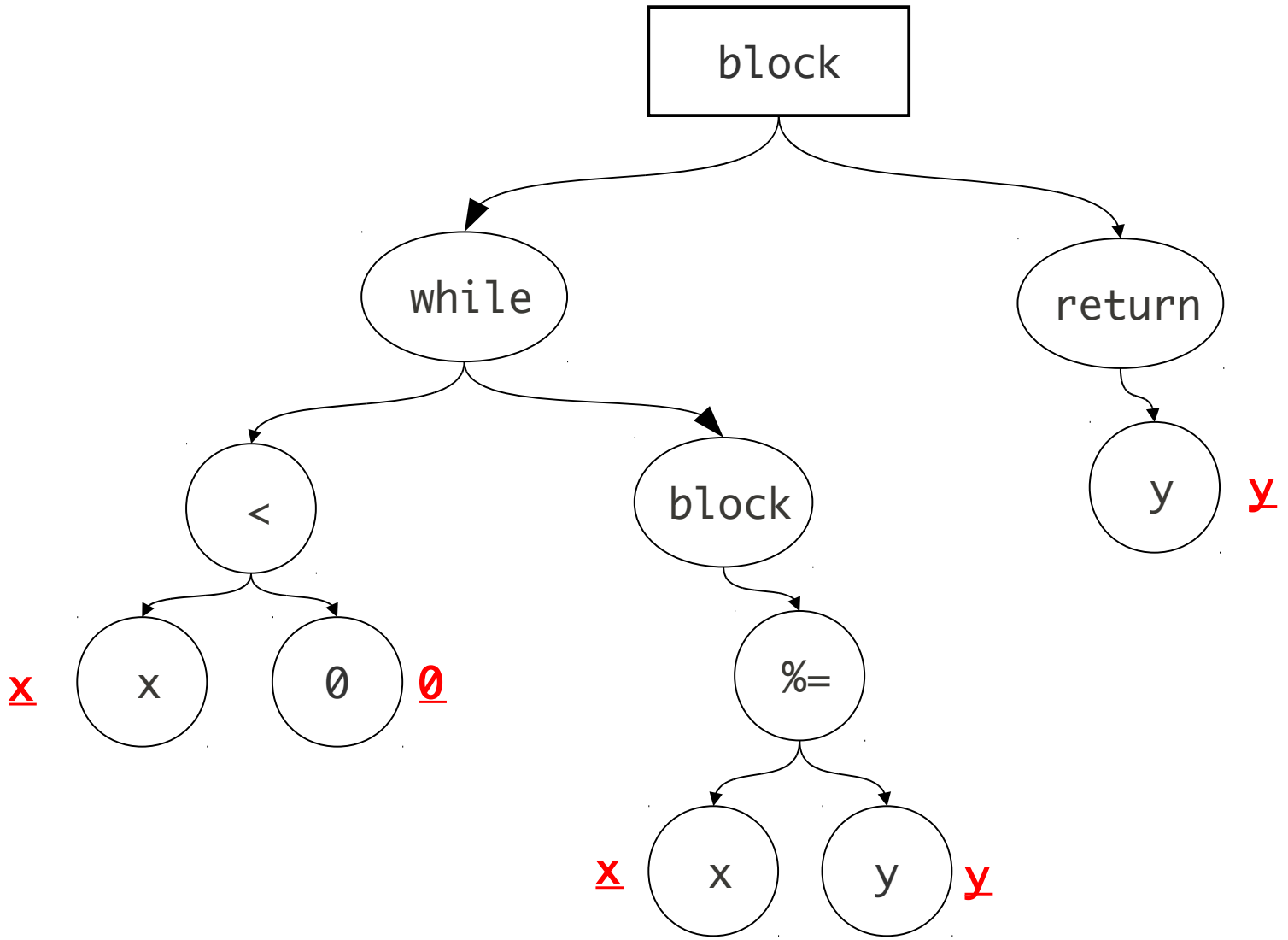
Lexemes Feature

- For leaf nodes:
  - It is the lexeme associated to the node
- For internal nodes:
  - It is the set of lexemes that recursively comes from subtrees with minimum height

