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)
  - String-matching based differencing techniques: Hunt1972 & Tichy1984.

Agenda

- Lecture 9
  - AST-based differencing techniques: Yang1992 & Neamtiu2005.
  - CFG-based program differencing technique (Jdiff): Apiwattanapong et al, 2004.
- 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

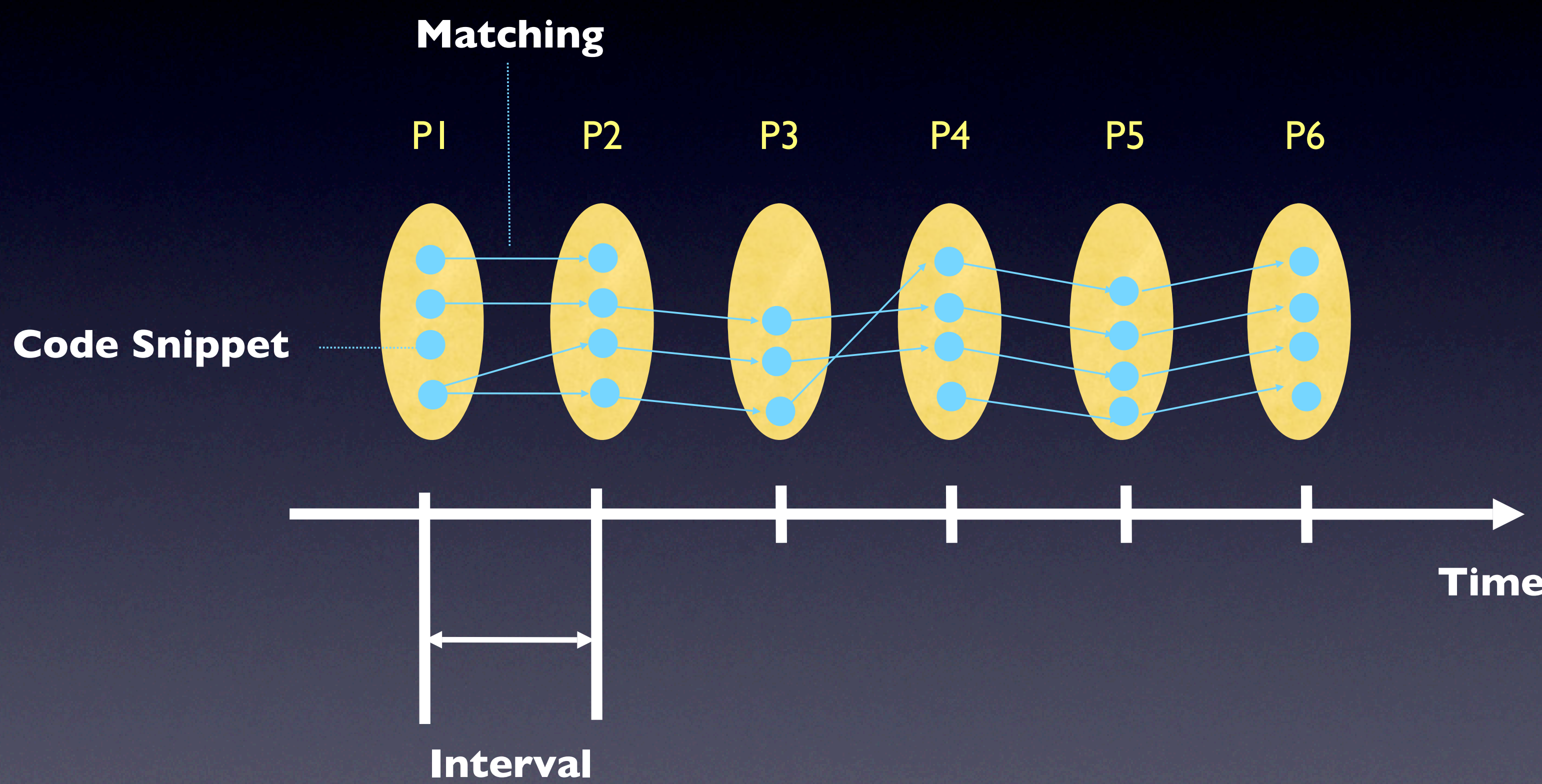
Motivation of Program Differencing Techniques

