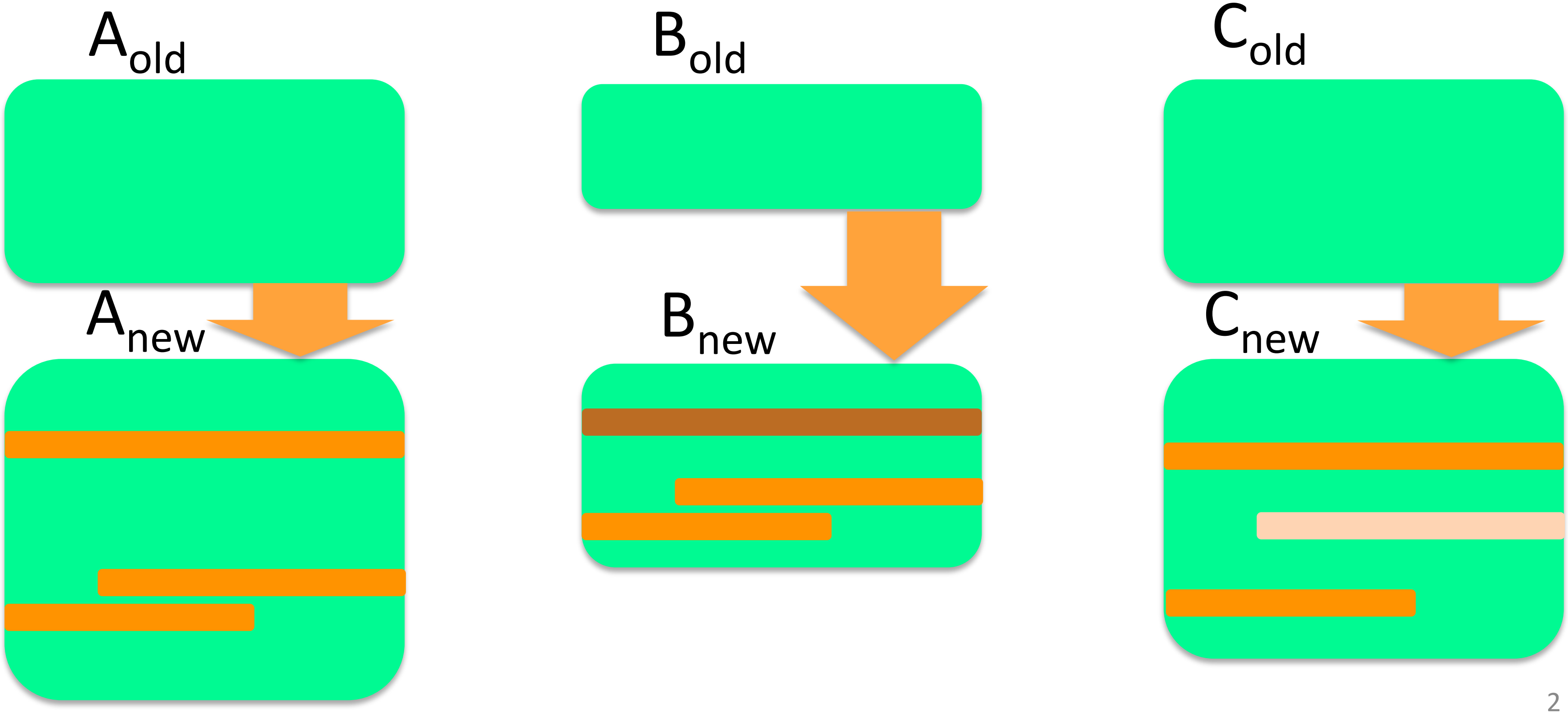
Lase: Locating and Applying Systematic Edits by Learning from Examples