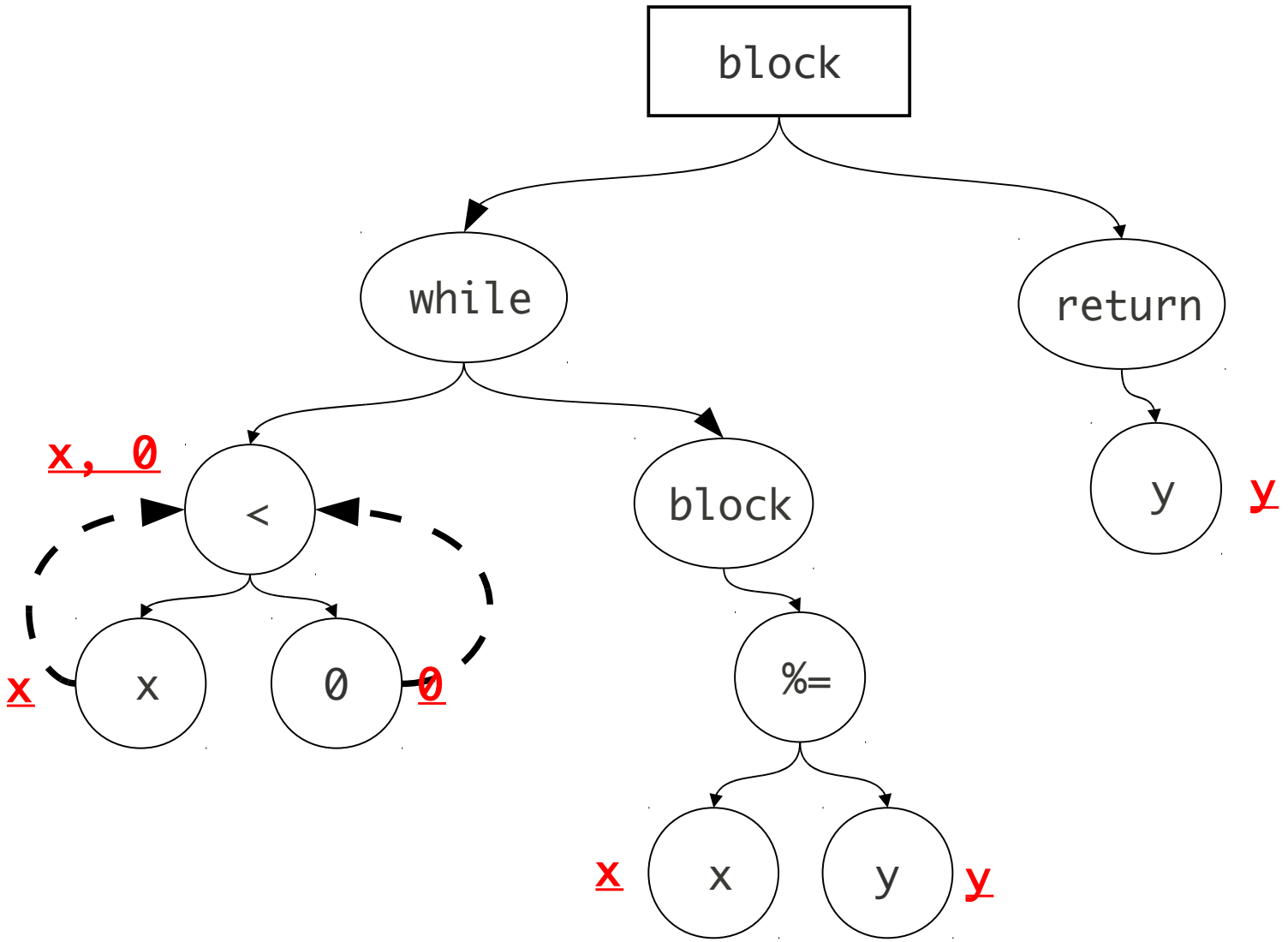
Lexemes Propagation



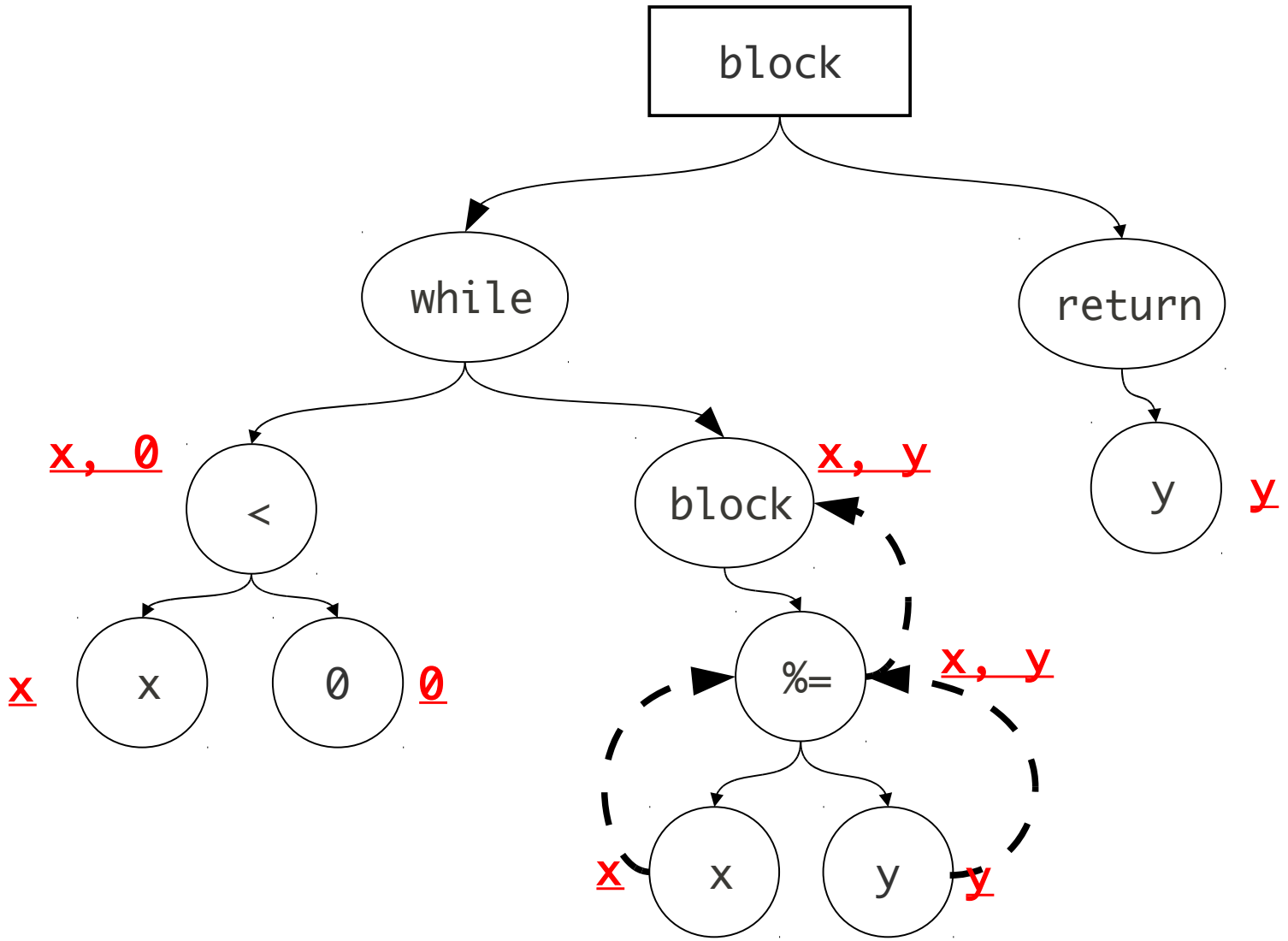
Lexemes Propagation



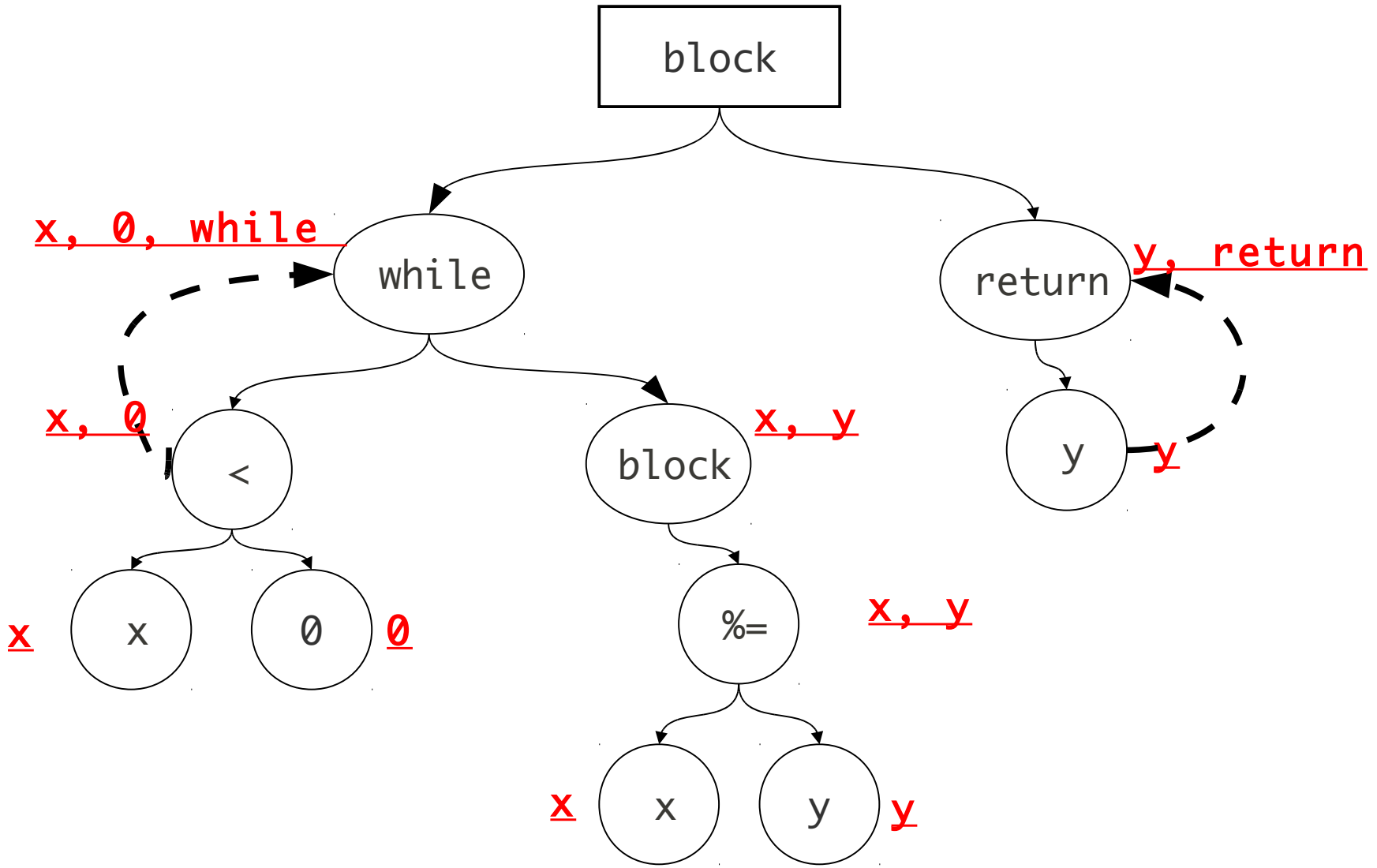
Lexemes Propagation



