Lecture 8 and 9

Program Differencing
Agenda - Lecture 8 and 9

- Motivation for Program Differencing Techniques
- Problem Definition: What is a Program Differencing Problem?
- Lecture 8 (Today)
Agenda

- Lecture 9

- Lecture 10
  - Synthesis - Program Differencing Techniques
  - If time permits, Logical Structural Diff (LSdiff) by Kim & Notkin, ICSE 2009
Motivation: When do you use program differencing tools such as `diff`?

- Identify which change led to a bug
- Code reviews
- Generalization task
- Regression testing
Motivation of Program Differencing Techniques

- Code Reviews
- Software Version Merging
  - To detect possible conflicts among parallel updates
- Regression Testing
  - Prioritize or select test cases that need to be re-run by analyzing matched code elements
Motivation of Program Differencing Techniques

- Profile Propagation
- Mining Software Repositories Research
- Multi-Version Software Analysis
Multi-Version Analysis

Matching

Code Snippet

Interval

Time
Matching between Two Versions

Code Snippet

P1 P2
P3
P4
P5
P6

Two Version Matching

Time
Matching between Two Versions

Code Snippet

P1 P2 P3 P4 P5 P6

Two Version Matching
Matching between Two Versions

Code Snippet

P1  P2  P3  P4  P5  P6

Two Version Matching
Matching between Two Versions

Code Snippet

P1

P2

P3

P4 P5

P6

Two Version Matching
Matching between Two Versions

Code Snippet

P1 P2 P3 P4 P5 P6

Two Version Matching
Problem Definition: Program Differencing

• Input:
  • Two programs

• Output:
  • Differences between the two programs
  • Unchanged code fragments in the old version and their corresponding locations in the new version
Problem Definition: Program Differencing

Determine the \textit{differences} \( \Delta \) between \( O \) and \( N \).
For a code fragment \( nc \in N \), determine whether \( nc \in \Delta \).
If not, find \( nc \)'s corresponding origin \( oc \) in \( O \).
Characterization of Matching Problem

<table>
<thead>
<tr>
<th></th>
<th>e.g. diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Representation</td>
<td>string (a sequence of lines)</td>
</tr>
<tr>
<td>Matching Granularity</td>
<td>line</td>
</tr>
<tr>
<td>Matching Multiplicity</td>
<td>1:1</td>
</tr>
<tr>
<td>Matching Criteria / Heuristics</td>
<td>Two lines have the same sequence of characters.</td>
</tr>
</tbody>
</table>

Old File

- line 1
- line 2
- line 3
- line 4

New File

- line 1
- line 2
- line 3
- line 4
- line 5
- line 6
Recap of Lecture 8

• Comparison of two empirical study papers
  • Qualitative vs. Quantitative
  • Finding Hypothesis vs. Proving Hypothesis
• Moved on to Program Differencing
  • When do programmers use diff tools?
  • Motivation from software engineering research perspectives
• Characterization of Differencing Problem
  • Representation, Granularity, Multiplicity, Equivalence Criteria
Agenda Lecture 9

- Example
- String matching
  - diff (LCS) - class activity
- AST matching
- Yang 1992
- CFG matching (Jdiff)
  - Adam Duley’s presentation on Jdiff
  - Jdiff’s evaluation section
## Example

### Past

<table>
<thead>
<tr>
<th>p0</th>
<th>mA (){}</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>if (pred_a) {}</td>
</tr>
<tr>
<td>p2</td>
<td>foo()</td>
</tr>
<tr>
<td>p3</td>
<td>}</td>
</tr>
<tr>
<td>p4</td>
<td>}</td>
</tr>
<tr>
<td>p5</td>
<td>mB (b) {}</td>
</tr>
<tr>
<td>p6</td>
<td>a := 1</td>
</tr>
<tr>
<td>p7</td>
<td>b := b+1</td>
</tr>
<tr>
<td>p8</td>
<td>fun (a,b)</td>
</tr>
<tr>
<td>p9</td>
<td>}</td>
</tr>
</tbody>
</table>