- 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

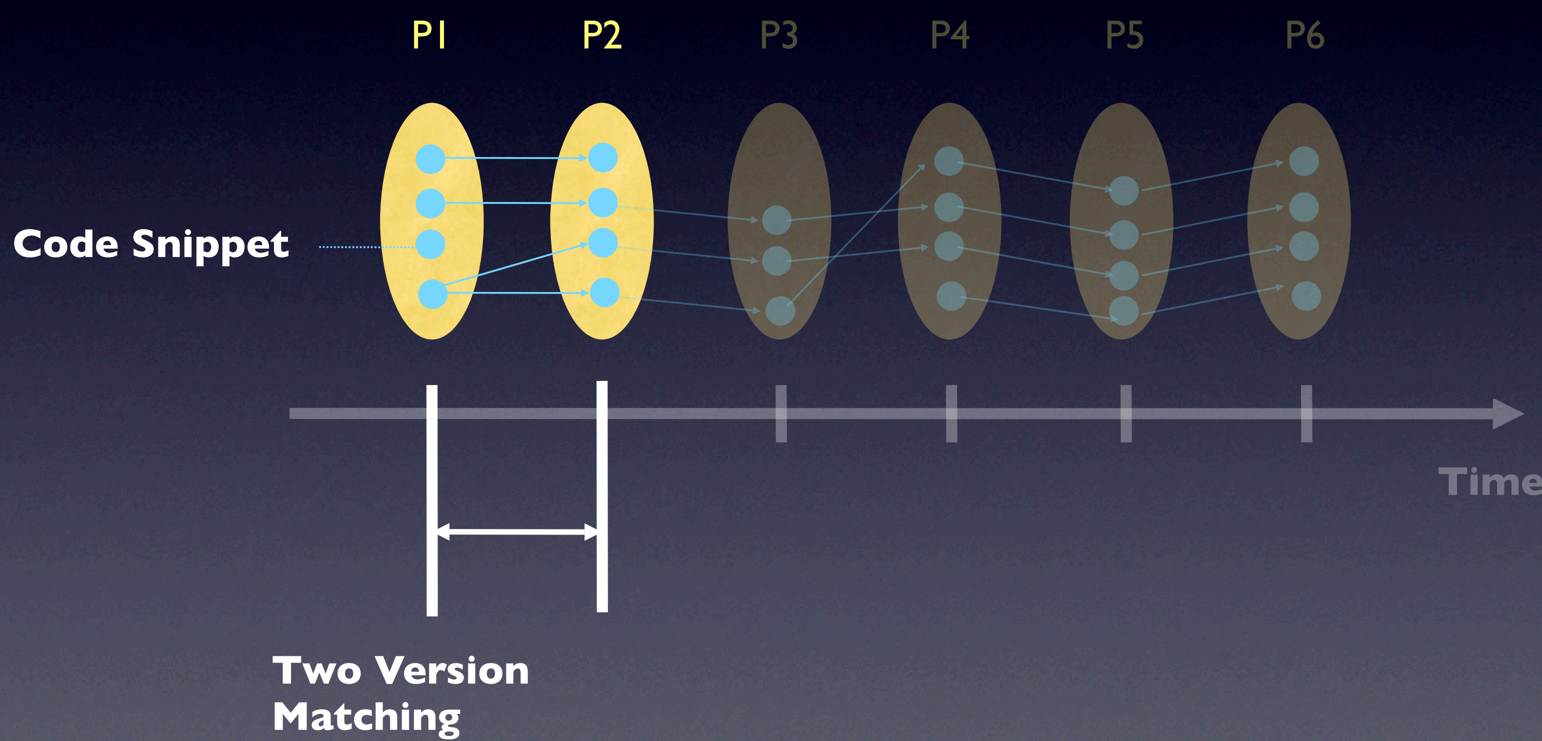
Motivation of Program Differencing Techniques