Na Meng Miryung Kim Kathryn S. McKinley
The University of Texas at Austin

Motivating Scenario

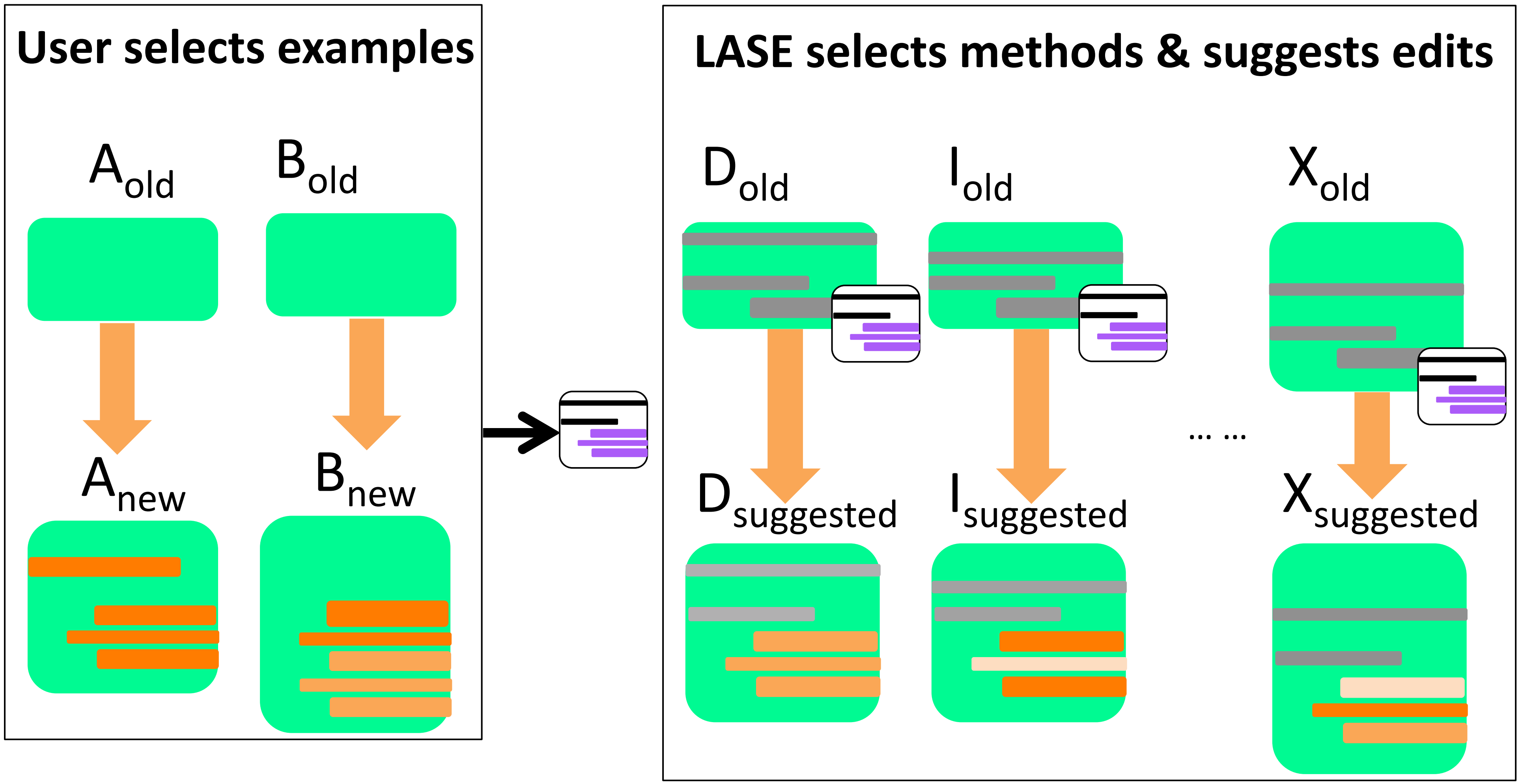
Pat needs to update database transaction code to prevent SQL injection attacks



