INTELLIGENT TUTORING

CS3213 FSE

Prof. Abhik Roychoudhury

National University of Singapore

WHAT WE DID EARLIER

Requirements and Modeling

- System Requirements: Use-cases, Scenarios, Sequence Diagrams
- System structure: Class diagrams
- Discussion on semantics
- System behavior: State diagrams

- Today
  - **Discussion on the thinking behind your course project**

INTELLIGENT TUTORING

Prof. Abhik Roychoudhury

National University of Singapore

ONLINE TEACHING

Lack of personalized feedback?

GOALS OF INTELLIGENT TUTOR

Solution Generation

- Generate complete solution of a given problem. Useful for
  - Completing student's incorrect attempt
  - Generate partial hints to guide towards next step
  - Possible automated grading.

Similar Problem Generation

- Given a problem, search for other problems having similar solution
- Useful for generating example problems

Parameterized Problem Generation

- Create new problems satisfying given solution characteristics.
- Useful for generating plagiarism free assignment problems

MOTIVATING EXAMPLE

Problem Statement: write a Python program to count the number of elements smaller than **x** in a sorted sequence **seq**.

```
def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
    return len(seq)
```

Reference Solution

Input	Output
search(2, [1,2,3])	1
search(3, [4,5,6])	0

Sample Test Cases

MOTIVATING EXAMPLE

Consider ***grading*** the following student program.

```python
def search(x, seq):
    if seq == () or seq == []:
        return 0
    elif x > seq[-1]:
        return len(seq)
    else:
        for num in range(len(seq)):
            if x > seq[num]:
                continue
            elif x < seq[num]:
                return num
    return 0
```

Deduct grades due to:

- Fail to pass all test cases.
  e.g., search(2, [1,2,3])
- Far different from the reference solution.

Understanding student programs is usually time-consuming.

Incorrect Student Program **Human TA**

MOTIVATING EXAMPLE

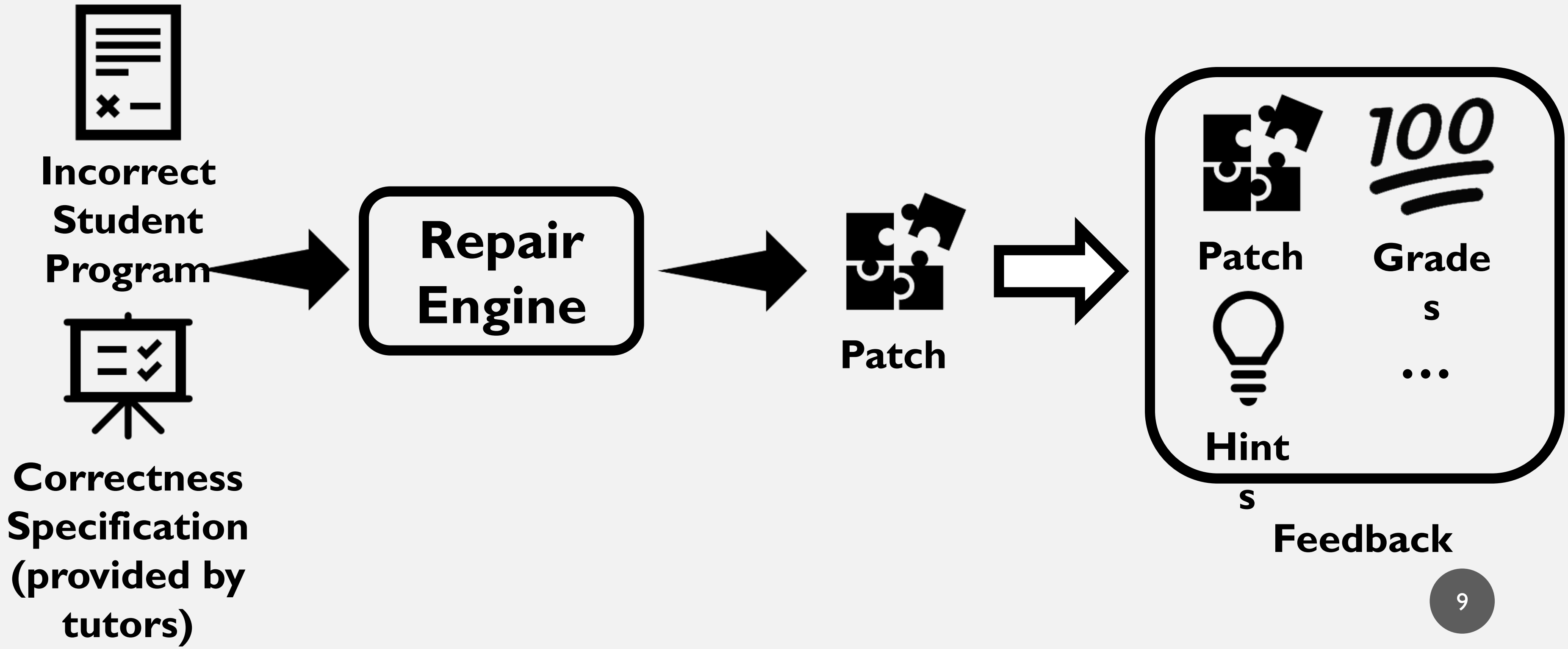
In fact, only one operator is wrong.

```
def search(x, seq):
    if seq == () or seq == []:
        return 0
    elif x > seq[-1]:
        return len(seq)
    else:
        for num in range(len(seq)):
            if x > seq[num]:
                continue
            elif x < seq[num]: # fix: <=
                return num
    return 0
```