### Current

<table>
<thead>
<tr>
<th>c0</th>
<th>mA (){}</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>if (pred_a0) {}</td>
</tr>
<tr>
<td>c2</td>
<td>if (pred_a) {}</td>
</tr>
<tr>
<td>c3</td>
<td>foo()</td>
</tr>
<tr>
<td>c4</td>
<td>}</td>
</tr>
<tr>
<td>c5</td>
<td>}</td>
</tr>
<tr>
<td>c6</td>
<td>}</td>
</tr>
<tr>
<td>c7</td>
<td>mB (b) {}</td>
</tr>
<tr>
<td>c8</td>
<td>b := b+1 \ c</td>
</tr>
<tr>
<td>c9</td>
<td>a := 1</td>
</tr>
<tr>
<td>c10</td>
<td>fun (a,b)</td>
</tr>
<tr>
<td>c11</td>
<td>}</td>
</tr>
</tbody>
</table>
String Matching : LCS

- The goal of `diff` is to report the minimum number of line changes necessary to convert one file into the other.
- `=>` to maximize the number of unchanged lines
Longest Common Subsequence

shanghai
shahaining
Longest Common Subsequence

- shahai
- shanghai
Longest Common Subsequence Algorithm

- Dynamic programming algorithm, $O(mn)$ in time and space
- Available operations are addition and deletion.
- Matched pairs cannot cross one another.
Dynamic Programming LCS: Step (1) Computing the length of LCS

function LCSLength (X[1..m], Y[1..n]) {
    C = array (0..m, 0..n)
    for row=0..m
        C[row,0] = 0;
    for col =0..n
        C[0,col] = 0
    for row=1..m
        for col =1..n
            if X[row] = Y[col]
                C[row,col] = C[row-1, col-1] +1
            else
                C[row,col] = max(C[row, col-1], C[row-1, col])
    return C[row, col]
function LCSLength (X[1..m], Y[1..n]) {
  C = array (0..m, 0..n)
  for row = 0..m
    C[row,0] = 0;
  for col = 0..n
    C[0,col] = 0
  for row = 1..m
    for col = 1..n
      if X[row] = Y[col]
        C[row,col] = C[row-1, col-1] + 1
      else
        C[row,col] = max(C[row, col-1], C[row-1, col])
  return C[row, col]
Dynamic Programming LCS:
Step (2) Reading out an LCS

function backTrace (C[0..m, 0..n], X[1..m], Y[1..n],
row, col) {
    if row=0 or col=0
        return ""
    else if X[row] = Y[col]
        return backTrace(C, X, Y, row-1, col-1) + X[row]
    else
        if C[row, col-1] > C[row-1, col]
            return backTrace(C, X, Y, row, col-1)
        else
            return backTrace(C, X, Y, row-1, col)
Line-level LCS based matching

<table>
<thead>
<tr>
<th>Past</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0 mA ()</td>
<td>c0 mA ()</td>
</tr>
<tr>
<td>p1 if (pred_a)</td>
<td>c1 if (pred_a0)</td>
</tr>
<tr>
<td>p2 foo()</td>
<td>c2 if (pred_a)</td>
</tr>
<tr>
<td>p3 }</td>
<td>c3 foo()</td>
</tr>
<tr>
<td>p4 }</td>
<td>c4 }</td>
</tr>
<tr>
<td>p5 mB (b)</td>
<td>c5 }</td>
</tr>
<tr>
<td>p6 a := 1</td>
<td>c6 }</td>
</tr>
<tr>
<td>p7 b := b+1</td>
<td>c7 mB (b)</td>
</tr>
<tr>
<td>p8 fun (a,b)</td>
<td>c8 b := b+1 \ c</td>
</tr>
<tr>
<td>p9 }</td>
<td>c9 a := 1</td>
</tr>
<tr>
<td></td>
<td>c10 fun (a,b)</td>
</tr>
<tr>
<td></td>
<td>c11 }</td>
</tr>
</tbody>
</table>
### Line-level LCS based matching