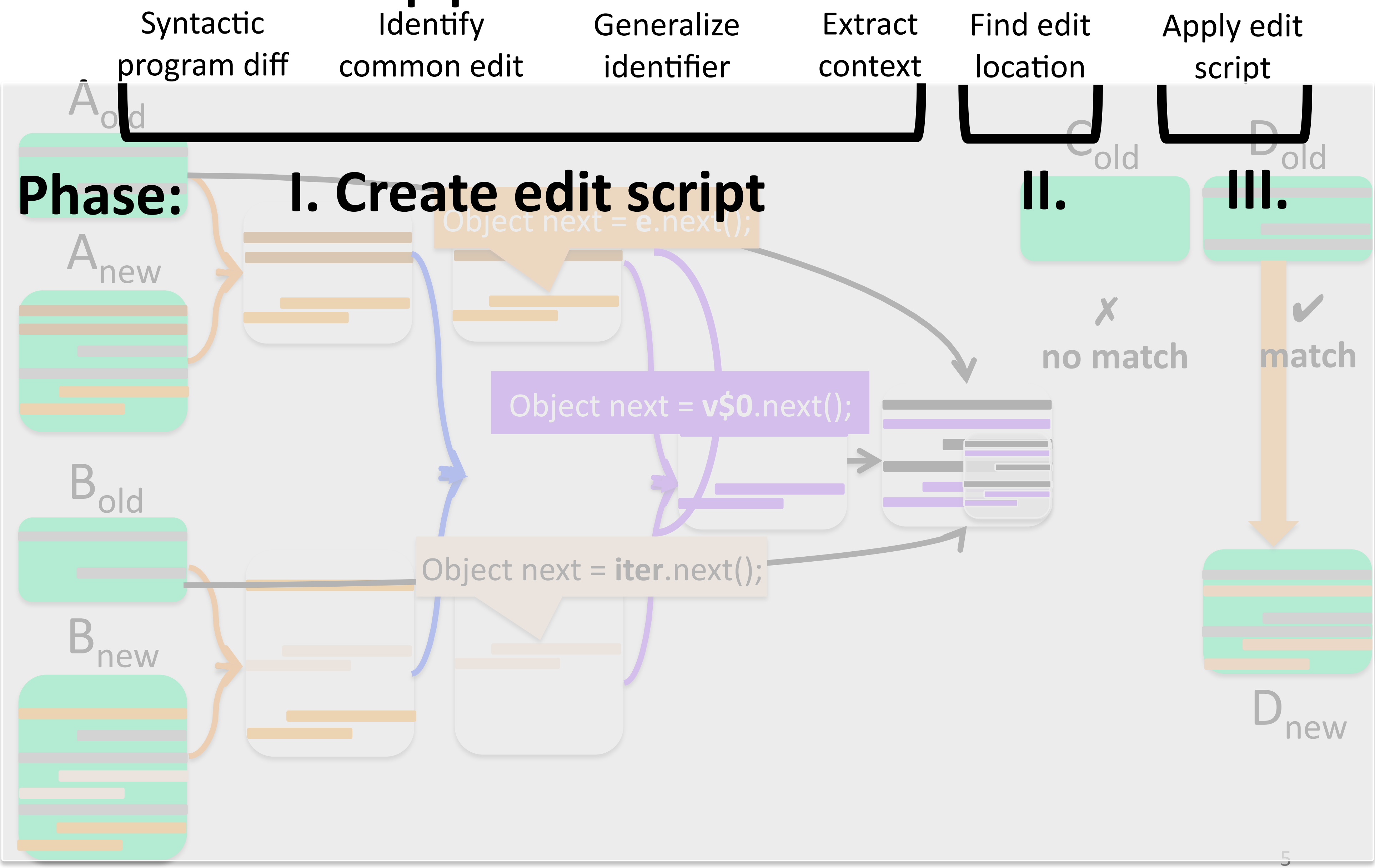
Systematic Editing

- ***Similar but not identical changes*** to multiple contexts
- Manual, tedious, and error-prone
- Source transformation tools require describing edits in a formal language
- Bug fixing tools locate and apply simple or limited stylized code changes
  - Coccinelle, CFix, FixMeUp
- Sydit applies an edit inferred from a code example to user-selected targets

Workflow of Lase



Approach Overview



Step 1. Syntactic Program Diff

Input: m_{old}, m_{new}

Output: Edit operations

operation	**definition**
insert(**u, v, k**)	insert node *u* and position it as the (k+1)th child of node *v*
delete(**u**)	delete node *u*
update(**u, v**)	replace *u* with *v*
move(**u, v, k**)	delete *u* from its current position and insert *u* as the (k+1)th child of *v*

Step 2: Identify Common Edit

- Longest Common Edit Operation Subsequence

Edit script A

insert(Object next = e.next()…)
insert(if(next instanceof MVAction)
insert(((MVAction)next).update())
update(print(next.toString())) to …

Edit script B

insert(Object next = iter.next()…)
update(print(next.getString())) to …
insert(if(next instanceof MVAction)
insert(((MVAction)next).update())
delete(System.out.println(…))

insert(Object next = e.next()…)
insert(if(next instanceof MVAction))
insert(((MVAction)next).update())

insert(Object next = iter.next()…)
insert(if(next instanceof MVAction))
insert(((MVAction)next).update())