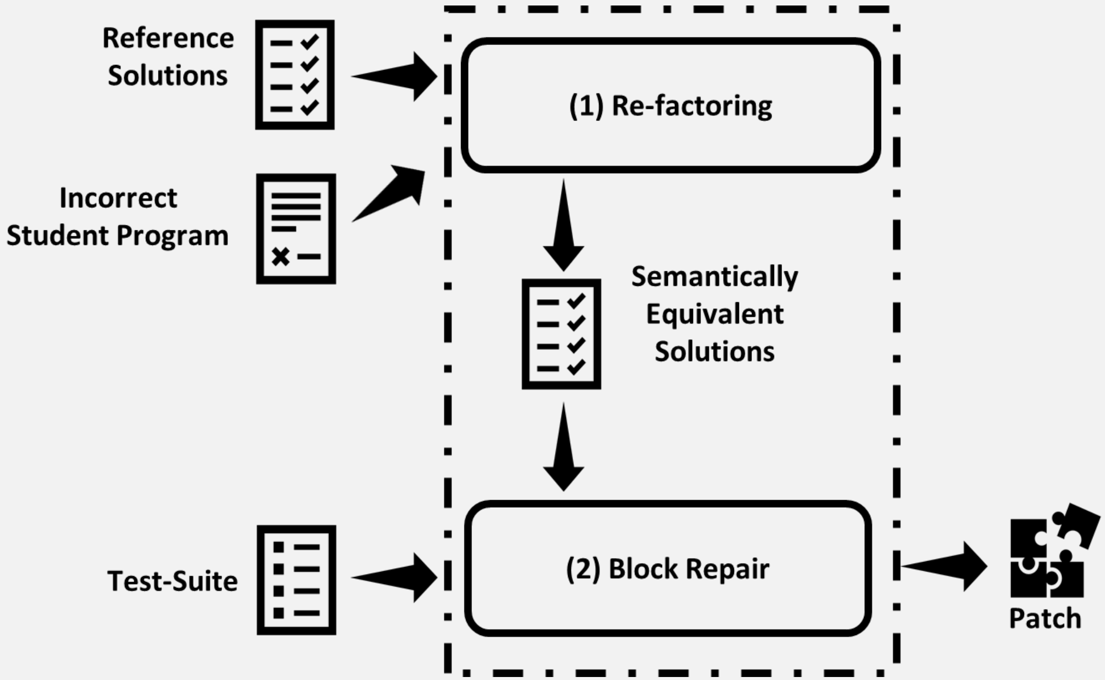
Incorrect Student Program

REPAIR-BASED FEEDBACK GENERATION

- Envision the feedback generation problem as an *Automated Program Repair* (*APR*) problem.



HINT GENERATION



RUNNING EXAMPLE

Problem Statement: Write a Python program which

* Given a sorted sequence **seq**
* Counts the number of elements smaller than **x**

Reference Solution	Incorrect Student Program

<pre>def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
 	
    return len(seq)</pre>
	<pre>def search(e, lst):
	    for j in range(len(lst)):
	        if e < lst[j]:
	            return j
	        else:
	            j = j + 1
	    return len(lst) + 1</pre>

STEP 1: REFACTORING

Refactored Correct Solution

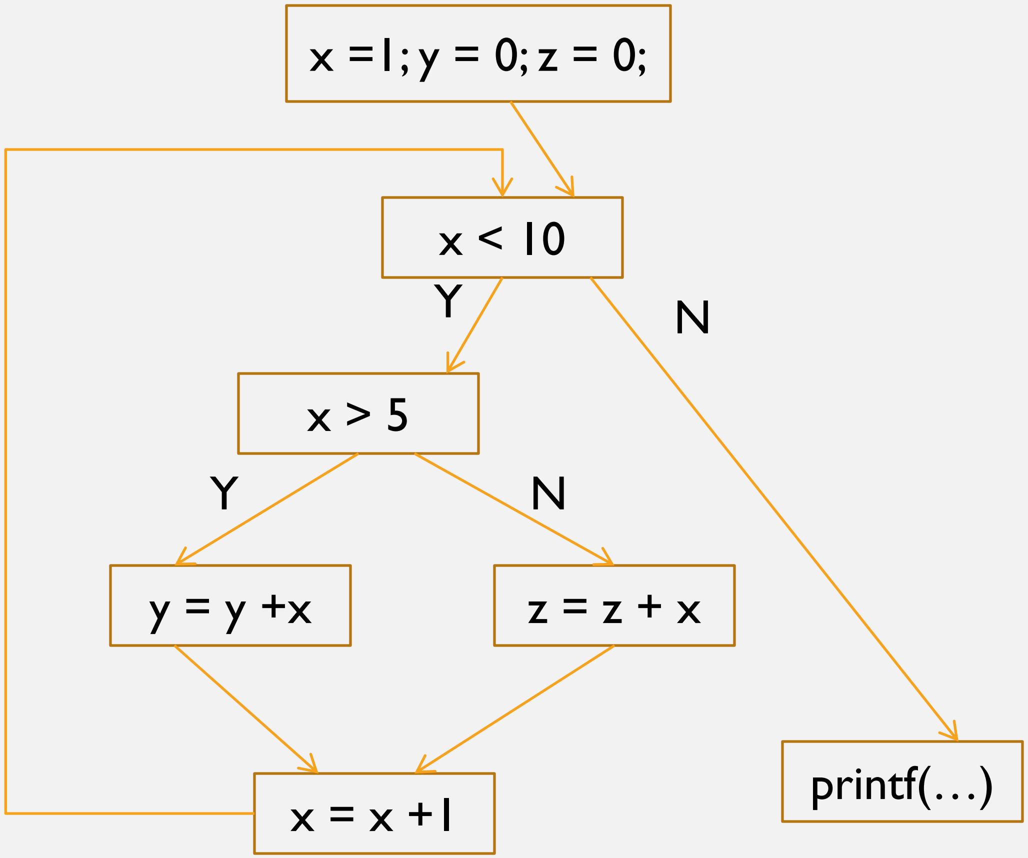
```python
def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
        else:
            pass
    return len(seq)
```

Incorrect Student Program

```python
def search(e, lst):
    for j in range(len(lst)):
        if e < lst[j]:
            return j
        else:
            j = j + 1
    return len(lst) + 1
```

CONTROL FLOW GRAPH

```
x = 1; y = 0; z = 0;
while (x < 10){
    if (x > 5)
            y = y + x;
    else  z = z + x;
    x = x + 1;
}
printf(…);
```



Nodes of the graph, basic blocks, are maximal code fragments executed without control transfer. The edges denote control transfer.

STEP 2: VARIABLE MAPPING

Refactored Correct Solution	Incorrect Student Program

`def search(x, seq):`
`    for i in range(len(seq)):`
`        if x <= seq[i]:`
`            return i`
`        else:`
`            pass`
`    return len(seq)`
	`def search(e, lst):`
	`    for j in range(len(lst)):`
	`        if e < lst[j]:`
	`            return j`
	`        else:`
	`            j = j + 1`
	`    return len(lst) + 1`

- Dynamic equivalence analysis (trace based)
- Followed by define/use analysis (block based)