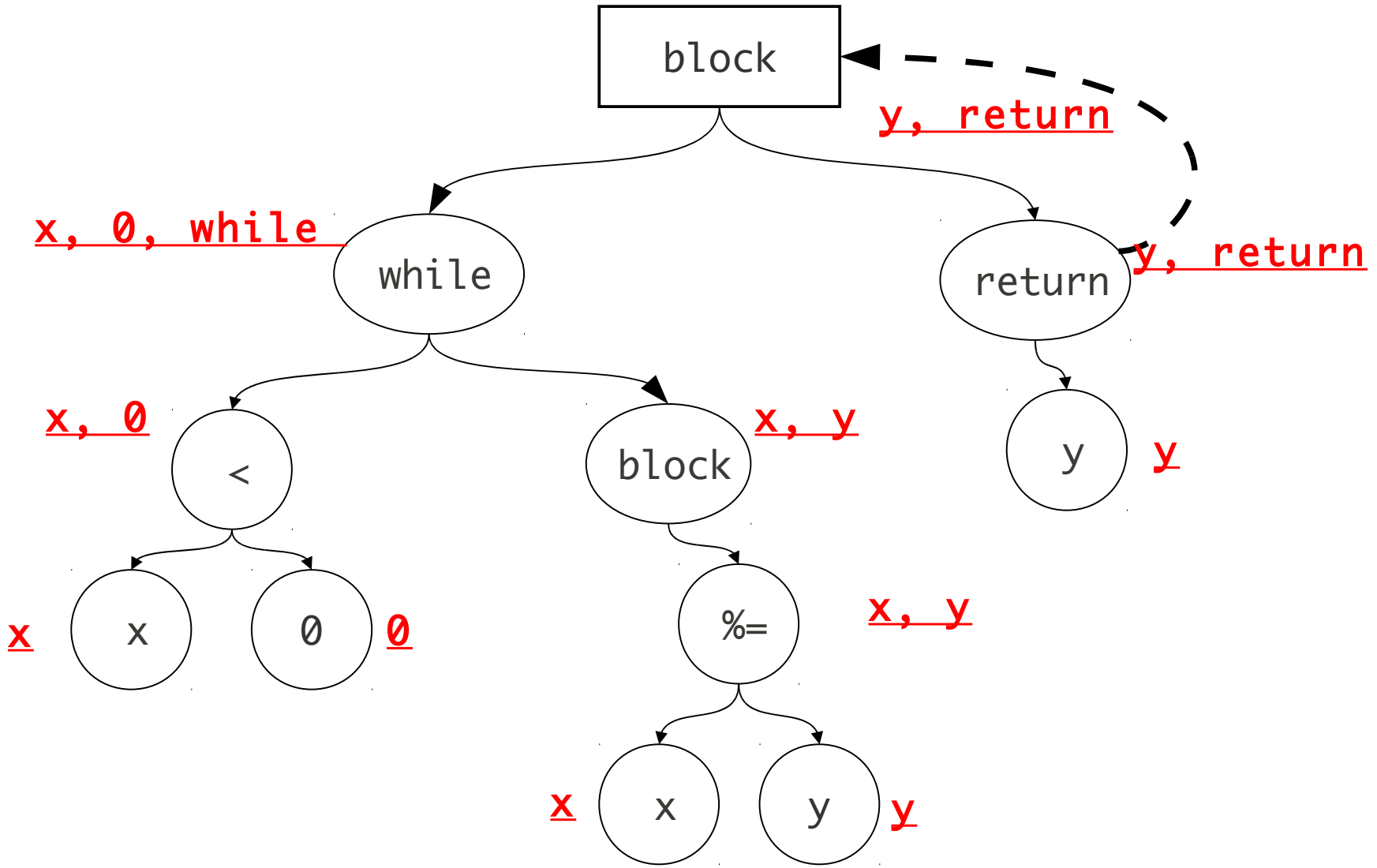
Lexemes Propagation



Lexemes Propagation



Lexemes Propagation



(2) Applying features in a Kernel

We exploits these features to compute similarity among pairs of nodes, as follows:

- `Instruction Class` filters comparable nodes
  - We compare only nodes with the same **Instruction Class**
- `Instruction`, `Context` and `Lexemes` are used to define a value of similarity between compared nodes

