Lecture 10

Synthesis of Program Differencing Techniques
Logical Structural Diff

Today's Agenda

- Discuss Yang's syntactic *diff*
- Synthesis of Program Differencing Techniques
- Logical Structural *diff*

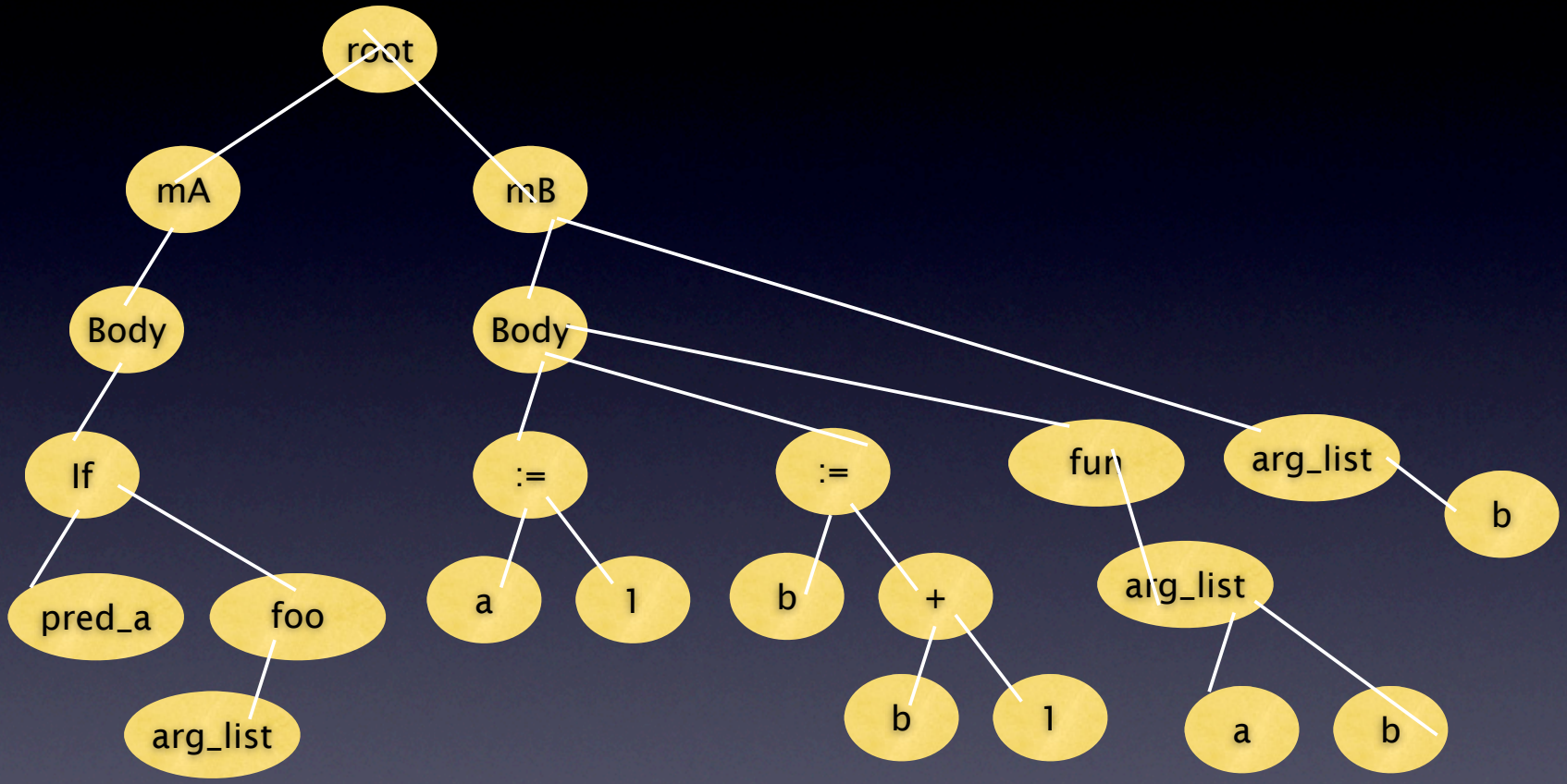
Example

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 }

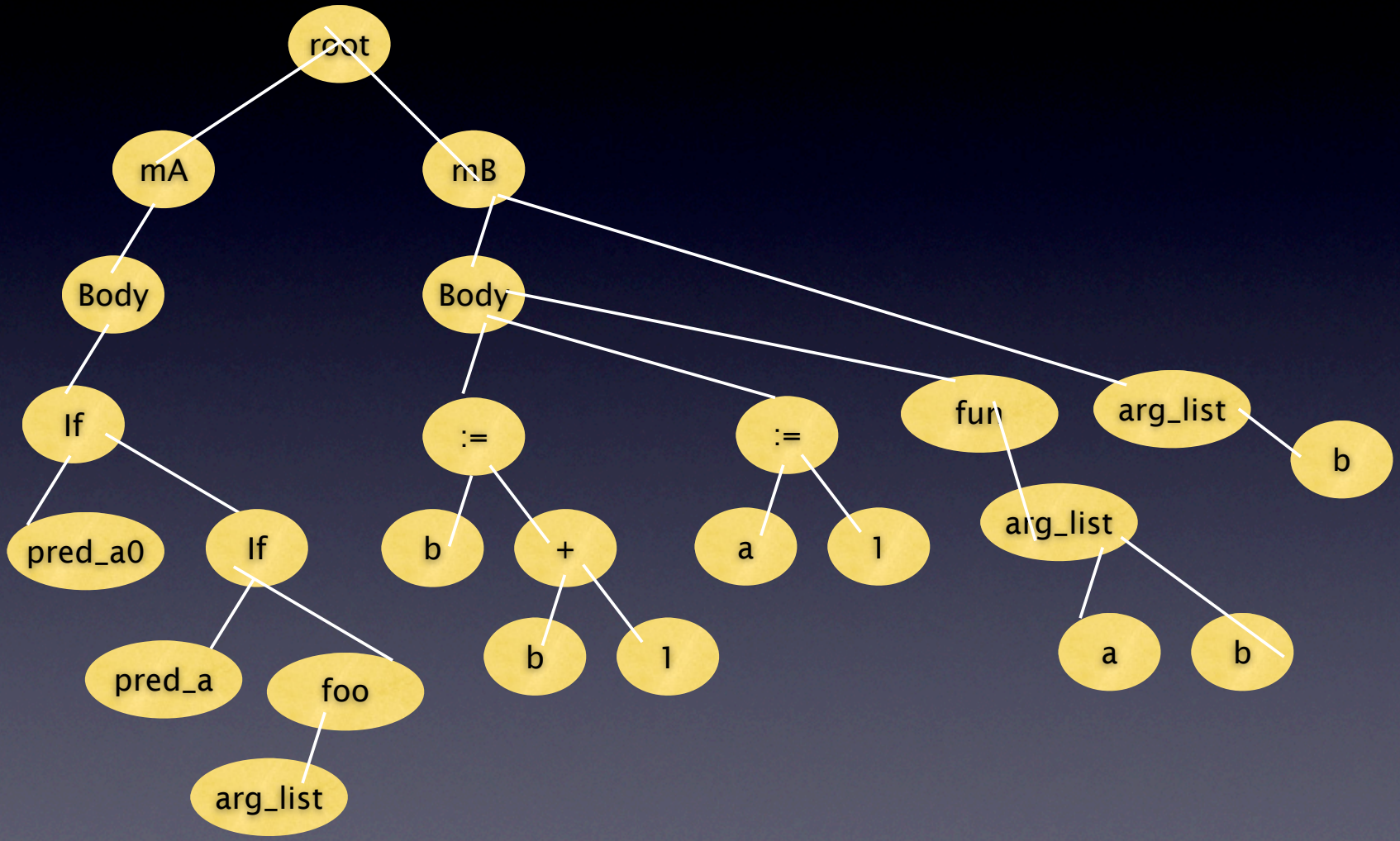
Yang 1992

```
 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
```