$\{x \Leftrightarrow e,\ seq \Leftrightarrow lst,\ i \Leftrightarrow j\}$

BLOCK MAPPING

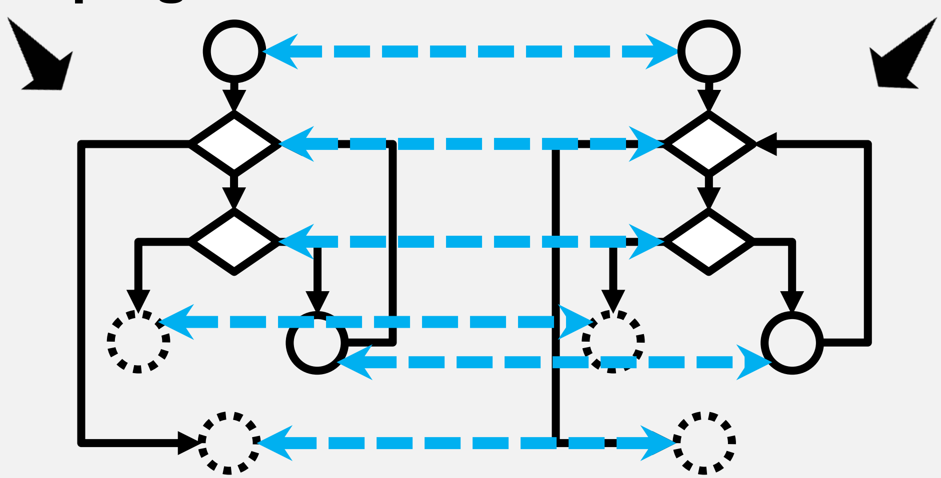
```
def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
        else:
            pass

    return len(seq)
```

```
def search(e, lst):
    for j in range(len(lst)):
        if e < lst[j]:
            return j
        else:
            j = j + 1

    return len(lst) + 1
```

Refactored correct program **Incorrect student program**



Block Mapping

STEP 3: INFER SPECIFICATION

Refactored Correct Solution	Incorrect Student Program

<pre>def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
        else:
            pass
    return len(seq)</pre>
	<pre>def search(e, lst):
	    for j in range(len(lst)):
	        if e < lst[j]:
	            return j
	        else:
	            j = j + 1
	    return len(lst) + 1</pre>

Input			Output	
x/e	seq/lst	i/j	x <= seq[i]	e < lst[j]
2	[1, 2, 3]	0	False	False
2	[1, 2, 3]	1	True	False
0	[1, 2, 3]	0	True	True

4. PATCH SYNTHESIS

`x <= seq[i] //Correct-Block`

`e < lst[j]  //Buggy-Block`

Expression Template

v_1 <= v_2[v_3]

```
e <= lst[j]
...
```

Operator Mutation

```
e <= lst[j]
e == lst[j]
e >= lst[j]
...
```

Variable Mutation

```
j < lst[j]
e < lst[e]
j < lst[e]
...
```

Block Patch Validation

Final Block Patch `e <= lst[j]`

Reference Solution	Incorrect Student Program

<pre>def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
 	
    return len(seq)</pre>
	<pre>def search(e, lst):
	    for j in range(len(lst)):
	        if e < lst[j]:
	            return j
	        else:
	            j = j + 1
	    return len(lst) + 1</pre>

Refactored Correct Solution	Incorrect Student Program

<pre>def search(x, seq):
    for i in range(len(seq)):
        if x <= seq[i]:
            return i
        else:
            pass
    return len(seq)</pre>
	<pre>def search(e, lst):
	    for j in range(len(lst)):
	        if e < lst[j]:
	            return j
	        else:
	            j = j + 1
	    return len(lst) + 1</pre>

Repair	Incorrect Student Program

<pre>def search(e, lst):
    for j in range(len(lst)):
        if e <= lst[j]:
            return j
        else:
            pass
    return len(lst)</pre>
	<pre>def search(e, lst):
	    for j in range(len(lst)):
	        if e < lst[j]:
	            return j
	        else:
	            j = j + 1
	    return len(lst) + 1</pre>

AUTOMATED PROGRAM REPAIR - BACKGROUND

Prof. Abhik Roychoudhury

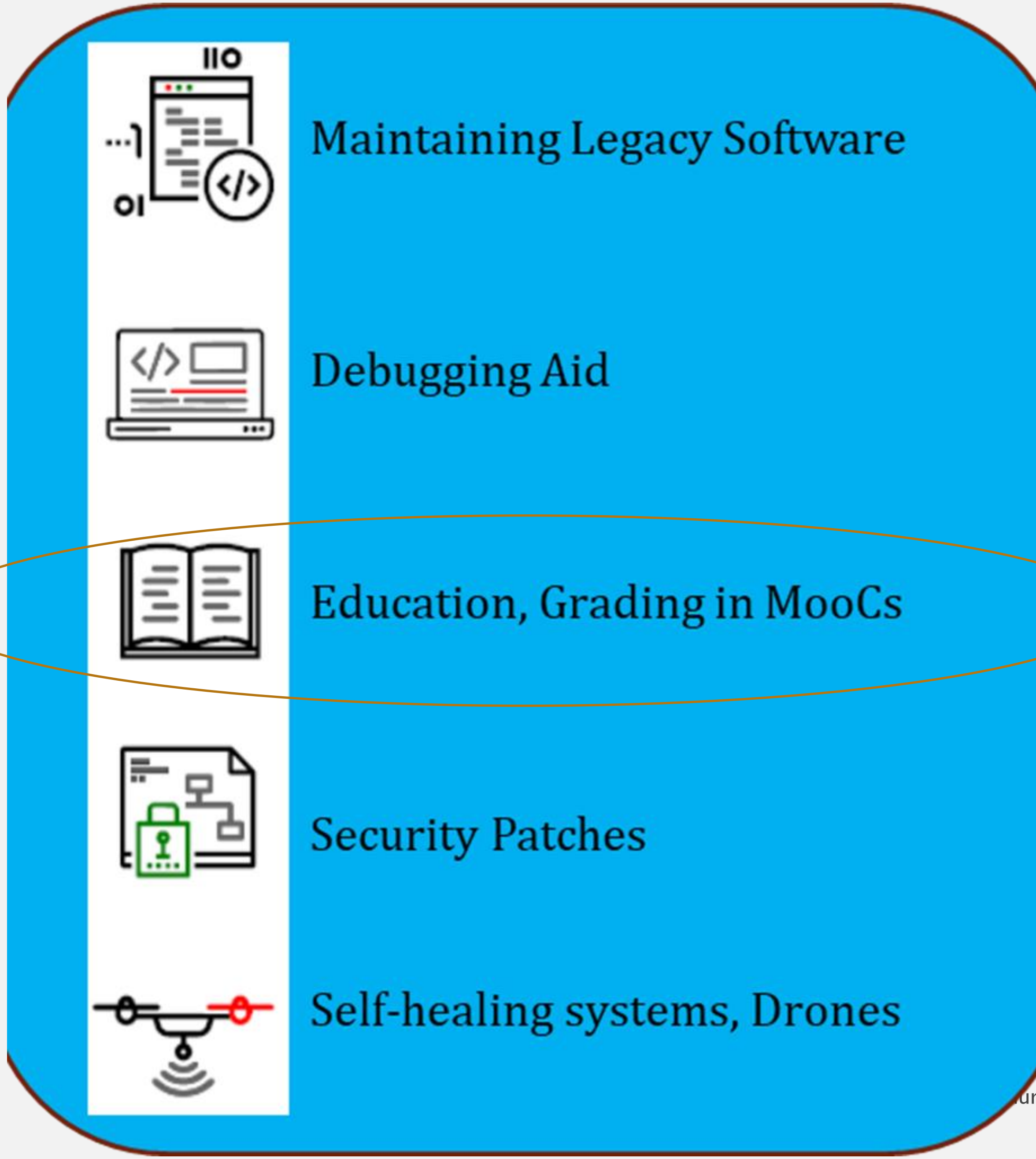
National University of Singapore

FIXING BUGS: HOW BAD IS IT?