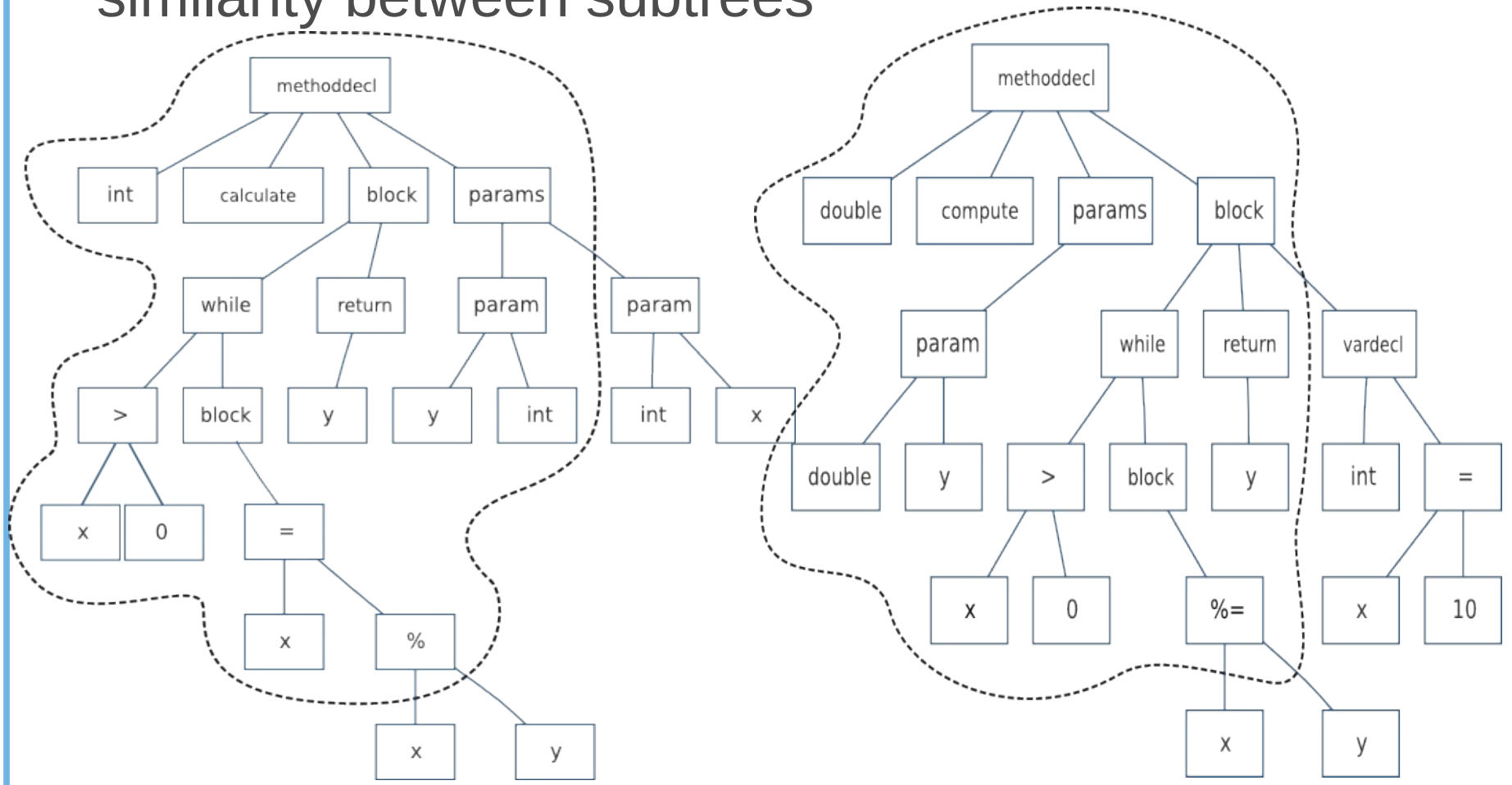
(*Primitive*) Kernel Function between nodes

s(n1,n2)=

- `1.0` If two nodes have the same values of features
- `0.8` If two nodes differ in lexemes (same `instruction and context`)
- `0.7` If two nodes share `lexemes` and are the same `instruction`
- `0.5` If two nodes share `lexemes` and are enclosed in the same `context`
- `0.25` If two nodes have at least one feature in common
- `0.0` no match

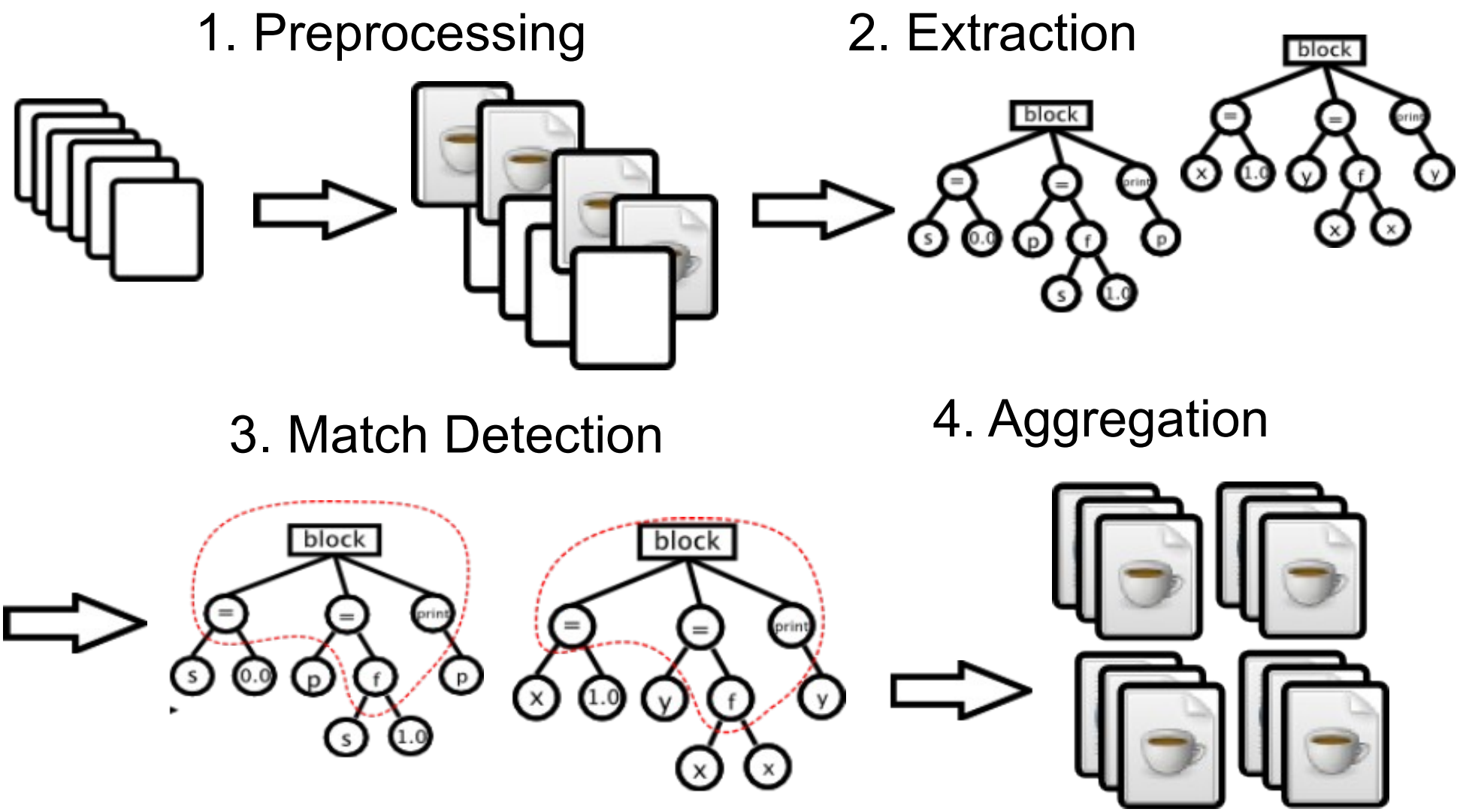
(3) Tree Kernel: Kernel on entire Tree Structures