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

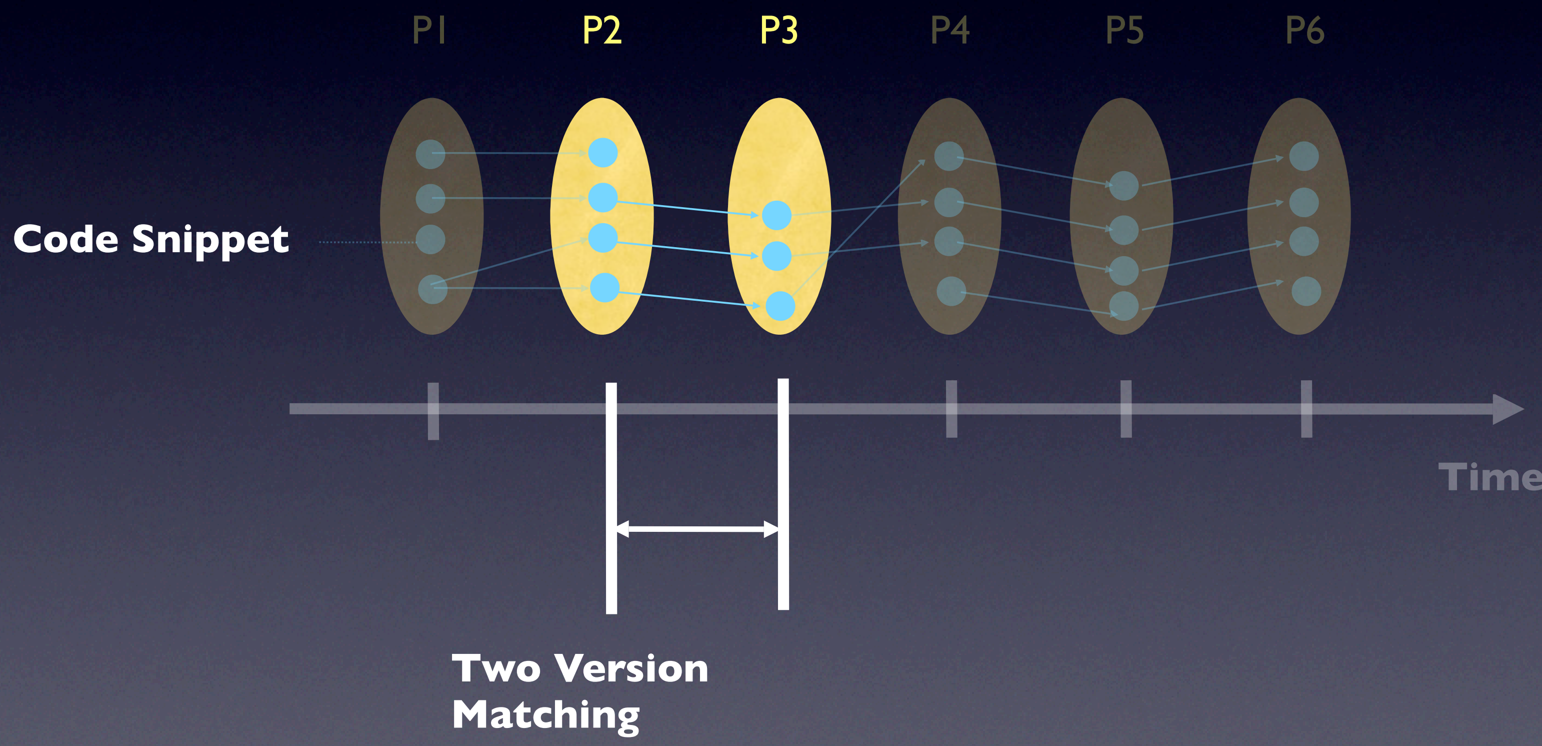
Multi-Version Analysis



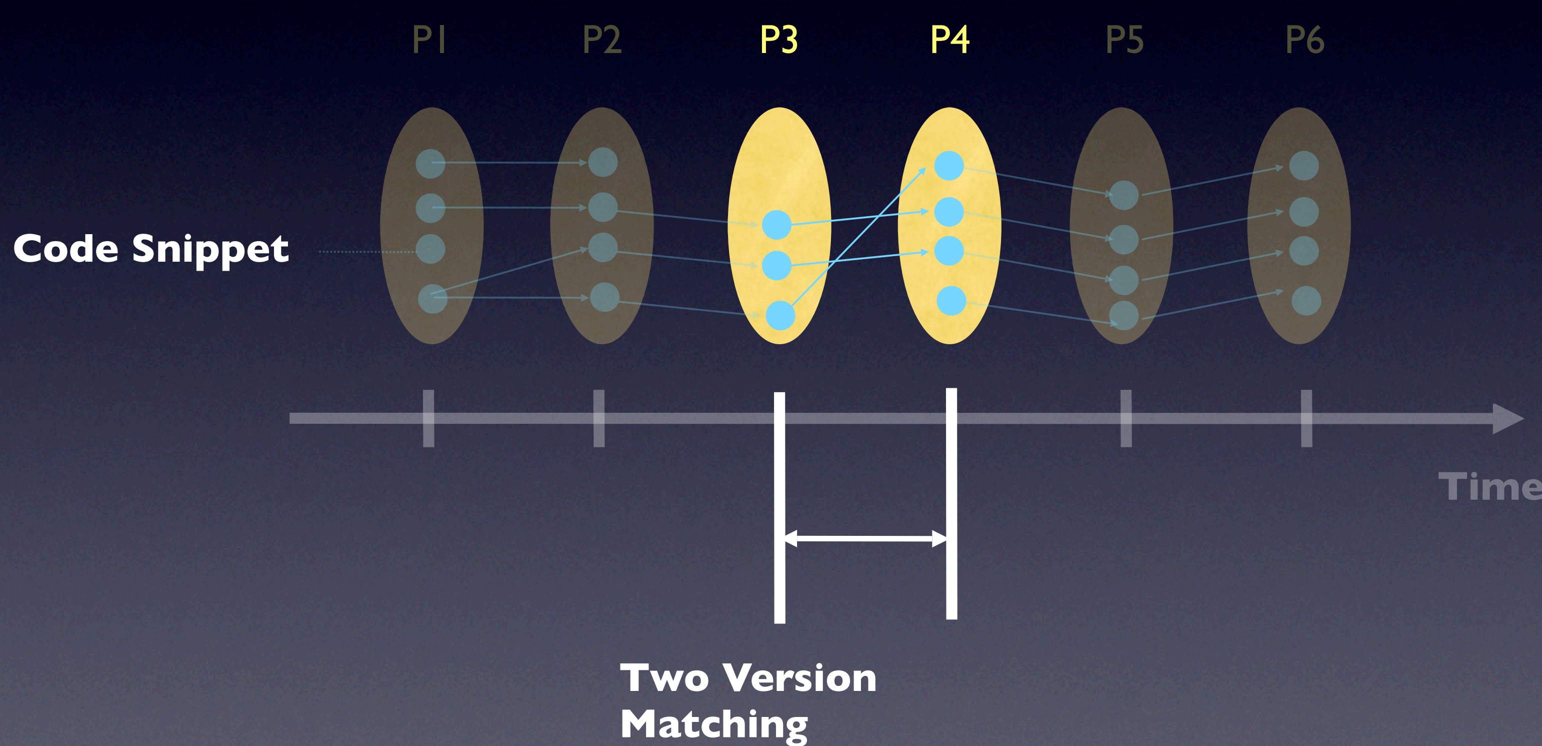
Matching between Two Versions



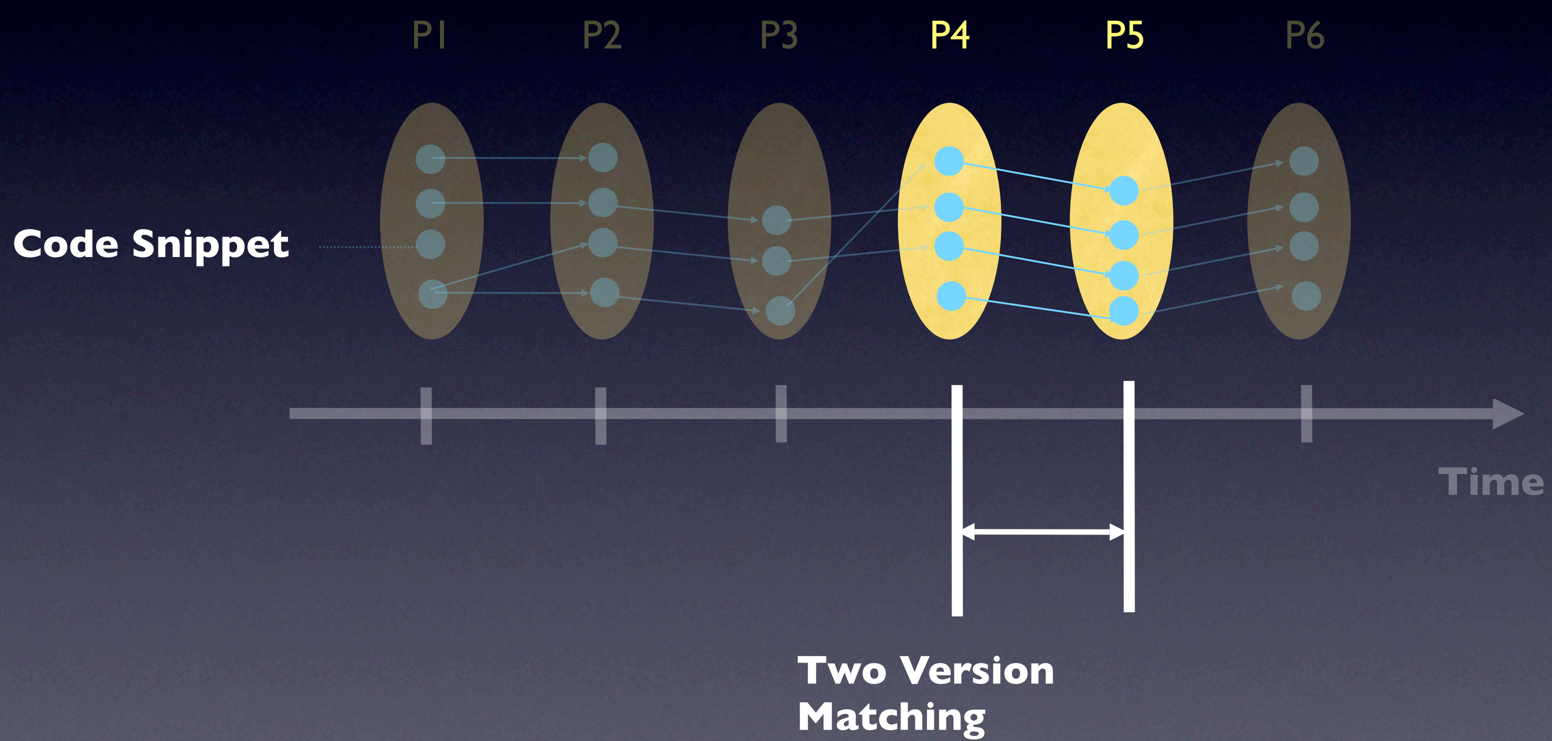
Matching between Two Versions



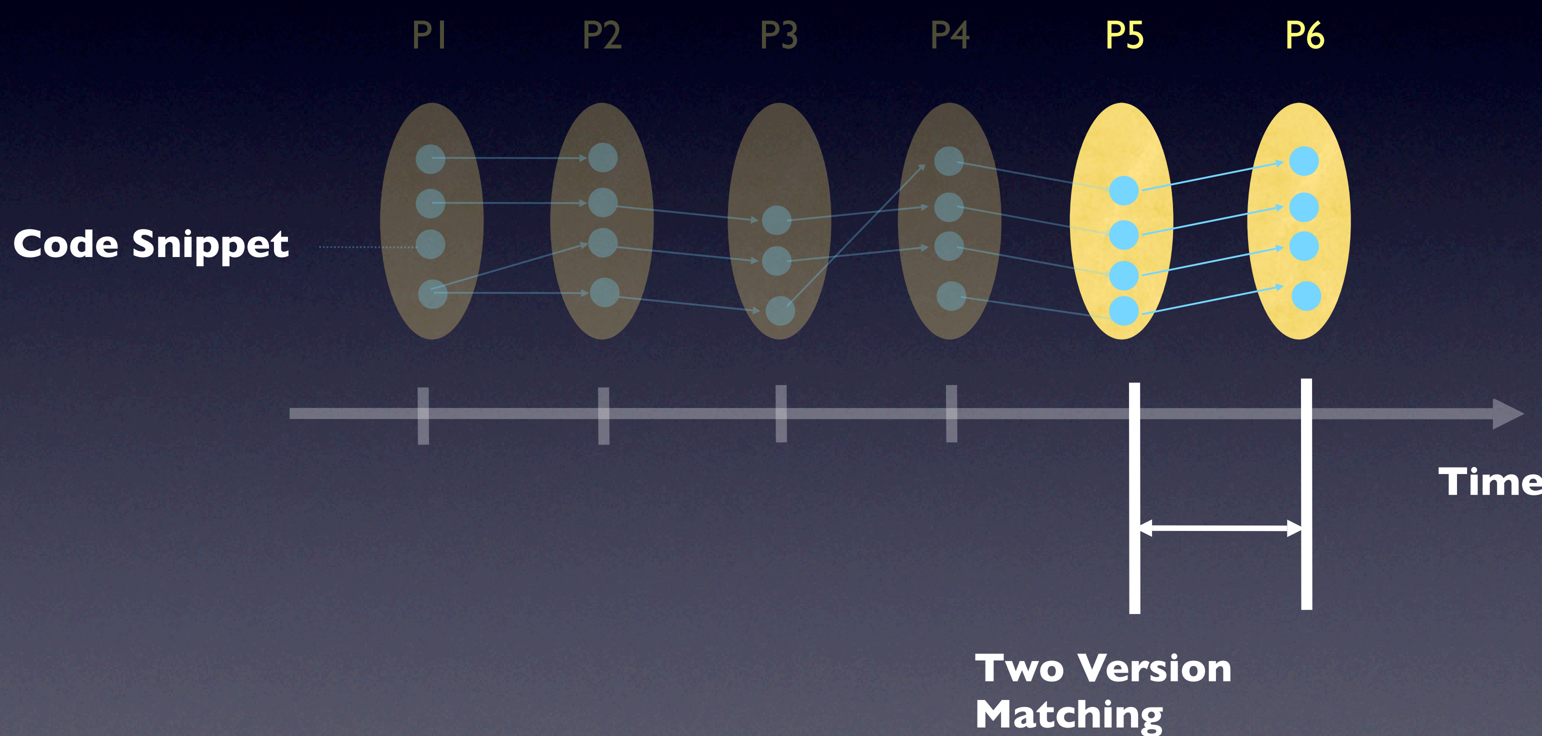
Matching between Two Versions