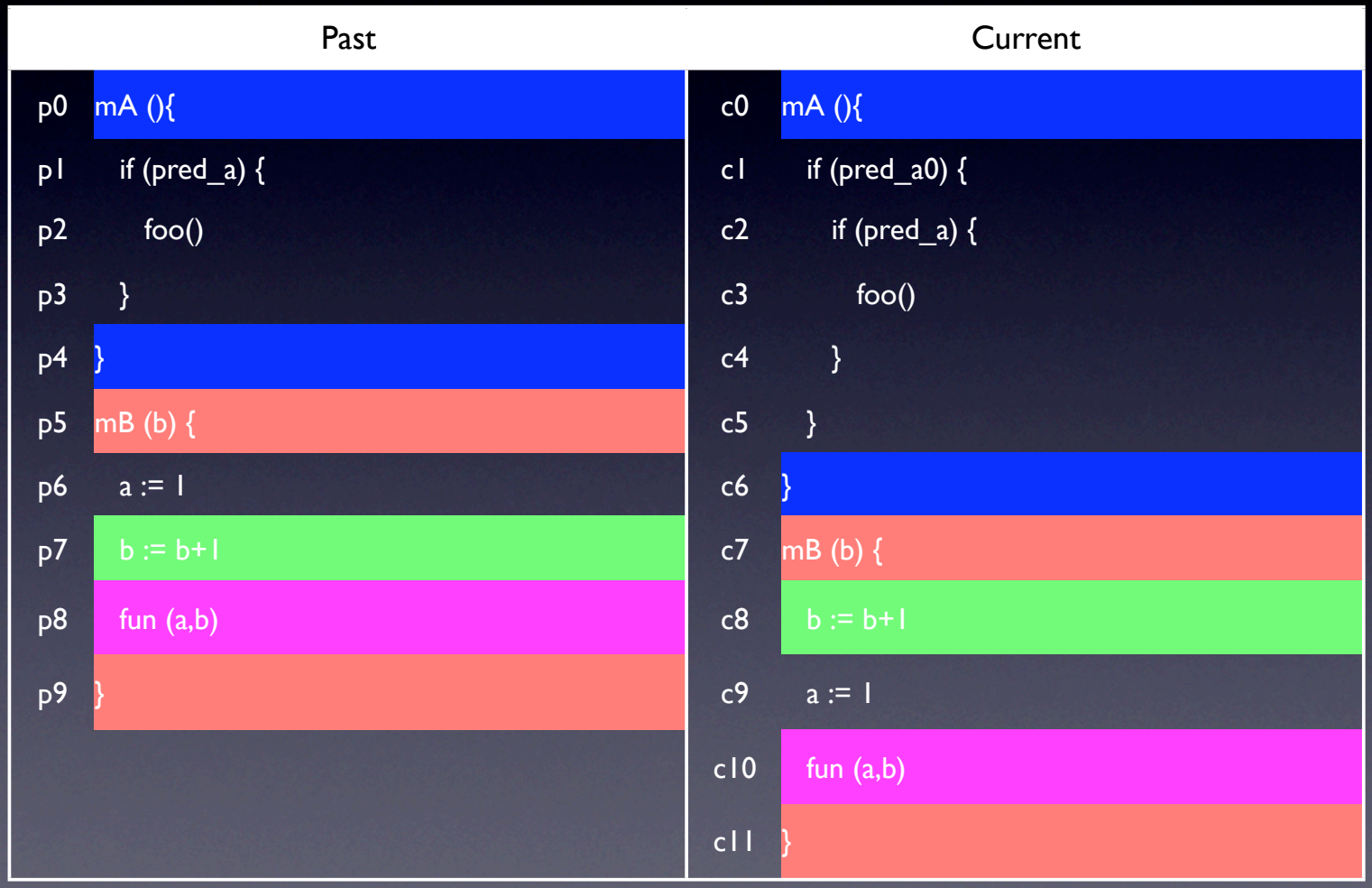
Past


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

AST matching



Program Differencing Techniques

- Problem: Computing semantic differences requires solving the problem of semantic program equivalence, which is an undecidable problem.
- Solution: The problem is approximated by matching a code element by its syntactic and textual similarity.
-

Characterization of differencing techniques

- Differencing techniques depends on the choices of
  1. an underlying program representation
  2. matching granularity
  3. matching multiplicity
  4. matching heuristics

What are the challenges of building program differencing techniques?