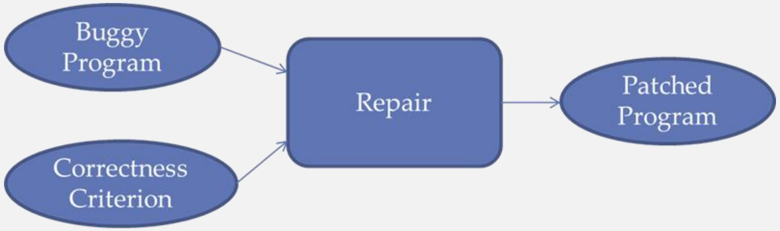
90% of cost and resources in software project

Legacy Crisis!

CS3213 FSE course by Abhik Roychoudhury


Maintaining Legacy Software
Debugging Aid
Education, Grading in MooCs
Security Patches
Self-healing systems, Drones

AUTOMATED PROGRAM REPAIR

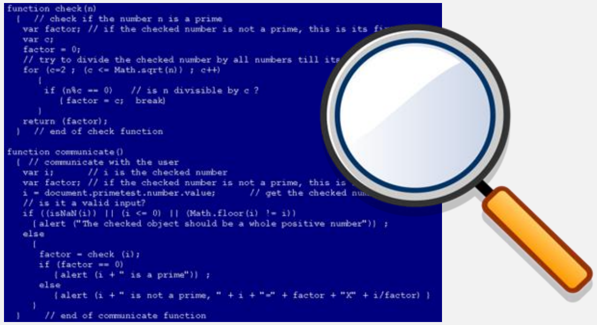


- Weak description of intended behavior / *correctness criterion* e.g. tests
- Weak applicability of repair techniques e.g. only overflow errors
- *Large search space* of candidate patches for general-purpose repair tools.
- Patch suggestions and *Interactive Repair*