Step 3: Generalize Identifier

- Keep the original identifiers if examples agree
- Abstract identifiers if examples disagree

Object next = **e**.next();

Object next = **iter**.next();

Object next = **v$0**.next();

	Generalized identifier	**Identifier in mA**	**Identifier in mB**
Variable Map	next	next	next
	v$0	e	iter
Method Map	next	next	next
Type Map	Object	Object	Object
	Iterator	Iterator	Iterator

Step 4: Extract Context

```
Iterator e = fActions.values().iterator();
… …
while(e.hasNext())
    Object next = e.next();
```

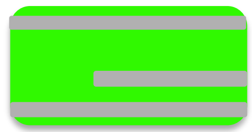
```
Iterator iter = getActions().values().iterator();
… …
while(iter.hasNext())
    Object next = iter.next();
```

```
Iterator v$0 = u$0:FieldAccessOrMethodInvocation.values().iterator();
… …
while(v$0.hasNext())
```

	Generalized identifier	Identifier in mA	Identifier in mB
Uncertain Map	u$0:FieldAccessOrMethodInvocation	fActions	getActions()
Variable Map	v$0	e	iter
Method Map	values	values	values
	iterator	iterator	iterator
	hasNext	hasNext	hasNext
TypeMap	Iterator	Iterator	Iterator

Phase II. Find Edit Locations

D_{old}



```
Iterator e = fActions.values().iterator();
```

```
Iterator v$0 = u$0:FieldAccessOrMethodInvocation.values().iterator();
```

	Generalized identifier	Identifier in mD
Uncertain Map	u$0:FieldAccessOrMethodInvocation	fActions
Variable Map	v$0	e
Method Map	values	values
	iterator	iterator
TypeMap	Iterator	Iterator

Phase III. Applying Edit Script

- Customize general edit scripts
  - Identifier concretization
  - Edit position concretization
- Apply the customized edit scripts

```
Comment[] getLeadingComments(ASTNode node){
-   if (this.leadingComments != null) {
+   if (this.leadingPts >= 0) {
-       int[] range = (int[]) this.leadingComments.get(node);
+       int[] range = null;
+       for (int i = 0; range == null && i <= this.leadingPtr; i++) {
+           if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
+       }
        if (range != null) {
            int length = range[1] - range[0] + 1;
            Comment[] leadComments = new Comment[length];
            System.arraycopy(this.comments, range[0], leadComments, 0, length);
             return leadComments;
        }
    }
}
```

```
Comment[] getTrailingComments(ASTNode node){
-   if (this.trailingComments != null) {
+   if (this.trailingPts >= 0) {
-       int[] range = (int[]) this.trailingComments.get(node);
+       int[] range = null;
+       for (int i = 0; range == null && i <= this.trailingPtr; i++) {
+           if (this.trailingNodes[i] == node) range = this.trailingIndexes[i];
+       }
        if (range != null) {
            int length = range[1] - range[0] + 1;
            Comment[] trailComments = new Comment[length];
            System.arraycopy(this.comments, range[0], trailComments, 0,
length);
             return trailComments;
        }
    }
}
```

```
public int getExtendedEnd (ASTNode node){
    int end = node.getStartPosition() + node.getLength();
-   if (this.v$_1_ != null) {
+   if (this.trailingPts >= 0) {
-       int[] range = (int[]) this.trailingComments.get(node);
+       int[] range = null;
+       for (int i = 0; range == null && i <= this.v$_1_; i++) {
+           if (this.v$_2_[i] == node) range = this.v$_3_[i];
+       }
        if (range[0] == -1 && range[1] == -1) {
            … …
        } else {
            … …
        }
    }
    return end - 1;
}
```

Outline

- Phase I: Creating Abstract Edit Scripts
  - Syntactic Program Diff
  - Identify Common Edit
  - Generalize Identifier
  - Extract Context
- Phase II: Find Edit Locations
- Phase III: Apply Edit Script
- Evaluation

Test Suite

- 24 repetitive bug fixes that require multiple check-ins [Park et al., MSR 2012]
  - 2 from Eclipse JDT and 22 from Eclipse SWT
  - Each bug is fixed in multiple commits
  - Clones of at least two lines between patches checked in at different times
- 37 systematic edits that require similar changes to different methods

RQ1: Precision, Recall, and Accuracy

Precision (P): What percentage of all *found* locations are correctly identified?

Recall (R): What percentage of all *expected* locations are correctly identified?