<table>
<thead>
<tr>
<th>Past</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>c0</td>
</tr>
<tr>
<td>mA ()</td>
<td>mA ()</td>
</tr>
<tr>
<td>p1</td>
<td>c1</td>
</tr>
<tr>
<td>if (pred_a)</td>
<td>if (pred_a0)</td>
</tr>
<tr>
<td>p2</td>
<td>c2</td>
</tr>
<tr>
<td>foo()</td>
<td>foo()</td>
</tr>
<tr>
<td>p3</td>
<td>c3</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>p4</td>
<td>c4</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>p5</td>
<td>c5</td>
</tr>
<tr>
<td>mB (b)</td>
<td>mB (b)</td>
</tr>
<tr>
<td>p6</td>
<td>c6</td>
</tr>
<tr>
<td>a := 1</td>
<td>a := l</td>
</tr>
<tr>
<td>p7</td>
<td>c7</td>
</tr>
<tr>
<td>b := b+1</td>
<td>b := b+1 \ c</td>
</tr>
<tr>
<td>p8</td>
<td>c8</td>
</tr>
<tr>
<td>fun (a,b)</td>
<td>fun (a,b)</td>
</tr>
<tr>
<td>p9</td>
<td>c9</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

EE382V Software Evolution: Spring 2009, Instructor Miryung Kim
What are assumptions of LCS algorithm?

- Assumptions
  - One-to-one mapping
  - No crossing blocks

- Limitations
  - When the equally likely LCSs are available, the output depends on implementation details of LCS.
What are assumptions of LCS algorithm?

- **Assumptions**
  - one-to-one mapping
  - no crossing matches

- **Limitations**
  - cannot find copy and paste
  - cannot detect moves
### Bdiff [Tichy 84]

<table>
<thead>
<tr>
<th></th>
<th>Diff</th>
<th>Bdiff [Tichy84]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basis</strong></td>
<td>Longest common subsequence</td>
<td>Minimal covering set</td>
</tr>
<tr>
<td><strong>Available operations</strong></td>
<td>Addition, deletion</td>
<td>Addition, deletion, move, copy, paste</td>
</tr>
<tr>
<td><strong>Multiplicity (S:T)</strong></td>
<td>1:1</td>
<td>n:1</td>
</tr>
<tr>
<td><strong>Assumption</strong></td>
<td>Linear ordering</td>
<td>Crossing block moves</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>sha ng hai</td>
<td>sha ng hai</td>
</tr>
<tr>
<td></td>
<td>sha hai ng</td>
<td>sha hai ng hai</td>
</tr>
</tbody>
</table>

- copy, paste and move operations are available
- crossing block moves are permitted
- one-to-one correspondences are not required
Abstract Syntax Tree Level Differencing

- Compare parse trees
- AST Node: token, variable name, or non-terminal expression
Abstract Syntax Tree

2 + 3

+ 2 3

if (x==2) x = x+3

= x 2

+ x 3
Yang 1992

```python
function simple_tree_matching(A, B)
if the roots of the two trees A and B contain distinct symbols, then
return (0)
m := the number of the first level subtrees of A
n := the number of the first level subtrees of B
Initialization M[i,0] := 0 for i=0,...,m, M[0,j] := 0 for j=0,...,n
for i:= 1 to m do
  for j:= 1 to n do
    M[i, j] = max (M[i, j-1], M[i-1,j] M[i-1,j-1]+W[i,j])
    where W[i,j] = simple_tree_matching (A_i, B_j) where A_i and B_j are the ith and jth first level subtrees of A and B
  end for
end for
return M[m,n]+1
```
Current

```
Body :=
  a
  1
:=
  b
 +
  b
  1
arg_list
fun
arg_list
a
b
If
pred_a
If
b
+
  a
  1
arg_list
b
pred_a0
pred_a
foo
arg_list
```
• Assumptions
  • respect parent-child relationships
  • the order between sibling nodes
• Limitations
  • sensitive to tree level changes
## AST-Based Matching