DIVISION OF LABOR

Semantic Program Repair

Syntactic Program Repair



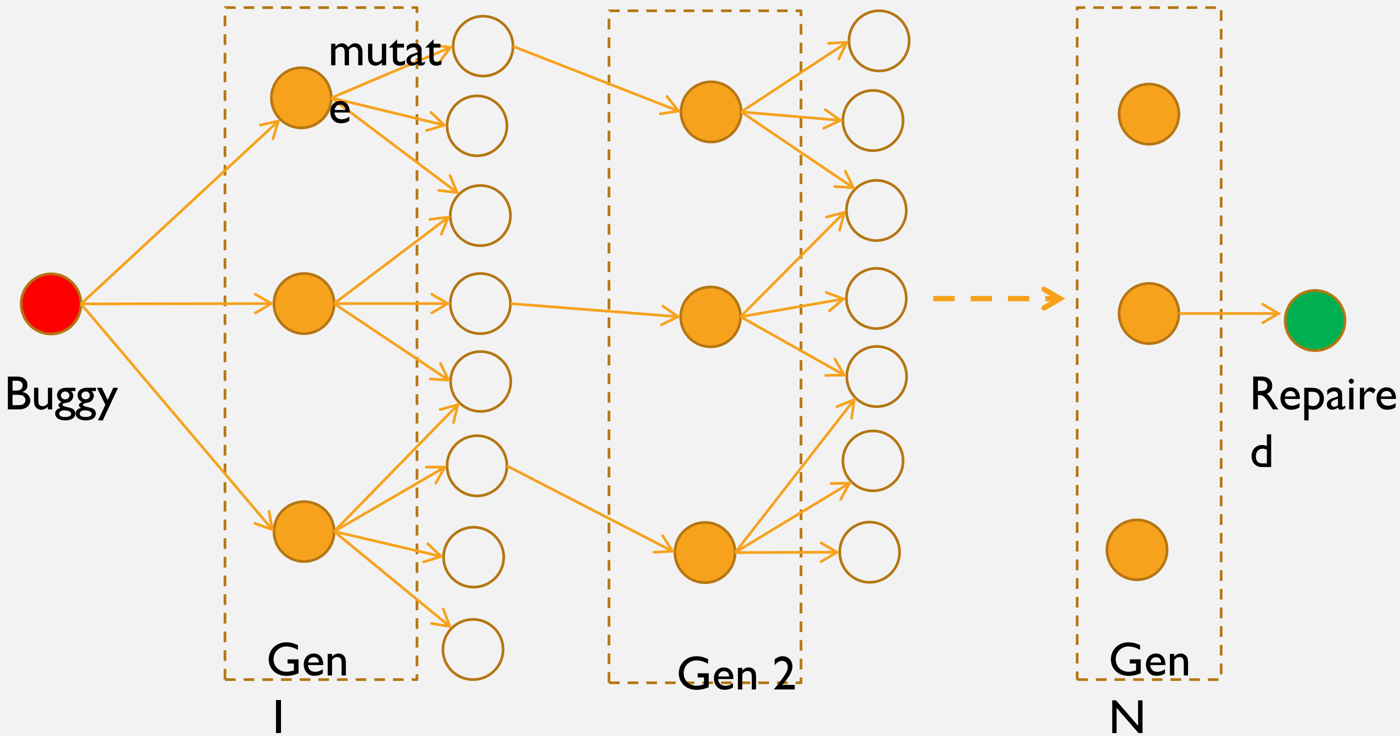
Syntactic Program Repair

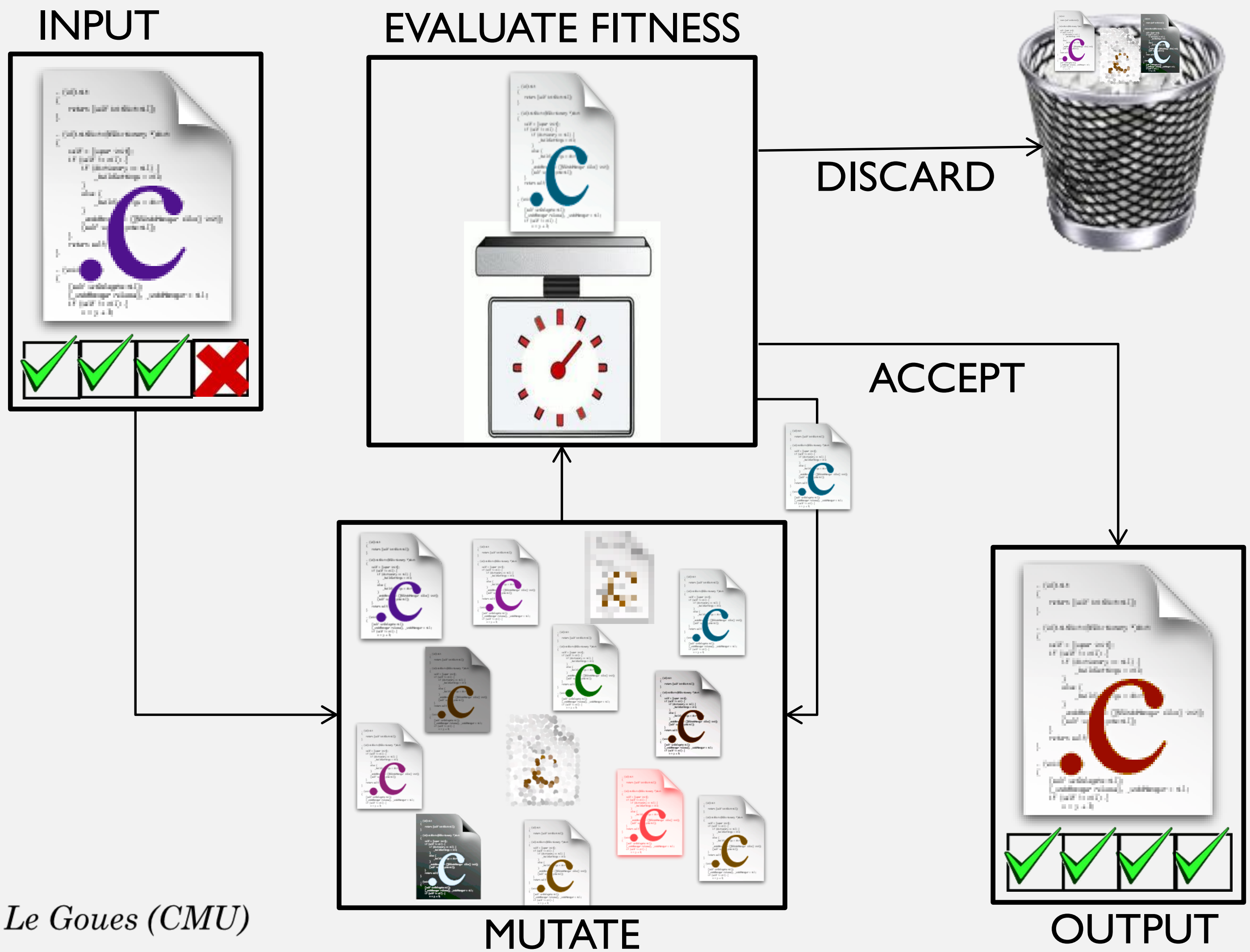
1. Where to fix, which line?
2. Generate patches in the candidate line
3. Validate the candidate patches against correctness criterion.

Semantic Program Repair

1. Where to fix, which line(s)?
2. What values should be returned by those lines, e.g. <inp ==1, ret== 0>
3. What are the expressions which will return such values?

GENPROG – REPAIR VIA SEARCH *(ACK: CLAIRE LE GOUES, 6 SLIDES)*





Ack: Claire Le Goues (CMU)

CANDIDATE PATCH

- An individual is a candidate patch or set of changes to the input program.
- A patch is a series of statement-level edits:
  - delete X
  - replace X with Y
  - insert Y after X.
- Replace/insert: pick Y from **somewhere else in the program.**

Ack: Claire Le Goues (CMU)

```
> gcd(4,2)
> 2
>
> gcd(1071,1029)
> 21
>
> gcd(0,55)
> 55
```

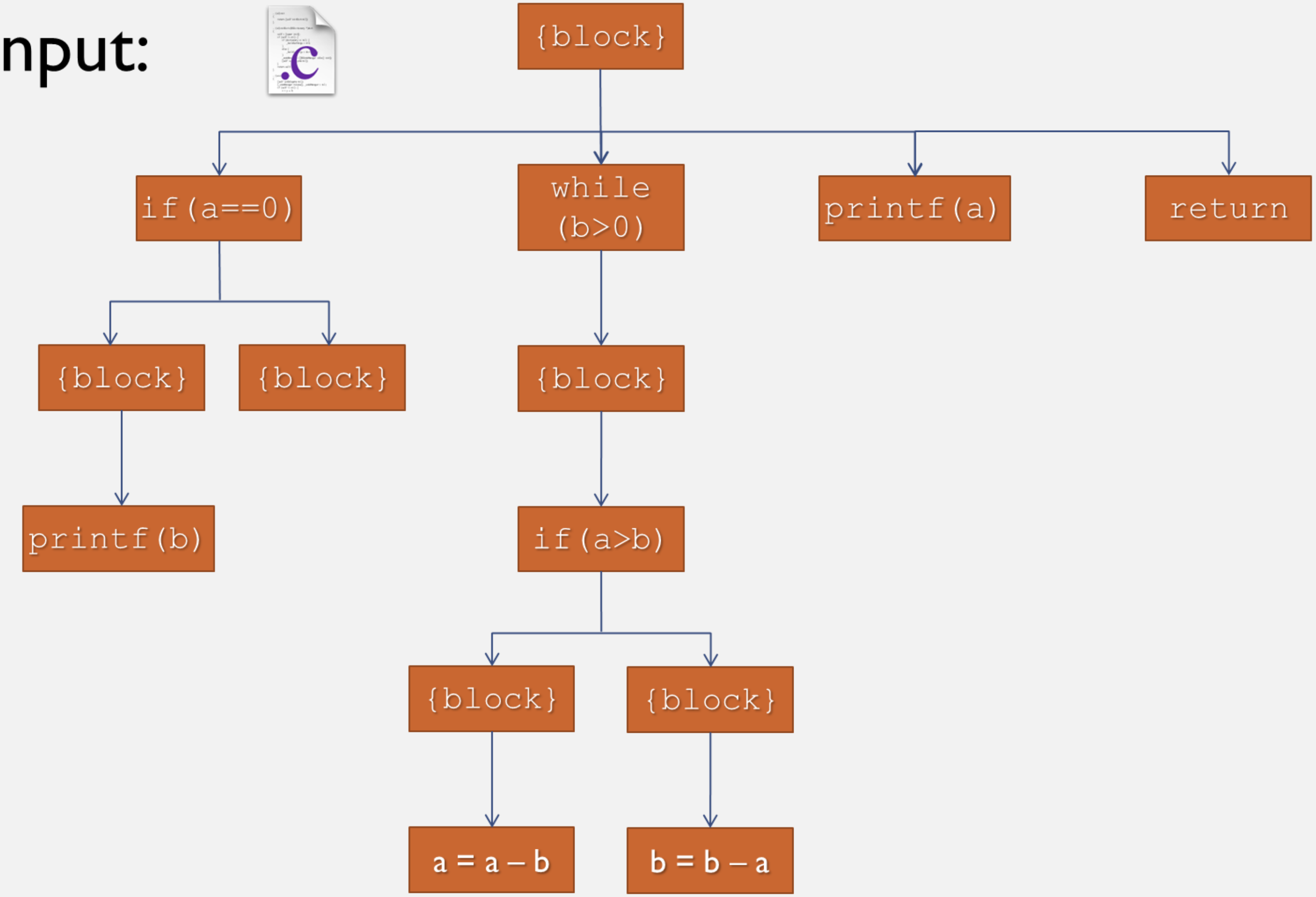
(looping forever)

```c
void gcd(int a, int b) {
  if (a == 0) {
    printf("%d", b);
  }
  while (b > 0) {
    if (a > b)
      a = a - b;
    else
      b = b - a;
  }
  printf("%d", a);
  return;
}
```