Accuracy (A): How similar is Lase-generated version to developer-generated version?

Index	Bug(patches)	m_i	Edit Location					Operations		
			Σ	✔	P%	R%	A%	E	C	$E_{A\%}$
2	82429(2)	16	13	12	92	75	81	9	9	100
4	139329(3)	6	2	2	100	33	74	6	3	50
7	103863(5)	7	7	7	100	100	100	34	34	100
8	129314(3)	3	4	4	100	100	100	2	2	100
16	95409(3)	7	9	9	100	100	78	4	4	100
24	98198(2)	9	15	15	100	100	95	3	3	100

On average, Lase finds edit locations with 99% precision, 89% recall, and 91% accuracy.

For three bugs, Lase suggests in total 9 edits that developers missed and later confirmed.

RQ2: Sensitivity to number of exemplar edits

- 7 cases in the oracle data set
- Enumerate subsets of exemplar edits

	# of exemplars	P%	R%	A%
Index 4	1	100	17	100
	2	100	51	72
	3	100	82	67
	4	100	96	67
	5	100	100	67
Index 7	1	100	59	100
	2	100	83	100
	3	100	84	100
	4	100	88	100
	5	100	92	100
	6	100	96	100
Index 12	1	100	54	92
	2	78	90	85
	3	49	98	83
	4	31	100	82

As the number of exemplar edits increases,

- P does not change except for case 12

- R is more sensitive to the number of exemplar edits
- R increases as a function of exemplar edits

- A decreases when exemplar edits are different
- A remains the same or may increase when the exemplar edits are very similar

Conclusion

- Lase automates edit location search and program transformation application
- Lase achieves
  99% precision, 89% recall, and 91% accuracy

- Future Work
  – Integrate with ***automated compilation and testing***
  – Automatically detect repetitive change examples to infer program transformations

Thank You !

Questions?

References I

- [Meng et al. 2011] Na Meng, Miryung Kim and Kathryn S. McKinley. Systematic editing: Generating program transformations from an example. In PLDI ‘11.
- [Kamiya et al. 2002] Toshihiro Kamiya and Shinji Kusumoto and Katsuro Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. In TSE ’02.
- [Lozano et al. 2004] Antoni Lozano and Gabriel Valiente. On the maximum common embedded subtree problem for ordered trees. In C. Iliopoulos and T Lecroq, editors, String Algorithmics, 2004.
- [Park et al. MSR 2012] J. Park, M. Kim, B. Ray, and D.-H. Bae. An empirical study of supplementary bug fixes. In MSR ’12.

References II

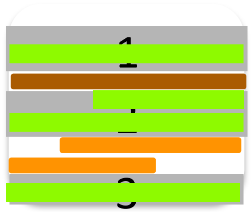
- [Nguyen et al.] H. A. Nguyen, T. T. Nguyen, G. W. Jr., A. T. Nguyen, M. Kim, and T. Nguyen. A graph-based approach to api usage adaptation. In OOPSLA ’10.
- [Cordy et al.] J. R. Cordy, C. D. Halpern, and E. Promislow. Txl: A rapid prototyping system for programming language dialects. Computer Languages, 1991.
- [Gulwani et al.] S. Gulwani. Dimensions in program synthesis. In PPDP ’10.
- [Weimer et al.] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In ICSE ’09.

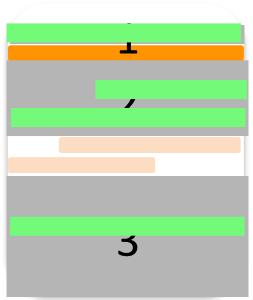
Step 4: Common Edit Context Extraction

- Extract all potential common context
- Refine the common context
  - Consistent identifier mapping
  - Embedded subtree isomorphism
  - Program dependence equivalence

Step 4: Common Edit Context Extraction (1/4)

- Finding common *text* with clone detection (CCFinder [Kamiya et al. 2002])





Step 4: Common Edit Context Extraction (2/4)

- Identifier generalization

```
Iterator e = fActions.values().iterator();
while (e.hasNext()) {
```