- Various Granularity Support / Problem of which granularity to support
  - Fine-grained vs course grained --> what to use this for?
- Consideration of representing diff results / matches
- Do you consider refactoring or not?
  - Ground truth
  - Lack of benchmarks
- Copy and paste
- Program languages

Challenges of Program Differencing (Code Matching)

- Absence of Benchmarks
  - Low inter-rater agreement
  - Programmers may have to inspect results
- Various Granularity Support
  - Depends on which application it will be used for
- Types of Code Changes
  - Merging, splitting, and renaming of procedures, files, etc.
  - Copy and paste

Comparison

Matching Technique	Program Representation	Granularity	Multiplicity (O:N)	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✔		
diff [HS77]	String	Line	1:1			✔
bdiff [Tic84]	String	Line	1:n			✔
cdiff [Yang91]	AST	AST node	1:1	✔		✔
Neamtiu et al.	AST	Type, Variable	1:1	✔	✔	
jdiff [AOH04]	ECFG	CFG node,	1:1	✔		✔
BMAT [WPM00]	Binary code	Code block	1:1, n:1	✔	✔	✔
Clone detectors	Various	Various	n:n			✔
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✔		✔
S. Kim et al.	Hybrid	Procedure	1:1	✔		✔

Comparison

Matching Technique	Program Representation	Granularity	Multiplicity (O:N)	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✔		
diff [HS77]	String	Line	1:1			✔
bdiff [Tic84]	String	Line	1:n			✔
cdiff [Yang91]	AST	AST node	1:1			✔
Neamtiu et al.	AST	Type, Variable	1:1	✔	✔	
jdiff [AOH04]	CFG	CFG node	1:1	✔		✔
BMAT [WPM00]	Binary code	Code block	1:1, n:1	✔	✔	✔
Clone detectors	Various	Various	n:n			✔
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✔		✔
S. Kim et al.	Hybrid	Procedure	1:1	✔		✔

Look at the "Granularity" column

Matching Technique	Program Representation	Granularity	Multiplicity	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✓		
diff [HS77]	String	Line	1:1			✓
bdiff [Tic84]	String	Line	1:n			✓
cdiff [Yang91]	AST	AST node	1:1			✓
Neamtiu et al.	AST	Type, Variable	1:1	✓		
jdiff [AOH04]	CFG	CFG node	1:1	✓		✓
BMAT [WPM00]	Binary code	Code block	1:1, n:1	✓	✓	✓
Clone detectors	Various	Various	n:n			✓
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✓		✓
S. Kim et al.	Hybrid	Procedure	1:1			✓

Any observations?

- Most techniques support code matching at a fixed granularity
-

Look at the "Granularity" column

Matching Technique	Program Representation	Granularity	Multiplicity	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✓		
diff [HS77]	String	Line	1:1			✓
bdiff [Tic84]	String	Line	1:n			✓
cdiff [Yang91]	AST	AST node	1:1			✓
Neamtiu et al.	AST	Type, Variable	1:1	✓		
jdiff [AOH04]	CFG	CFG node	1:1	✓		✓
BMAT [WPM00]	Binary code	Code block	1:1, n:1	✓	✓	✓
Clone detectors	Various	Various	n:n			✓
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✓		✓
S. Kim et al.	Hybrid	Procedure	1:1			✓

Any observations?

- Many techniques produce mappings at a fixed granularity
- Many fine-grained techniques require mappings at a higher level
- Many techniques will produce a long list of mappings, making it difficult to comprehend

Look at the "Multiplicity" column

Matching Technique	Program Representation	Granularity	Multiplicity	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✓		
diff [HS77]	String	Line	1:1			✓
bdiff [Tic84]	String	Line	1:n			✓
cdiff [Yang91]	AST	AST node	1:1			✓
Neamtiu et al.	AST	Type, Variable	1:1	✓		
jdiff [AOH04]	CFG	CFG node	1:1	✓		✓
BMAT [WPM00]	Binary code	Code block	1:1, n:1	✓	✓	✓
Clone detectors	Various	Various	n:n			✓
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✓		✓
S. Kim et al.	Hybrid	Procedure	1:1	✓		✓

Any observations?