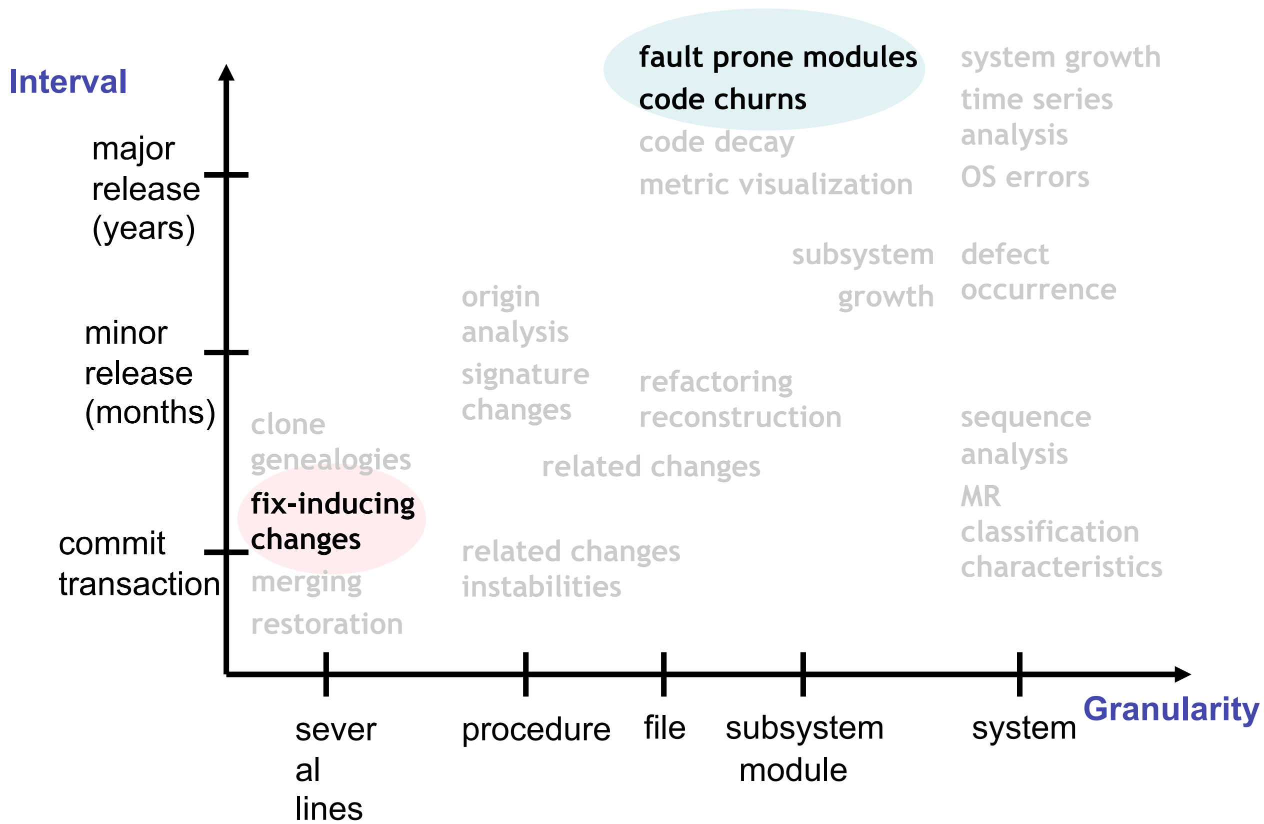
Matching between Two Versions



Matching between Two Versions



Multi-Version Program Analyses



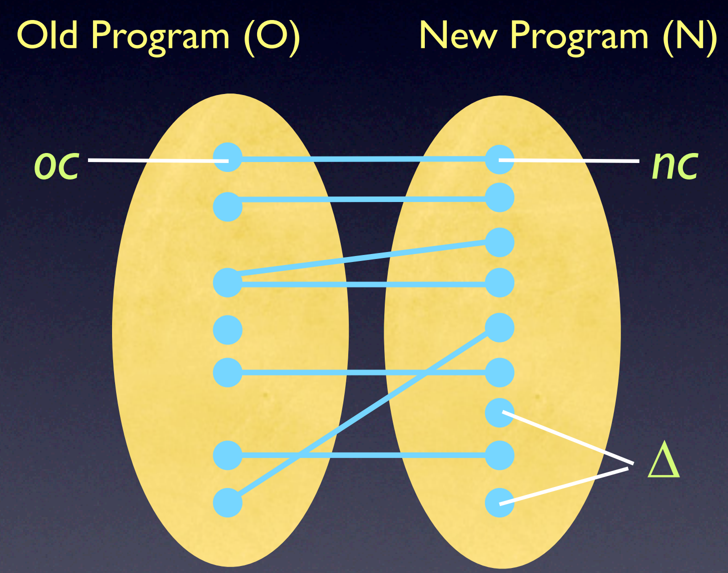
Problem Definition: Program Differencing

- 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

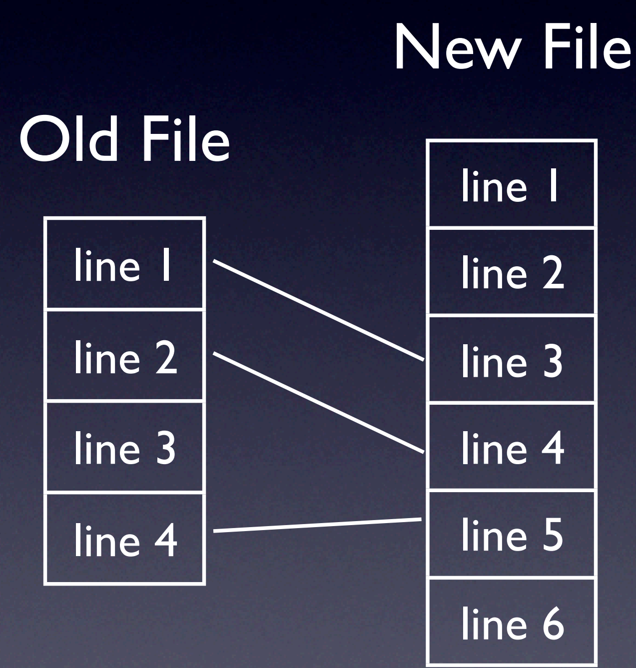
*Determine the **differences** Δ between O and N.*
For a code fragment nc ∈ N, determine whether nc ∈ Δ.

If not, find nc's corresponding origin oc in O.



Characterization of Matching Problem

	e.g. *diff*
Program Representation	string (a sequence of lines)
Matching Granularity	line
Matching Multiplicity	1:1
Matching Criteria / Heuristics	Two lines have the same sequence of characters.