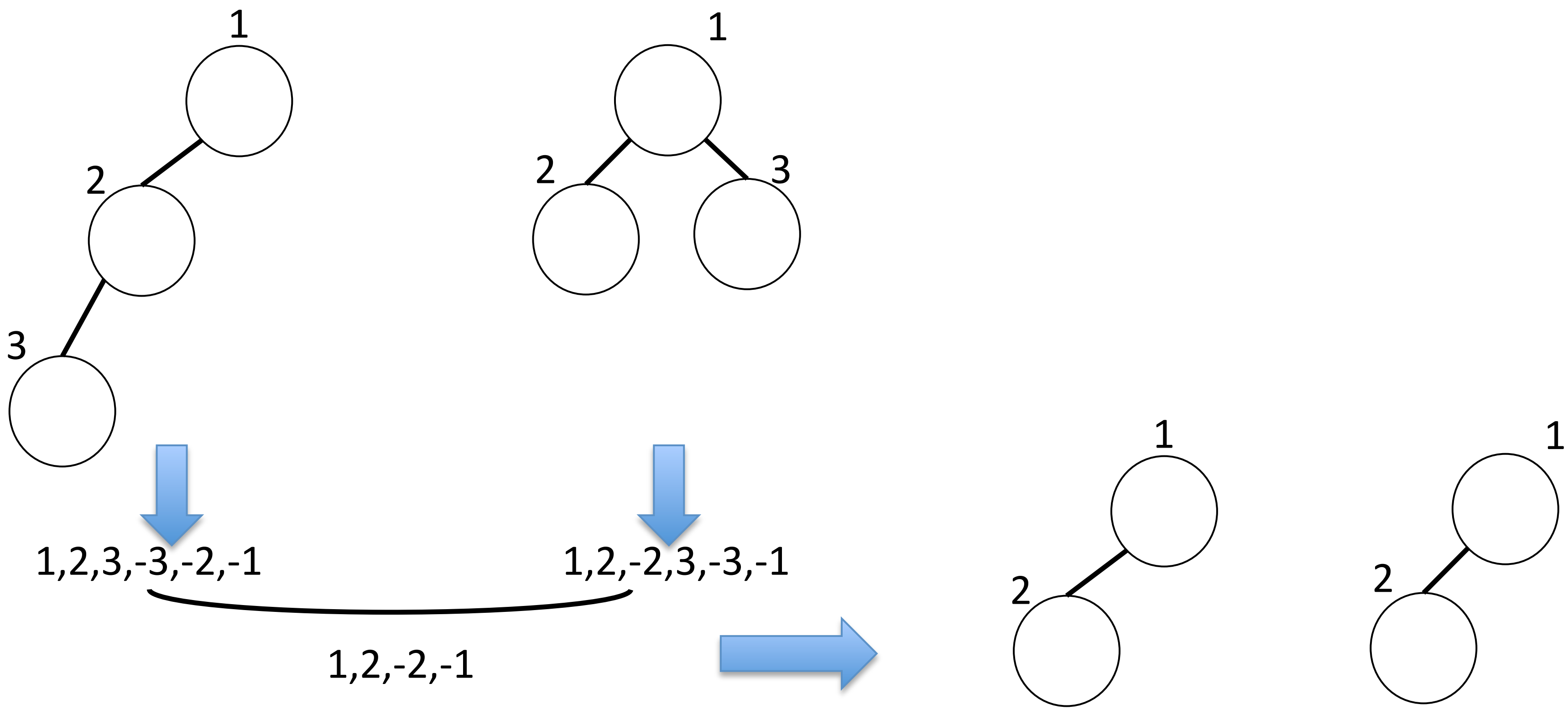
```
Iterator iter = getActions().values().iterator();
while (iter.hasNext()) {
```

```
Iterator v$0 = u$0:FieldAccessOrMethodInvocation.values().iterator();
while (v$0.hasNext()) {
```

	Abstract identifier	Identifier in mA	Identifier in mB
Uncertain Map	u$0:FieldAccessOrMethodInvocation	fActions	getActions()
Variable Map	v$0	e	iter
Method Map	values	values	values
	iterator	iterator	iterator
	hasNext	hasNext	hasNext
TypeMap	Iterator	Iterator	Iterator

Step 4: Common Edit Context Extraction (3/4)

- Maximum Common Embedded Subtree Extraction (MCESE) [Lozano et al. 2004]



Step 4: Common Edit Context Extraction (4/4)

ence analysis

Iterator e = fActions.values().iterator();

while (**e**.hasNext()) {

Object next = e.next();

	Abstract identifier	Identifier in mA	Identifier in mB
Variable Map	v$0	e	iter
Method Map	values	values	values
… ….			

?When more than two examples?