- copy and paste
- merging and splitting
-

Look at the "Heuristics" column

Matching Technique	Program Representation	Granularity	Multiplicity	Heuristics		
				Name	Posi-tion	Similarity
name matching	Entity	Procedure/ File	1:1	✓		
diff [HS77]	String	Line	1:1			✓
bdiff [Tic84]	String	Line	1:n			✓
cdiff [Yang91]	AST	AST node	1:1			✓
Neamtiu et al.	AST	Type, Variable	1:1	✓		
idiff [AOH04]	CFG	CFG node	1:1	✓		✓
BMAT [WPM001]	Binary code	Code block	1:1, n:1	✓	✓	✓
Clone detectors	Various	Various	n:n			✓
Zou, Godfrey	Hybrid	Procedure	1:1, n:1, 1:n	✓		✓
S. Kim et al.	Hybrid	Procedure	1:1	✓		✓

Any observations?

- Many use name-based matching

Evaluation

- We created a set of hypothetical program change scenarios.
  - small change scenario
    - changes in the nested level of a control structure
    - semantics-preserving statement reordering
  - large change scenario
    - procedure level renaming and splitting
  - renaming, splitting, and merging scenarios at various granularities

Small Change Scenario

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 }

Large Change Scenario

- A file PElmtMatch changed its name to PMatching.
- A procedure matchBlck are split into two procedures matchDBlck and matchCBlck.
- A procedure matchAST changed its name to matchAbstractSyntaxTree.

Evaluation based on Hypothetical Change Scenarios

Matching Technique	Scenario		Transformation				Weaknesses
			Split/Merge		Rename		
	Small	Large	Proc	File	Proc	File	
diff	○	○					
bdiff	●	◒					
cdiff	○	○					
Neamtiu et al.	○	○	○	○	○	○	✘ partial AST matching
jdiff	●	○					
BMAT	○	●	○	○	●	●	✘ 1:1 mapping only ✘ only applicable to binary code
Zou, Godfrey	○	●	●	●	●	●	✘ semi-automatic analysis
S. Kim et al.	○	●	○	○	●	●	✘ 1:1 mapping only

● good, ◒ mediocre, ○ poor

Evaluation based on Hypothetical Change Scenarios

Matching Technique	Scenario		Transformation				Weaknesses
			Split/Merge		Rename		
	Small	Large	Proc	File	Proc	File	
diff	◒	○	○	○	○	○	✘ require file level mapping
bdiff	●	○	◒				file level mapping
cdiff	○	○	○				procedure level mapping ve to nested level change
Neamtiu et al.	○	○	○	○	○	○	✘ partial AST matching
jdiff	●	◒	○	○	◒	◒	✘ sensitive control structure change
BMAT	○	●	○	○	●	●	✘ 1:1 mapping only ✘ only applicable to binary code
Zou, Godfrey	○	●	●	●	●	●	✘ semi-automatic analysis
S. Kim et al.	○	●	○	○	●	●	✘ 1:1 mapping only

Fine-grained matching techniques do not work well in case of large changes.

● good, ◒ mediocre, ○ poor

Evaluation based on Hypothetical Change Scenarios

Matching Technique	Scenario		Transformation				Weaknesses
			Split/Merge		Rename		
	Small	Large	Proc	File	Proc	File	
diff	◒	○	○	○	◒	○	✘ require file level mapping
bdiff	●	○	◒	○	◒		
cdiff	○	○	○	○	○		
Neamtiu et al.	○	○	○	○	○		
jdiff	●	◒	○	○	◒	◒	✘ sensitive control structure change
BMAT	○	●	○	○	●	●	✘ 1:1 mapping only ✘ only applicable to binary code
Zou, Godfrey	○	●	●	●	●	●	✘ semi-automatic analysis
S. Kim et al.	○	●	○	○	●	●	✘ 1:1 mapping only

Due to 1:1 mapping assumptions, they perform poorly when splitting or merging.

● good, ◒ mediocre, ○ poor

Evaluation based on Hypothetical Change Scenarios

Matching Technique	Scenario		Transformation				Weaknesses
			Split/Merge		Rename		
	Small	Large	Proc	File	Proc	File	
diff	◒	○	○	○	◒	○	✘ require file level mapping
bdiff	●	○	◒	○	◒		
cdiff	○	○	○	○	○		
Neamtiu et al.	○	○	○	○	○		
jdiff	●	◒	○	○	◒	◒	✘ sensitive control structure change
BMAT	○	●	○	○	●	●	✘ 1:1 mapping only ✘ only applicable to binary code
Zou, Godfrey	○	●	●	●	●	●	✘ semi-automatic analysis
S. Kim et al.	○	●	○	○	●	●	✘ 1:1 mapping only

Zou and Godfrey's origin analysis will work well but is semi-automatic.