- We apply nodes comparison recursively to compute similarity between subtrees



- We aim to identify the maximum isomorphic tree/subtree

Overall Process



A Preliminary evaluation

Evaluation Description

- We considered a small Java software system
  - We choose to identify clones at method level
- We checked system against the presence of up to Type 3 clones
  - Removed all detected clones through refactoring operations
- We manually and randomly injected a set of artificially created clones
  - One set for each type of clones
- We applied our prototype and CloneDigger* to mutated systems
- We evaluated performances in terms of Precision, Recall and F1

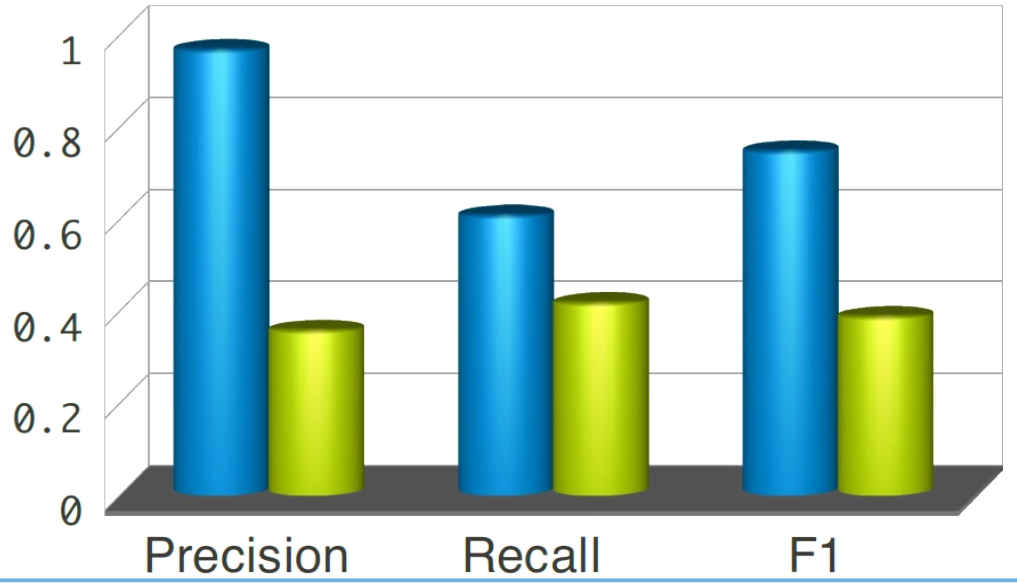
*http://clonedigger.sourceforge.net/

Results (1)

- Type 1 and Type 2 Clones:
  - We were able to detect all clones without any *false positive*
  - This was obtained **also by CloneDigger**
  - Both tools expressed the potential of AST-based approaches

Results (2)

- Type 3 clones:
  - We classified results as "*true Type 3 clones*" according to different thresholds on similarity values
  - We measured performance on different thresholds