Ack: Claire Le Goues (CMU)

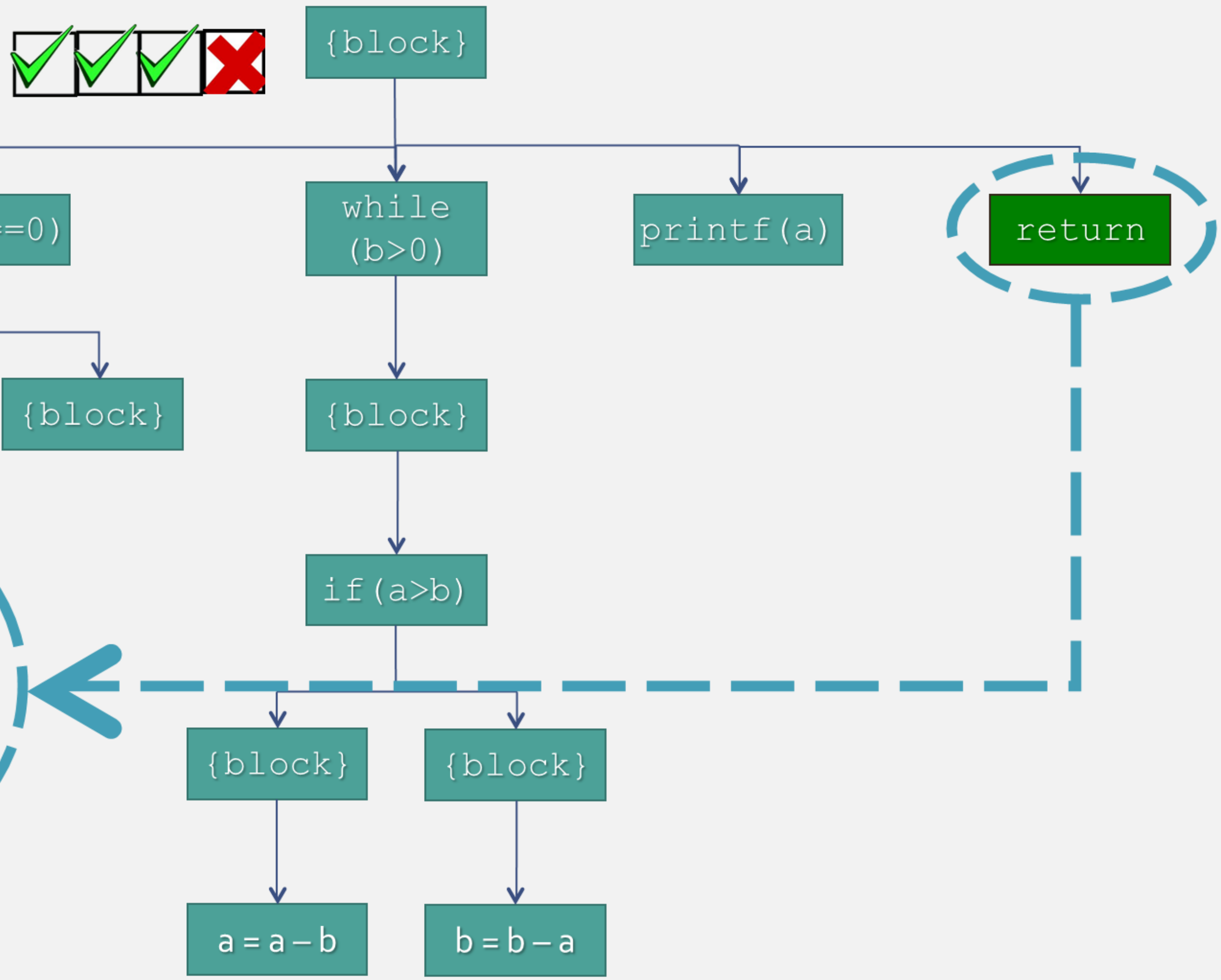
PROGRAM REPRESENTATION

Input:



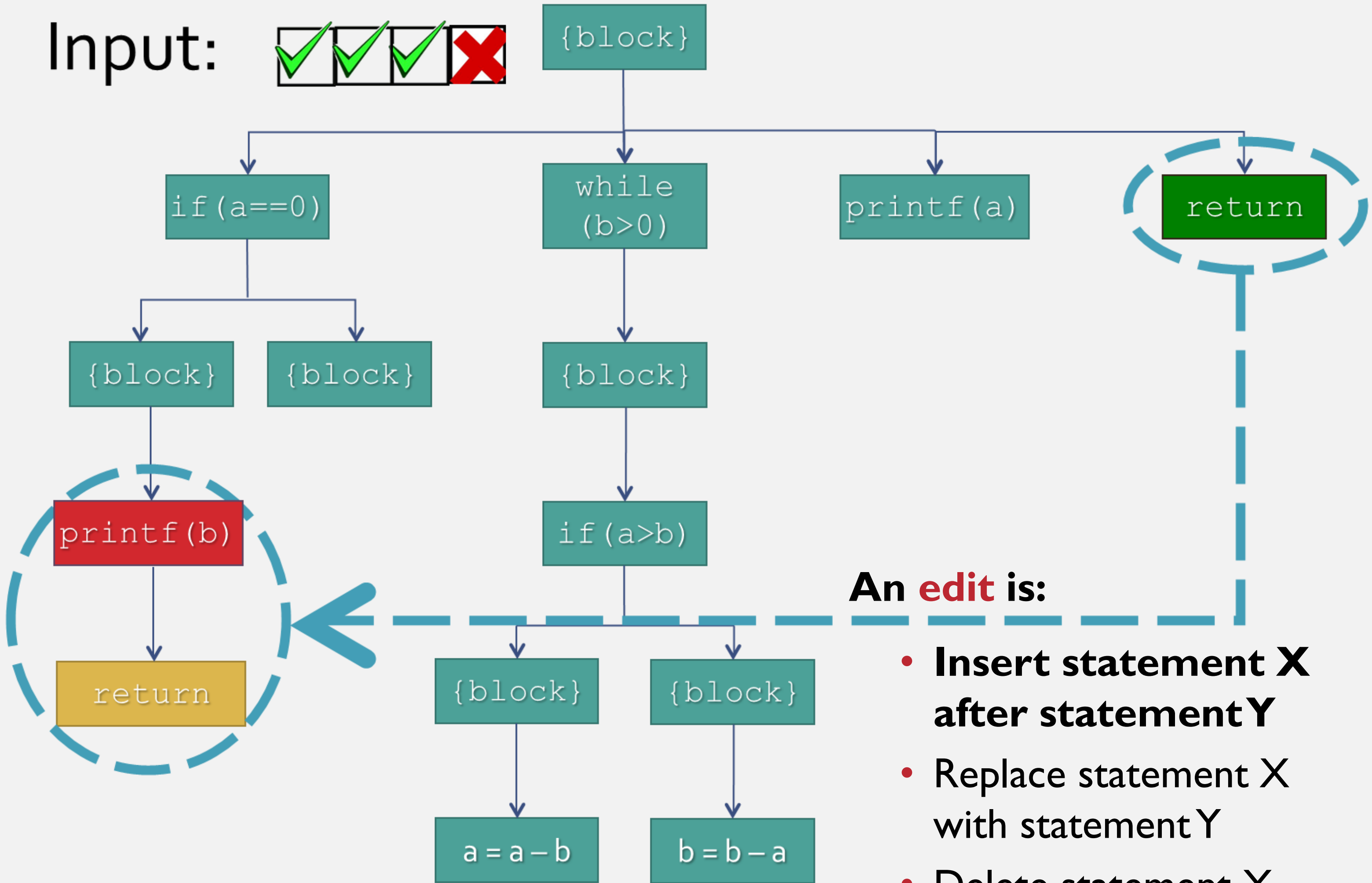
Ack: Claire Le Goues (CMU)

Input:



Ack: Claire Le Goues (CMU)

Input:



An **edit** **is:**

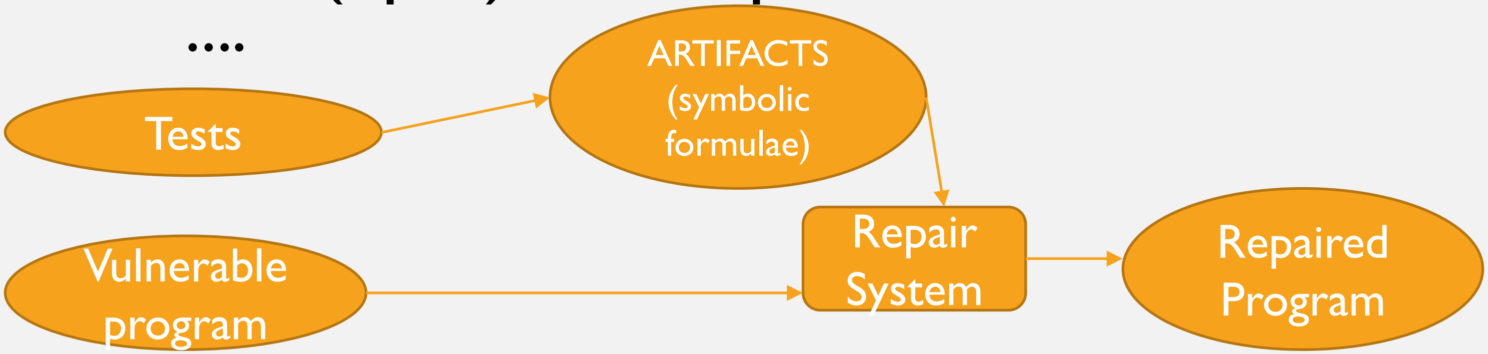
- **Insert statement X after statement Y**
- Replace statement X with statement Y
- Delete statement X

Ack: Claire Le Goues (CMU)

OVER-FITTING IN REPAIR

Avoid generating programs like

if (input1) return output1
else if (input2) return output2
else if (input3) return output3
….



Generalize beyond the provided tests using symbolic reasoning.

COMPARISON

Syntactic Program Repair

Syntax-based Schematic
for e in Search-space{
 Validate e against Tests
}

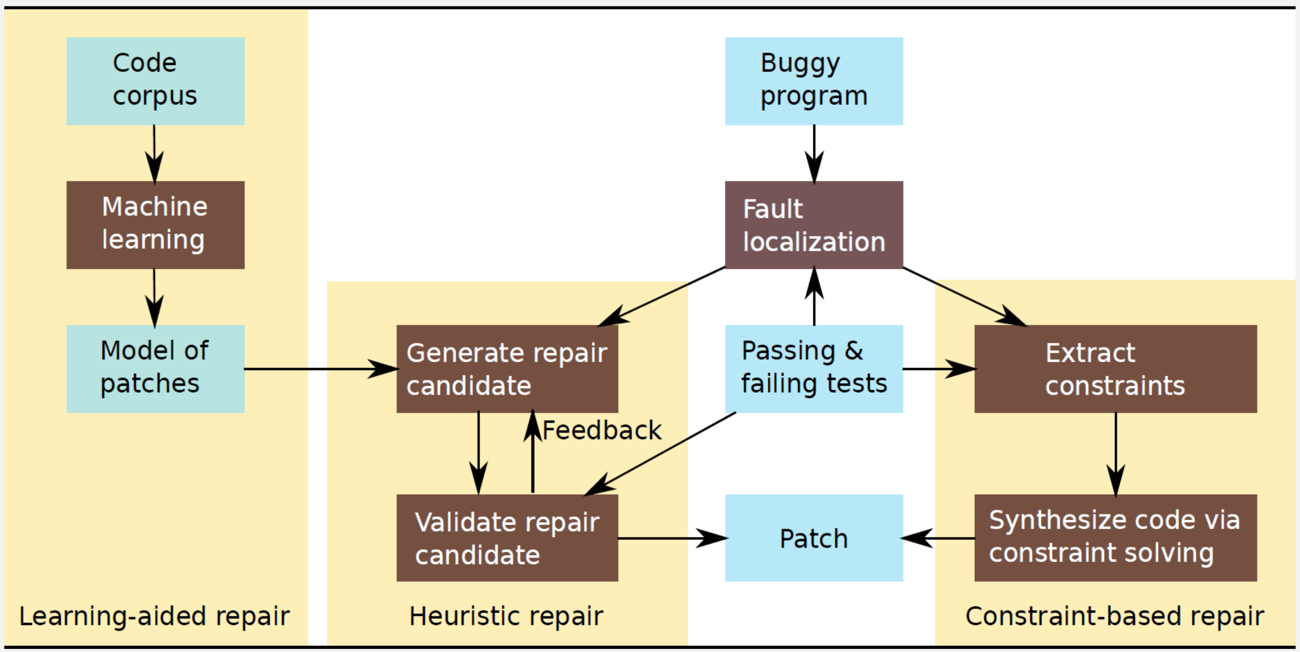
1. Where to fix, which line?
2. Generate patches in the candidate line
3. Validate the candidate patches against correctness criterion.