● good, ◒ mediocre, ○ poor

What are future research directions for program differencing tools?

- Hybrid
  - Voting?
- Improve representation
- Improve heuristics
- Design Benchmark
- Dynamic information

What are future research directions for program differencing tools?

- Hybrid matcher
  - Find consensus among many matching techniques?
- Hierarchical matching
- Iterative fix-point matching algorithm
- Leveraging dynamic information
- Concise representation / Aggregation differencing results
- Identification of exceptions

Logical Structural Diff

Kim and Notkin, to be presented in ICSE 2009

Motivating Scenarios

- "This program worked a month ago but is not working now. What changed since then? Which change led to a bug?"
- "Did Bob implement the intended changes correctly?"
- "There's a merge conflict. What did Alice change?"

Diff Output

Changed Code		
File Name	Status	Lines
DummyRegistry	New	20 lines
AbsRegistry	New	133 lines
JRMPRegistry	Modified	123 lines
JeremieRegistry	Modified	52 lines
JacORBCosNaming	Modified	133 lines
IIOPCosNaming	Modified	50 lines
CmiRegistry	Modified	39 lines
NameService	Modified	197 lines
NameServiceManager	Modified	15 lines
Total Change: 9 files, 723 lines		

```
- public class CmiRegistry implements
NameService {
+ public class CmiRegistry extends
AbsRegistry implements NameService {
-     private int port = ...
-     private String host = null
-     public void setPort (int p) {
-        if (TraceCarol. isDebug()) { ...
-        }
-      }
-      public int getPort() {
-        return port;
-       }
-      public void setHost(String host)
{ ....
```

Check-In Comment

"Common methods go in an abstract class. Easier to extend/maintain/fix"

Changed Code		
File Name	Status	Lines
`DummyRegistry`	New	20 lines
`AbsRegistry`	New	133 lines
`JRMPRegistry`	Modified	123 lines
`JeremieRegistry`	Modified	52 lines
`JacORBCosNaming`	Modified	133 lines
`IIOPCosNaming`	Modified	50 lines
`CmiRegistry`	Modified	39 lines
`NameService`	Modified	197 lines
`NameServiceManager`	Modified	15 lines
Total Change: 9 files, 723 lines		

which methods are the common methods?
Did this change happen for all subclasses?
Did this person create a new class?
Is it really easier to maintain?
Did this document other artifacts consistently?

What kinds of questions that programmers ask during change tasks?

LSdiff results

- Fact 1. AbsRegistry is a new class
- Rule 1. All host fields in NameSvc's subtypes were deleted except LmiRegisty class
- Rule 2. All setHost methods in NameSvc's subtypes were deleted except LmiRegistry class
- Rule 3. All getHost methods in NameSvc's subtypes deleted calls to SQL.exec except LmiRegistry class.

LSdiff's goal

- LSdiff concisely infers systematic changes and reports exceptions that deviate from these systematic changes.
- To complement existing uses of diff
- To detect potential bugs by finding potential inconsistent changes
- To help programmers reason about related changes
- To allow for top down reasoning as opposed to reading changes file-by-file

Systematic Changes

- Refactoring [Opdyke 92, Griswold 92, Fowler 99...]

"Move related classes from one package to another package"

Systematic Changes

- Refactoring [Opdyke 92, Griswold 92, Fowler 99...]
- API update [Chow&Notkin 96, Henkel&Diwan 05, Dig&Johnson 05...]

"Update an API and all call sites of the API"

Systematic Changes

- Refactoring [Opdyke 92, Griswold 92, Fowler 99...]
- API update [Chow&Notkin 96, Henkel&Diwan 05, Dig&Johnson 05...]
- Crosscutting concerns [Kiczales et. al. 97, Tarr et. al. 99, Griswold 01...]