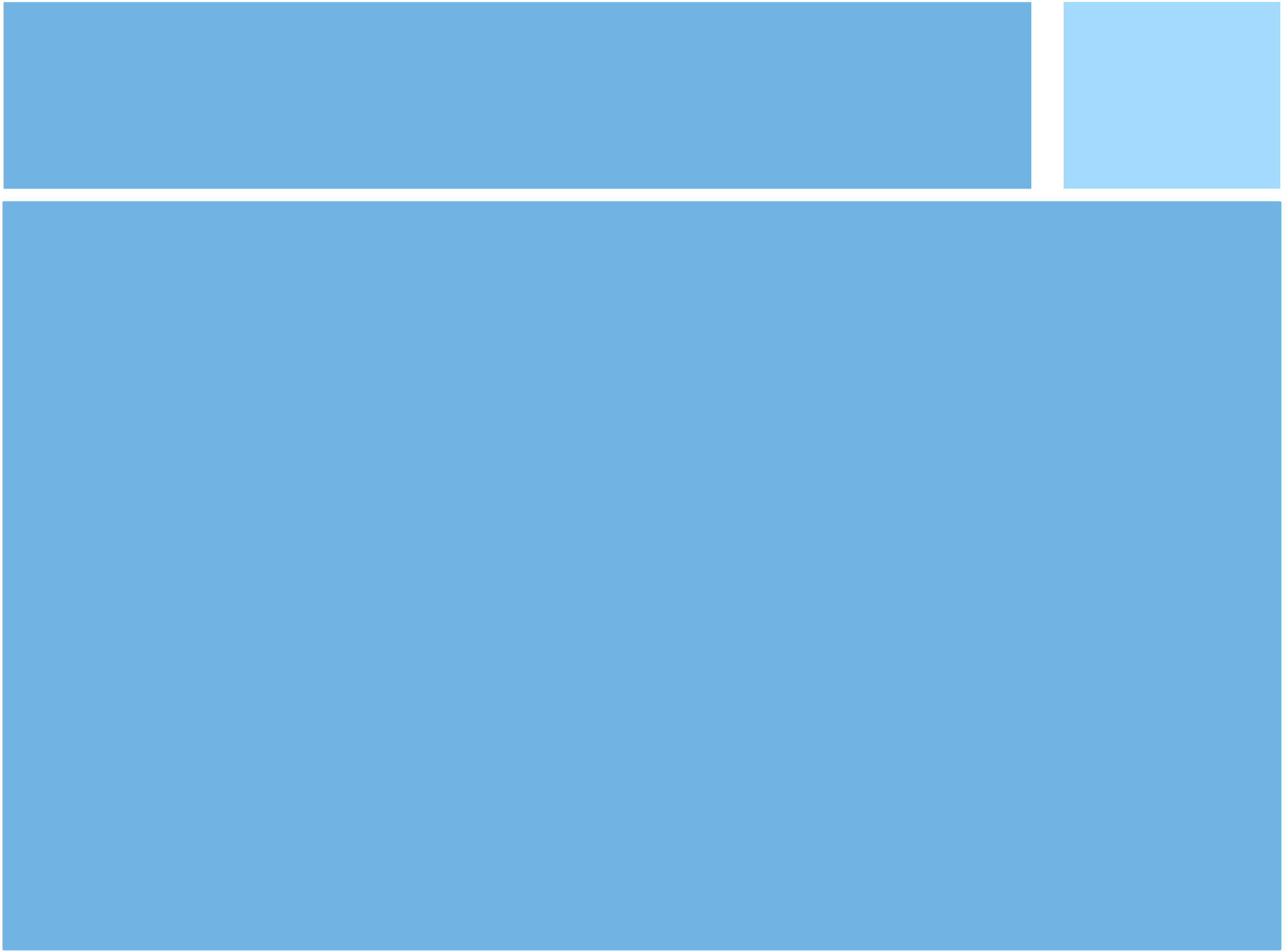
We get best results with threshold equals to `0.70`

Conclusions and Future Works

- Measure performance on real systems and projects
  - Bellon's Benchmark
  - Investigate best results with 0.7 as threshold
  - Measure Time Performances
- Improve the scalability of the approach
  - Avoid to compare all pairs
- Improve similarity computation
  - Avoid manual weighting features
- Extend Supported Languages
  - Now we support Java, C, Python

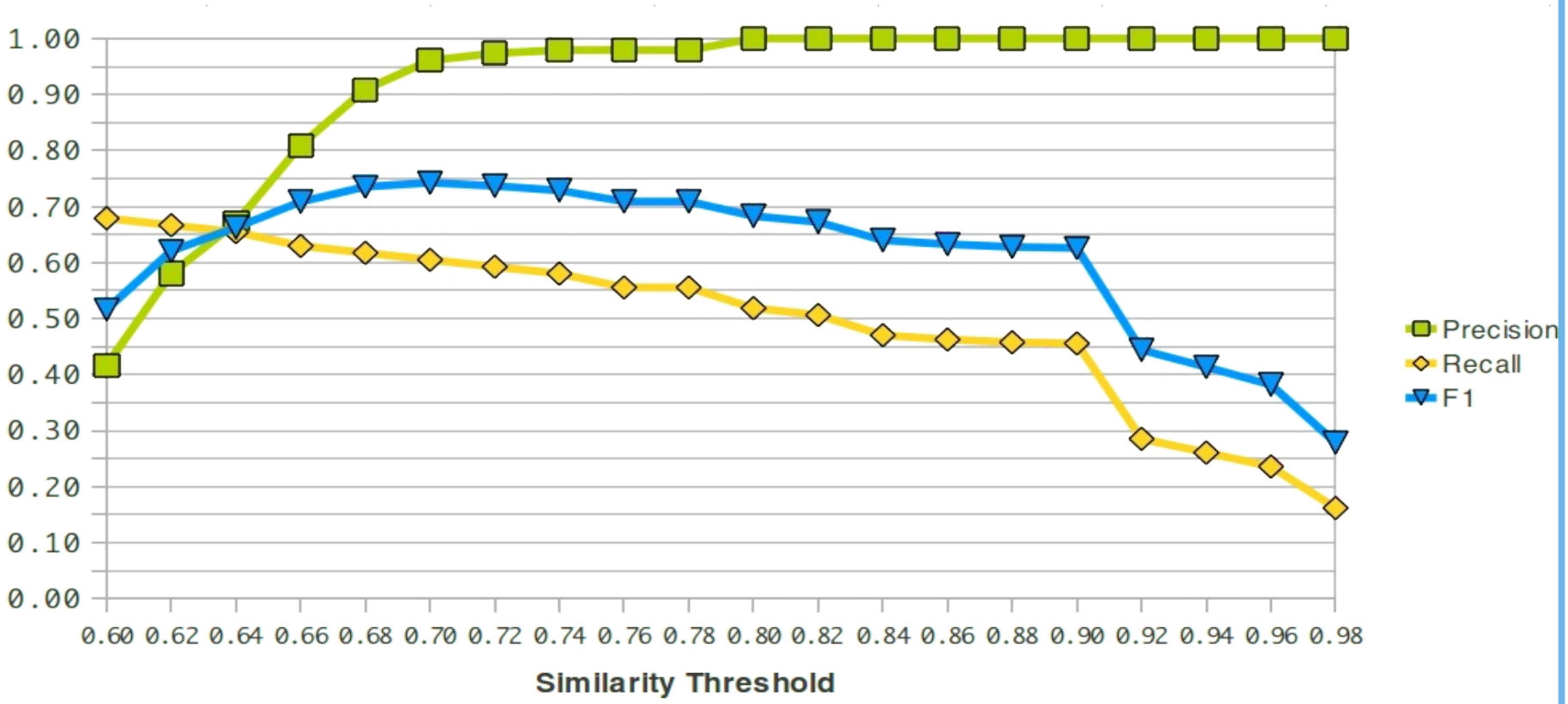
Thank you for listening.