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	Current
p0 mA (){	c0 mA (){
p1 if (pred_a) {	c1 if (pred_a0) {
p2 foo()	c2 if (pred_a) {
p3 }	c3 foo()
p4 }	c4 }
p5 mB (b) {	c5 }
p6 a := 1	c6 }
p7 b := b+1	c7 mB (b) {
p8 fun (a,b)	c8 b := b+1 \\ c
p9 }	c9 a := 1
	c10 fun (a,b)
	c11 }

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

s	h	a	n	g	h	a	i
s	h	a	h	a	i	n	g

Longest Common Subsequence

s	h	a	n	g	h	a	i
s	h	a	h	a	i	n	g

- shahai

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]
```

		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11
	0	0	0	0	0	0	0	0	0	0	0	0	0
p0	0												
p1	0												
p2	0												
p3	0												
p4	0												
p5	0												
p6	0												
p7	0												
p8	0												
p9	0												

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]
```

		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11
	0	0	0	0	0	0	0	0	0	0	0	0	0
p0	0	**1**	1	1	1	1	1	1	1	1	1	1	1
p1	0	1	1	**2**	2	2	2	2	2	2	2	2	2
p2	0	1	1	2	**3**	3	3	3	3	3	3	3	3
p3	0	1	1	2	3	**4**	**4**	**4**	4	4	4	4	**4**
p4	0	1	1	2	3	**4**	**5**	**5**	5	5	5	5	**5**
p5	0	1	1	2	3	4	5	5	**6**	6	6	6	6
p6	0	1	1	2	3	4	5	5	6	6	**7**	7	7
p7	0	1	1	2	3	4	5	5	6	**7**	7	7	7
p8	0	1	1	2	3	4	5	5	6	7	7	**8**	8
p9	0	1	1	2	3	**4**	**5**	**6**	6	7	7	8	**9**