"Adding logging feature throughout code"

Systematic Changes

- Refactoring [Opdyke 92, Griswold 92, Fowler 99...]
- API update [Chow&Notkin 96, Henkel&Diwan 05, Dig&Johnson 05...]
- Crosscutting concerns [Kiczales et. al. 97, Tarr et. al. 99, Griswold 01...]
- Consistent updates on code clones [Miller&Myers 02, Toomim et. al. 04, Kim et. al. 05]

"Apply similar changes to syntactically similar code fragments"

Logical Structural Diff Algorithm

Output: ***logic rules*** and ***facts*** that describe changes to ***code elements and structural dependencies***

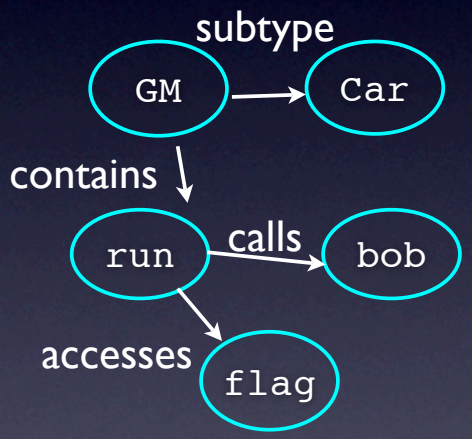
1. Extract a set of facts from a program using JQuery [Jensen & DeVolder 03]
2. Compute fact-level differences
3. Learn Datalog rules using an inductive logic programming algorithm

Detail Steps

Step 1. Extract Facts

```
class GM extends Car

   void run(int c) {

      if (Util.flag)...

      bob();}

}
```

Program




```
type("GM")

subtype("Car","GM")

method("GM.run","run","GM")

accesses("Util.flag","GM.run")

calls("GM.run","GM.bob")
```

Fact-base

A fact-base program representation approach has been used by many tools such as JQuery [Jensen&DeVolder 03], CodeQuest [Hajiev et. al. 06], Grok [Holt et. al.] , etc.

Step 2. Compute Fact-Level Differences

Old Program (FBo)
past_*

```
subtype
("Svc","X")
subtype
("Svc","Y")
subtype
("Svc","Z")
subtype
("Svc","NameSvc
")
```

-

New Program (FBn)
current_*

```
subtype
("Svc","X")
...

method
("exec","X")
method
("exec","Y")
...
```

=

Differences (ΔFB)
added_* / deleted_*

```
added_method
("exec","X")
added_method
("exec","Y")
added_method
("exec","Z")
```

set difference

ΔFB Example

FB_o (a fact-base of P_o)	FB_n (a fact-base of P_n)	ΔFB
subtype("Car","BMW"), ... method("BMW.start", "start", BMW) ...	subtype("Car","BMW"), ... method("BMW.start", "start", BMW) calls("BMW.start", "Key.chk")	+calls("BMW.start", "Key.chk")
subtype("Car","GM"), ... method("GM.start", "start", "GM") accesses("Key.on", "GM.start") ...	subtype("Car","GM"), ... method("GM.start", "start", "GM") calls("GM.start", "Key.chk") ...	-accesses("Key.on", "GM.start") +calls("GM.start", "Key.chk")
subtype("Car","Kia"), ... method("Kia,start", "start", "Kia") accesses("Key.on", "Kia.start"), ...	subtype("Car","Kia"), ... method("Kia,start", "start", "Kia") ...	-accesses("Key.on", "Kia.start")
type("Bus") method("Bus,start", "start", Bus) accesses("Key.on", "Bus.start")	type("Bus") method("Bus,start", "start", Bus) calls("Bus.start", "log")	-accesses("Key.on", "Bus.start") +calls("Bus.start", "log")
type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	

Limitation 1: Verbosity

FB_o (a fact-base of P_o)	FB_n (a fact-base of P_n)	ΔFB
subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) . . .	subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) calls("BMW.start", "Key.chk")	+calls("BMW.start", "Key.chk")
subtype("Car","GM"), . . . method("GM.start", "start", "GM") accesses("Key.on", "GM.start") . . .	subtype("Car","GM"), . . . method("GM.start", "start", "GM") calls("GM.start", "Key.chk")	-accesses("Key.on", "GM.start") +calls("GM.start", "Key.chk")
subtype("Car","Kia"), . method("Kia,start", "st accesses("Key.on", "Kia		-accesses("Key.on", "Kia.start")
type("Bus") method("Bus,start", "start", Bus) accesses("Key.on", "Bus.start")	type("Bus") method("Bus,start", "start", Bus) calls("Bus.start", "log")	-accesses("Key.on", "Bus.start") +calls("Bus.start", "log")
type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	