Questions?



- Let's go to the backup slides

Evaluation on Different Thresholds



Exact Clone (Type 1):

- is an exact copy of consecutive code fragments without modifications (except for white spaces and comments)

```
public int getAccessibleChildrenCount(JComponent a) {
    int returnValue =
    ((ComponentUI) (uis.elementAt(0))).getAccessibleChildrenCount(a);
    for (int i = 1; i < uis.size(); i++) {
        ((ComponentUI) (uis.elementAt(i))).getAccessibleChildrenCount(a); }
    return returnValue; }
```

```
public int getAccessibleChildrenCount(JComponent a) {
    // Get AccessibleChildrenCount of the UI Component
    int returnValue = ((ComponentUI) (uis.elementAt(0))).
                                        getAccessibleChildrenCount(a);
    for (int i = 1; i < uis.size(); i++)
        ((ComponentUI) (uis.elementAt(i))).getAccessibleChildrenCount(a);
    return returnValue;
}
```

Exact Clone (Type 1):

- is an exact copy of consecutive code fragments without modifications (except for white spaces and comments)

```
public int getAccessibleChildrenCount(JComponent a) {
    int returnValue =
    ((ComponentUI) (uis.elementAt(0))).getAccessibleChildrenCount(a);
    for (int i = 1; i < uis.size(); i++) {
        ((ComponentUI) (uis.elementAt(i))).getAccessibleChildrenCount(a); }
    return returnValue; }
```

```
public int getAccessibleChildrenCount(JComponent a) {
    // Get AccessibleChildrenCount of the UI Component
    int returnValue = ((ComponentUI) (uis.elementAt(0))).
                                        getAccessibleChildrenCount(a);
    for (int i = 1; i < uis.size(); i++)
        ((ComponentUI) (uis.elementAt(i))).getAccessibleChildrenCount(a);
    return returnValue;
}
```

- **Parameter-substituted clone (Type 2):**
  - is a copy where only parameters (identifiers or literals) have been substituted

```
public static Color getForeground(AttributeSet a) {
    // Get the Foreground color
    Color fg = (Color) a.getAttribute(Foreground);
    if (fg == null)
        fg = Color.black;
    return fg; }
```

```
public static Color getBackground(AttributeSet a) {
    // Get the Background color
    Color fg = (Color) a.getAttribute(Background);
    if (fg == null) {
        fg = Color.black; }
    return fg; }
```

- **Parameter-substituted clone (Type 2):**
  - is a copy where only parameters (identifiers or literals) have been substituted

```
public static Color getForeground(AttributeSet a) {
    // Get the Foreground color
    Color fg = (Color) a.getAttribute(Foreground);
    if (fg == null)
        fg = Color.black;
    return fg; }
```

```
public static Color getBackground(AttributeSet a) {
    // Get the Background color
    Color fg = (Color) a.getAttribute(Background);
    if (fg == null) {
        fg = Color.black; }
    return fg; }
```

Structure-substituted clone (Type 3):

- is a copy where program structures have been substituted, added and/or deleted.

```
public void removeSelectionInterval(int index0, int index1) {
    if (index0 == -1 || index1 == -1) {
        return; }
    updateLeadAnchorIndices(index0, index1);
    int clearMin = Math.min(index0, index1);
    int clearMax = Math.max(index0, index1);
    changeSelection(clearMin, clearMax); }
```

```
public void removeSelectionInterval(int index0, int index1) {
    if (index0 == -1 || index1 == -1) {
        return; }
    updateLeadAnchorIndices(index0, index1);
    int clearMin = Math.min(index0, index1);
    int clearMax = Math.max(index0, index1);
    // Extend the removal to the end of the selection.
    if (getSelectionMode() != MULTIPLE_INTERVAL_SELECTION &&
                clearMin > minIndex && clearMax < maxIndex) {
        clearMax = maxIndex;  }
    changeSelection(clearMin, clearMax); }
```

Structure-substituted clone (Type 3):

- is a copy where program structures have been substituted, added and/or deleted.

```
public void removeSelectionInterval(int index0, int index1) {
    if (index0 == -1 || index1 == -1) {
        return; }
    updateLeadAnchorIndices(index0, index1);
    int clearMin = Math.min(index0, index1);
    int clearMax = Math.max(index0, index1);
    changeSelection(clearMin, clearMax); }
```

```
public void removeSelectionInterval(int index0, int index1) {
    if (index0 == -1 || index1 == -1) {
        return; }
    updateLeadAnchorIndices(index0, index1);
    int clearMin = Math.min(index0, index1);
    int clearMax = Math.max(index0, index1);
    // Extend the removal to the end of the selection.
    if (getSelectionMode() != MULTIPLE_INTERVAL_SELECTION &&
                clearMin > minIndex && clearMax < maxIndex) {
        clearMax = maxIndex;  }
    changeSelection(clearMin, clearMax); }
```