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)
```

		c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11
	0	0	0	0	0	0	0	0	0	0	0	0	0
p0	0	1	1	1	1	1	1	1	1	1	1	1	1
p1	0	1	1	2	2	2	2	2	2	2	2	2	2
p2	0	1	1	2	3	3	3	3	3	3	3	3	3
p3	0	1	1	2	3	4	4	4	4	4	4	4	4
p4	0	1	1	2	3	4	5	5	5	5	5	5	5
p5	0	1	1	2	3	4	5	5	6	6	6	6	6
p6	0	1	1	2	3	4	5	5	6	6	7	7	7
p7	0	1	1	2	3	4	5	5	6	7	7	7	7
p8	0	1	1	2	3	4	5	5	6	7	7	8	8
p9	0	1	1	2	3	4	5	6	6	7	7	8	9

Line-level LCS based matching

Past		Current	
p0	mA (){	c0	mA (){
p1	if (pred_a) {	c1	if (pred_a0) {
p2	foo()	c2	if (pred_a) {
p3	}	c3	foo()
p4	}	c4	}
p5	mB (b) {	c5	}
p6	a := 1	c6	}
p7	b := b+1	c7	mB (b) {
p8	fun (a,b)	c8	b := b+1 \\ c
p9	}	c9	a := 1
		c10	fun (a,b)
		c11	}

Line-level LCS based matching

Past		Current	
p0	mA (){	c0	mA (){
p1	if (pred_a) {	c1	if (pred_a0) {
p2	foo()	c2	if (pred_a) {
p3	}	c3	foo()
p4	}	c4	}
p5	mB (b) {	c5	}
p6	a := 1	c6	}
p7	b := b+1	c7	mB (b) {
p8	fun (a,b)	c8	b := b+1 \\ c
p9	}	c9	a := 1
		c10	fun (a,b)
		c11	}

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]

	Diff	Bdiff [Tichy84]
Basis	Longest common subsequence	Minimal covering set
Available operations	Addition, deletion	Addition, deletion, move, copy, paste
Multiplicity (S:T)	1:1	n:1
Assumption	Linear ordering	Crossing block moves
Example	sha ng hai sha hai ng	sha ng hai sha hai ng hai

- copy, paste and move operations are available
- crossing block moves are permitted
- one-to-one correspondences are not required

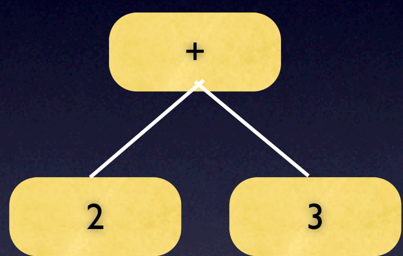
Abstract Syntax Tree Level Differencing

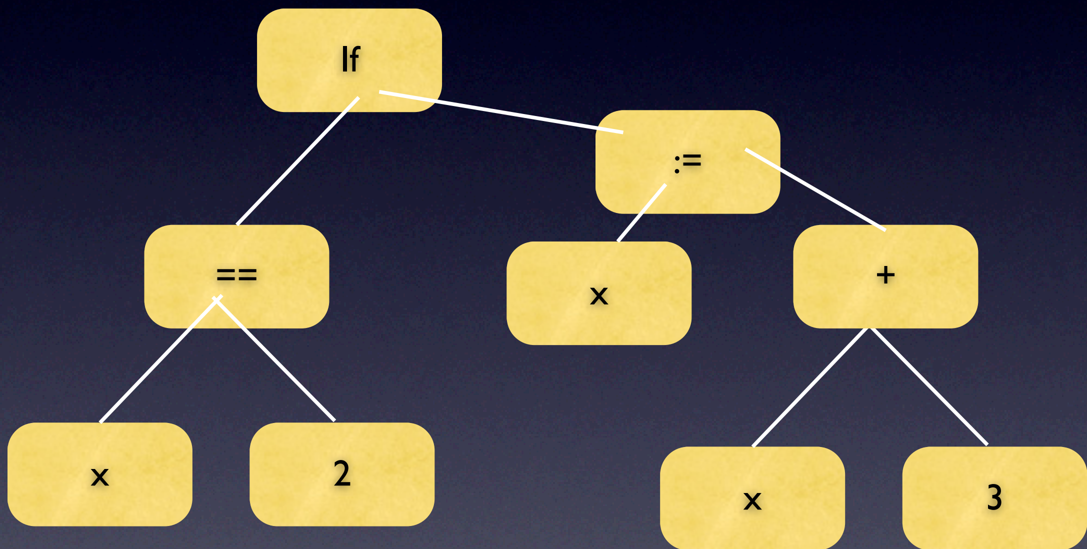
- Compare parse trees
- AST Node: token, variable name, or non-terminal expression

Abstract Syntax Tree

2 + 3

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



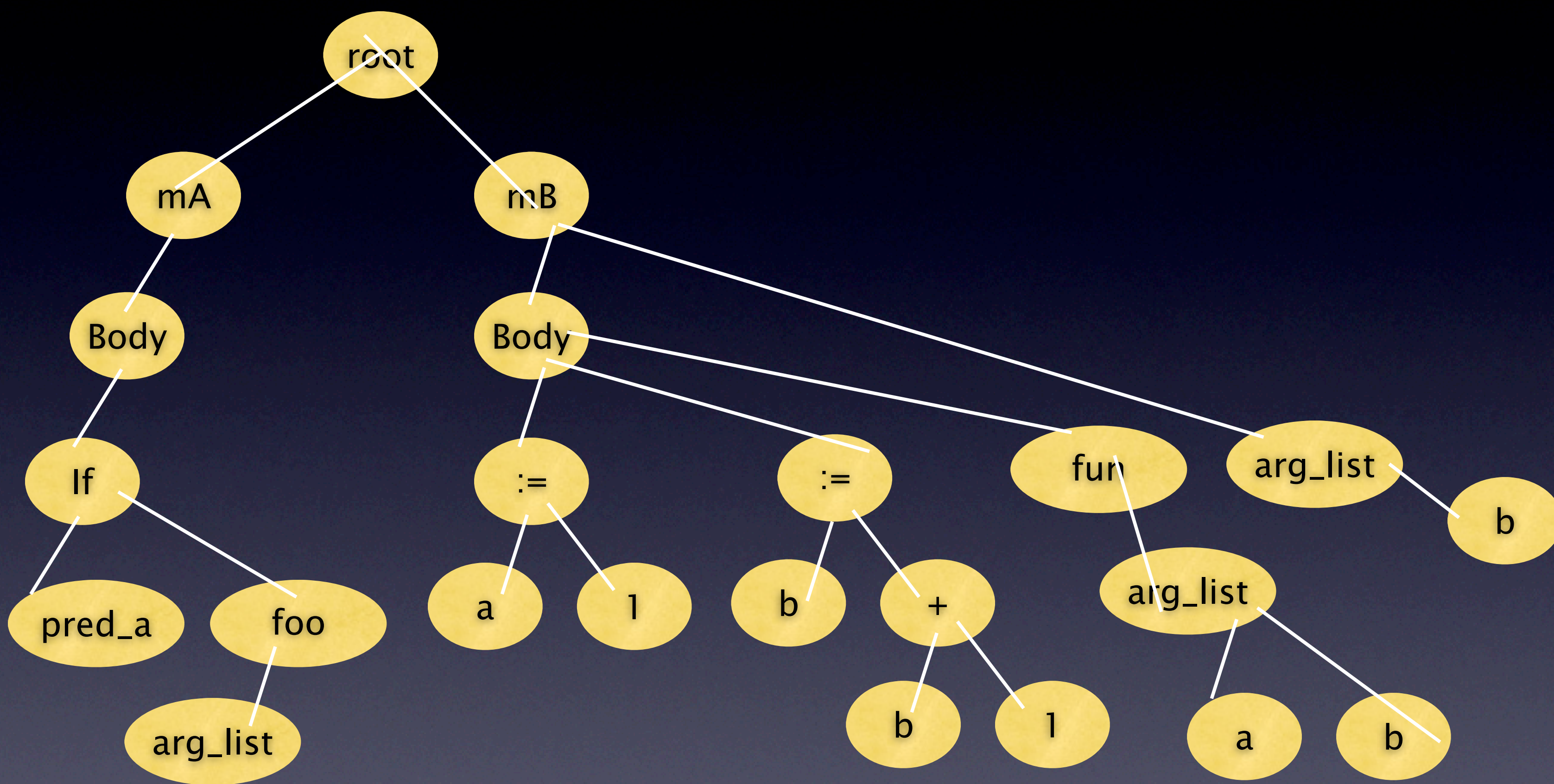


Yang 1992

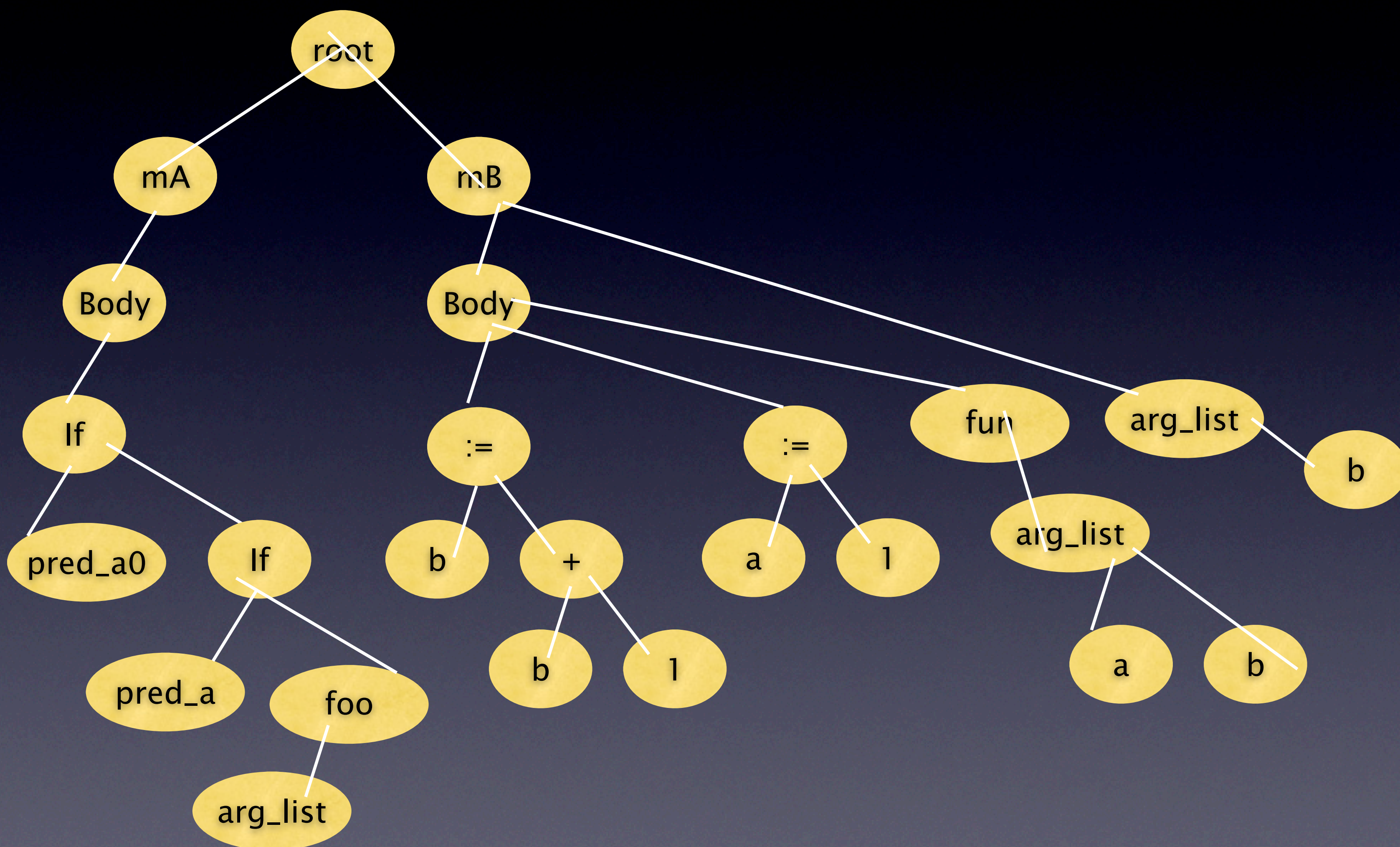
```
 function simple_tree_matching(A, B)
if the roots of the two trees A and B contain distinct symbols, then
return (0)
m := the number of the first level subtrees of A
n := the number of the first level subtrees of B
Initialization M [i,0] := 0 for i=0, .., m, M[0,j]:= 0 for j=0,...,n