remove all accesses to `Key.on`

Limitation 2: Lack of Contextual Information

FB_o (a fact-base of P_o)	FB_n (a fact-base of P_n)	ΔFB
subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) . . .	subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) calls("BMW.start", "Key.chk")	+calls("BMW.start", "Key.chk")
subtype("Car","GM"), . . . method("GM.start", "start", "GM") accesses("Key.on", "GM.start") . . .		-accesses("Key.on", "GM.start") +calls("GM.start", "Key.chk")
subtype("Car","Kia"), . . . method("Kia,start", "start", "Kia") accesses("Key.on", "Kia.start"), . . .		-accesses("Key.on", "Kia.start")
type("Bus") method("Bus,start", "start", Bus) accesses("Key.on", "Bus.start")	calls("Bus.start", "log")	-accesses("Key.on", "Bus.start") +calls("Bus.start", "log")
type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	

invoke `Key.chk` from the `start` methods in `Car's` subtypes.

Limitation 2: Lack of Contextual Information

FB_o (a fact-base of P_o)	FB_n (a fact-base of P_n)	ΔFB
subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) . . .	subtype("Car","BMW"), . . . method("BMW.start", "start", BMW) calls("BMW.start", "Key.chk")	+calls("BMW.start", "Key.chk")
subtype("Car","GM"), . . . method("GM.start", "start", "GM") accesses("Key.on", "GM.start") . . .		-accesses("Key.on", "GM.start") +calls("GM.start", "Key.chk")
subtype("Car","Kia"), . . . method("Kia,start", "start", "Kia") accesses("Key.on", "Kia.start"), . . .		-accesses("Key.on", "Kia.start")
type("Bus") method("Bus,start", "start", Bus) accesses("Key.on", "Bus.start")	calls("Bus.start", "log")	-accesses("Key.on", "Bus.start") +calls("Bus.start", "log")
type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	type ("Key") field("Key.on", "on", "Key") method ("Key.chk", "chk", "Key")	

invoke `Key.chk` from the `start` methods in `Car's` subtypes.

Step 3. Learn Rules

- Our rule learner uses a ***bounded depth search*** algorithm that finds ***Datalog rules*** in a domain specific form.
- We have input parameters that determine the ***validity*** of a rule.
  - *a*: accuracy
  - *m*: min support
  - *k*: the length of antecedant

```
Example.
past_calls (x, “foo”)
  => deleted_calls(x, “foo”)
(8/10) a = 0.80, m=8, k=1.
```

Step 3. Learn Rules

Input: FB_o, FB_n, ΔFB, m, a, k, and β
Output: L and U
/* Initialize R, a set of ungrounded rules; L, a set of learned rules; and U, a set of facts in ΔFB that are not covered by L. */
R := ∅, L := ∅, U := ΔFB;
U := applyDefaultWinnowingRules (ΔFB, FB_o, FB_n); /* reduce ΔFB with default winnowing rules. */
R := createInitialRules (m); /* create rules with an empty antecedent by enumerating all possible consequents. */
foreach $i = 1 \ldots k$ **do**
  R := extendUngroundedRules (R) ; /* extend all ungrounded rules in R by adding all possible literals to their antecedent. */
  foreach $r \in$ R **do**
    G := createPartiallyGroundedRules (r) ;
    /* try all possible constant substitutions for r's variable. */
    foreach g *in* G **do**
      if isValid *(g)* **then**
        L :=L ∪ {g};
        U :=U − {g.matches};
      end
    end
  end
  R :=selectRules (R, β) ; /* select the best β rules in R */
end

Back

Recap

- Many differencing techniques individually compare code elements at particular granularities using similarity measures.
  - Hard to comprehend as a long list of matches
  - Difficult to identify exceptions that violate systematic patterns
- LSdiff uses rule-based change representations to explicitly capture systematic changes and automatically infers these rules.

Preview for This 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?

Announcement

- Project Checkpoint Due on this thursday.
  - I won't grade them.
  - It is not mandatory.
  - You are encouraged to submit to seek my feedback.
  - Available for both research project, literature survey, and tool evaluation