<table>
<thead>
<tr>
<th></th>
<th>Cdiff[Yan91]</th>
<th>[NFT05]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Differencing&lt;br&gt;Version merging</td>
<td>Understanding type evolution</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>LCS variation</td>
<td>Name matching (procedure)&lt;br&gt;Parallel graph traversal</td>
</tr>
<tr>
<td><strong>Strength</strong></td>
<td>Respect the parent-child relationship as well as the order between sibling nodes.</td>
<td>Identify renaming of types and variables.</td>
</tr>
<tr>
<td><strong>Weakness</strong></td>
<td>Very sensitive to tree level changes</td>
<td>Cannot match structurally different trees</td>
</tr>
</tbody>
</table>
Jdiff

- Adam Duley
Jdiff

- Step 1. Hierarchical name based matching: classes => methods
- Step 2. Per a pair of matched methods, create a pair of ECFGs.
- Step 3. Recursively match hammocks
  - Why do they match hammocks?
  - Why do they need a look-ahead (LH)?
  - Why do they need a similarity threshold (S)?
CFG-Based Matching (1)

- Hammock = a single entry, single exit subgraph in CFG

- Hammock node = a replacement node for a hammock graph
## CFG-Based Matching (2)

<table>
<thead>
<tr>
<th></th>
<th>[LS94]</th>
<th>Jdiff [AOH04]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representation</strong></td>
<td>CFG</td>
<td>ECFG</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Reduction to a hammock node Recursive expansion and comparison</td>
<td></td>
</tr>
<tr>
<td><strong>Node alignment</strong></td>
<td>DFS (LCS)</td>
<td>DFS (a look-ahead)</td>
</tr>
<tr>
<td><strong>Hammock node comparison</strong></td>
<td>Start node’s label</td>
<td>Ratio of matched nodes in a hammock</td>
</tr>
<tr>
<td><strong>Nested level</strong></td>
<td>Same level</td>
<td>Different levels</td>
</tr>
<tr>
<td><strong>Strength</strong></td>
<td></td>
<td>(+) Flexible matches (+) Robust to control structure changes</td>
</tr>
</tbody>
</table>
Evaluation of Jdiff

1. Measure Jdiff’s effectiveness for coverage estimation
   - Compared estimated coverage and actual coverage
   - This evaluation actually measures the effectiveness of Jdiff for the purpose that it was built for.

2. Measured JDiff’s performance for various values of lookahead and similarity parameters

3. Compared with Laski and Szermer’s algorithm
   - Measured % increases in the number of matched nodes
   - Q: Is the differencing algorithm more effective when it finds more matched nodes?
My general thoughts on Jdiff

- Algorithm that is based on CFG matching, yet customized for OO program’s characteristics: mainly dynamic binding & exception handling
- Introduction of several parameters to make the tool more robust to insertions and changes in nesting structure
- Thorough evaluation of Jdiff: answering three different research questions
Questions from Lecture 8

• What exactly is the goal of Kemerer & Slaughter’s paper?
• Applicability of Software Evolution Study?
• The “Halting Problem?”
• A method for choosing research methods / presenting results?
• Application principal component analysis or clustering?
  • e.g., See Nagapaan et al.
Survey

• Thank you for filling them out!
• Class activities
• Reading assignments
• Scheduling, etc.
Adjusting Schedule & Class Presentation

- Option 1.
  - Students voted for the option 1.
  - Your presentation is associated with the paper. So you may have to shift your presentation to a later date.

- Option 2.
  - Your presentation is associated with the date. So you have to present a different paper assigned for the date.
Preview for Next Monday

- Synthesis of program differencing techniques
- If you are doing a literature survey, this is a good paper to read.
Preview for Next Monday

- Discovering and Representing Systematic Code Changes, to appear in ICSE 2009, Miryung Kim and David Notkin
  - What kinds of questions that programmers ask when reviewing code?
  - What would you like to have an ideal program differencing tool?
  - Strengths and limitations of LSdiff / its evaluation
  - Any other applications of LSdiff other than code reviews?
Preview for Next Wednesday


- Association rule mining

- How can we recover transactions from CVS history?

- What are the objectives of their evaluation? Are they sufficiently validating their claims?