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
```

Past


root
mA
mB
Body
Body
If
:=
:=
fun
arg_list
b
pred_a
foo
a
1
b
+
arg_list
arg_list
b
1
a
b

Current


root
mA
mB
Body
Body
If
:=
:=
fun
arg_list
b
pred_a0
If
b
+
a
1
arg_list
pred_a
foo
b
1
a
b
arg_list

- Assumptions
  - respect parent-child relationships
  - the order between sibling nodes
- Limitations
  - sensitive to tree level changes

AST-Based Matching

	Cdiff[Yan91]	[NFT05]
Goal	Differencing Version merging	Understanding type evolution
Algorithm	LCS variation	Name matching (procedure) Parallel graph traversal
Strength	Respect the parent-child relationship as well as the order between sibling nodes.	Identify renaming of types and variables.
Weakness	Very sensitive to tree level changes	Cannot match structurally different trees

Jdiff

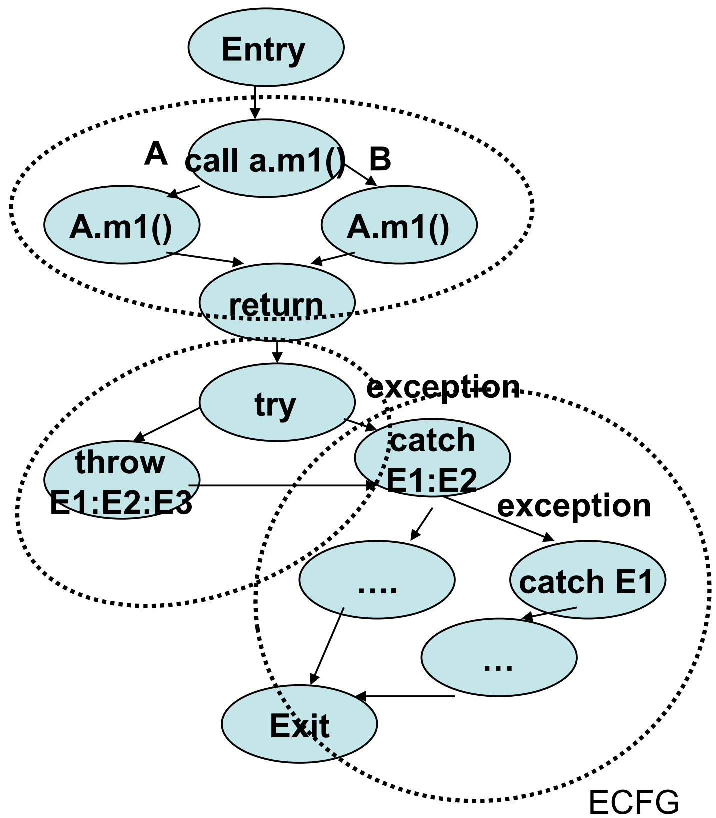
- Adam Duley

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)

	[LS94]	Jdiff [AOH04]
Representation	CFG	ECFG
Algorithm	Reduction to a hammock node Recursive expansion and comparison	
Node alignment	DFS (LCS)	DFS (a look-ahead)
Hammock node comparison	Start node's label	Ratio of matched nodes in a hammock
Nested level	Same level	Different levels
Strength		(+) Flexible matches (+) Robust to control structure changes

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
    - Miryung Kim and David Notkin. "Program element matching for multi-version program analyses". In Proceedings of the International Workshop on Mining Software Repositories, pages 58–64, 2006.
    - 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

- Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. "Mining version histories to guide software changes", IEEE Transactions on Software Engineering, 31(6):429–445, 2005.
  - Association rule mining
  - How can we recover transactions from CVS history?
  - What are the objectives of their evaluation? Are they sufficiently validating their claims?