Semantic Program Repair

Semantics-based Schematic
for t in Tests {
 generate repair constraint Ψ_t
}
Synthesize e from $\wedge_t \Psi_t$

1. Where to fix, which line(s)?
2. What values should be returned by those lines, e.g. <inp ==1, ret== 0>
3. What are the expressions which will return such values?

STATE-OF-THE-ART



Ack: Figure from Reading in our class, "Automated Program Repair" by Le Goues, Pradel, Roychoudhury, article in Communications of the ACM, 2019.

```
1 int triangle(int a, int b, int c){
2     if (a <= 0 || b <= 0 || c <= 0)
3         return INVALID;
4     if (a == b && b == c)
5         return EQUILATERAL;
6     if (a == b || b != c) // bug!
7         return ISOSCELES;
8 return SCALENE;
9 }
```

Correct fix
(a == b || b == c || a== c)

Test id	a	b	c	oracle	Pass
1	-1	-1	-1	INVALID	pass
2	1	1	1	EQUILATERAL	pass
3	2	2	3	ISOSCELES	pass
4	2	3	2	ISOSCELES	fail
5	3	2	2	ISOSCELES	fail
6	2	3	4	SCALENES	fail

Traverse all *mutations* of line 6, and check

Hard to generate correct fix since a==c never appears elsewhere in the program.

OR

Generate the constraint

$f(2,2,3) \wedge f(2,3,2) \wedge f(3,2,2) \wedge \neg f(2,3,4)$

And get the solution

```
f(a,b,c) = (a == b||b == c || a== c)
```

APPLICATION IN EDUCATION: FEASIBILITY

Prof. Abhik Roychoudhury

National University of Singapore

NOVEL APPLICATIONS: INTELLIGENT TUTORING

Use program repair in intelligent tutoring systems to give the students' individual attention.

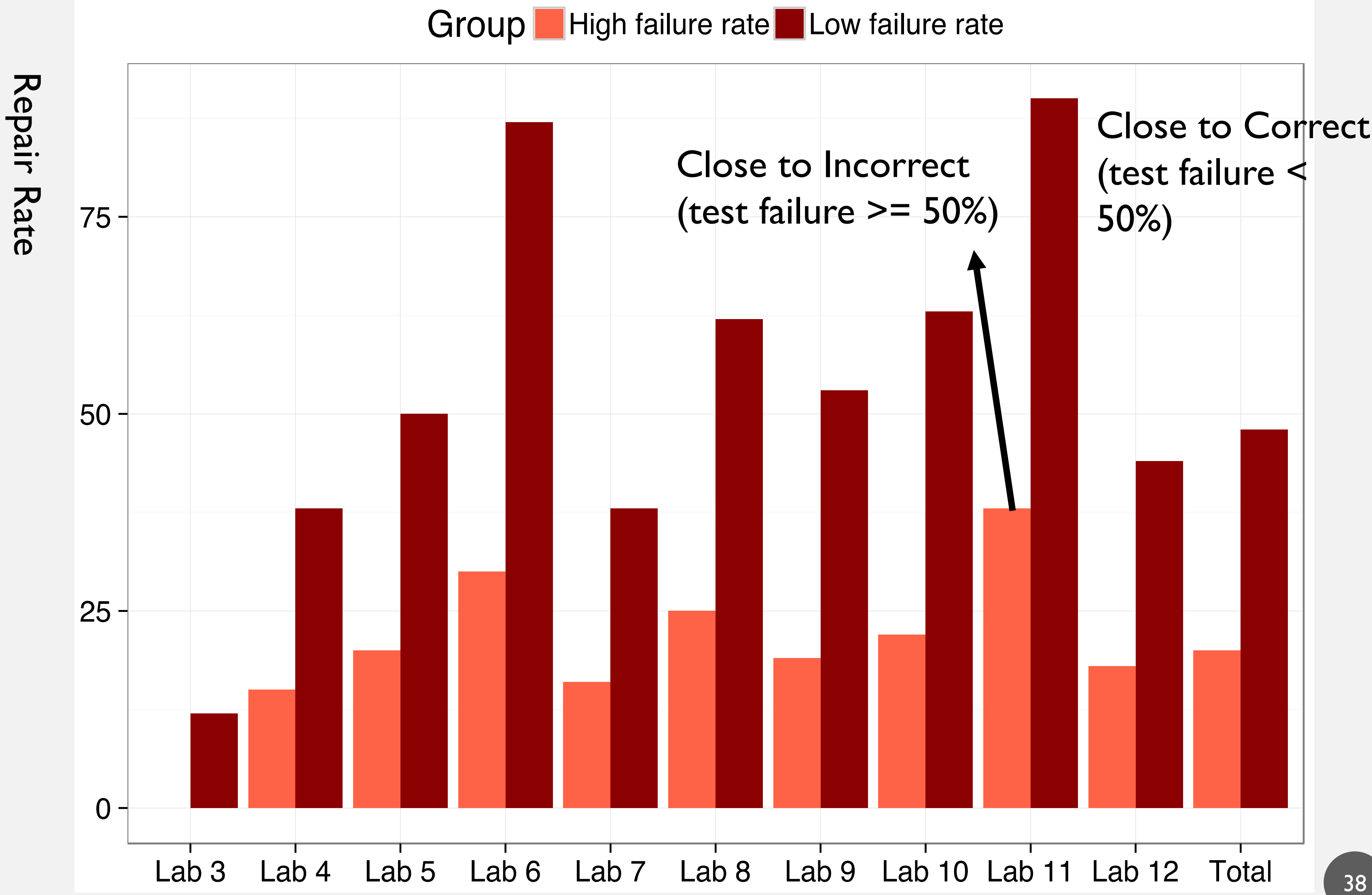
Conducted user studies

DATASET USED IN STUDIES

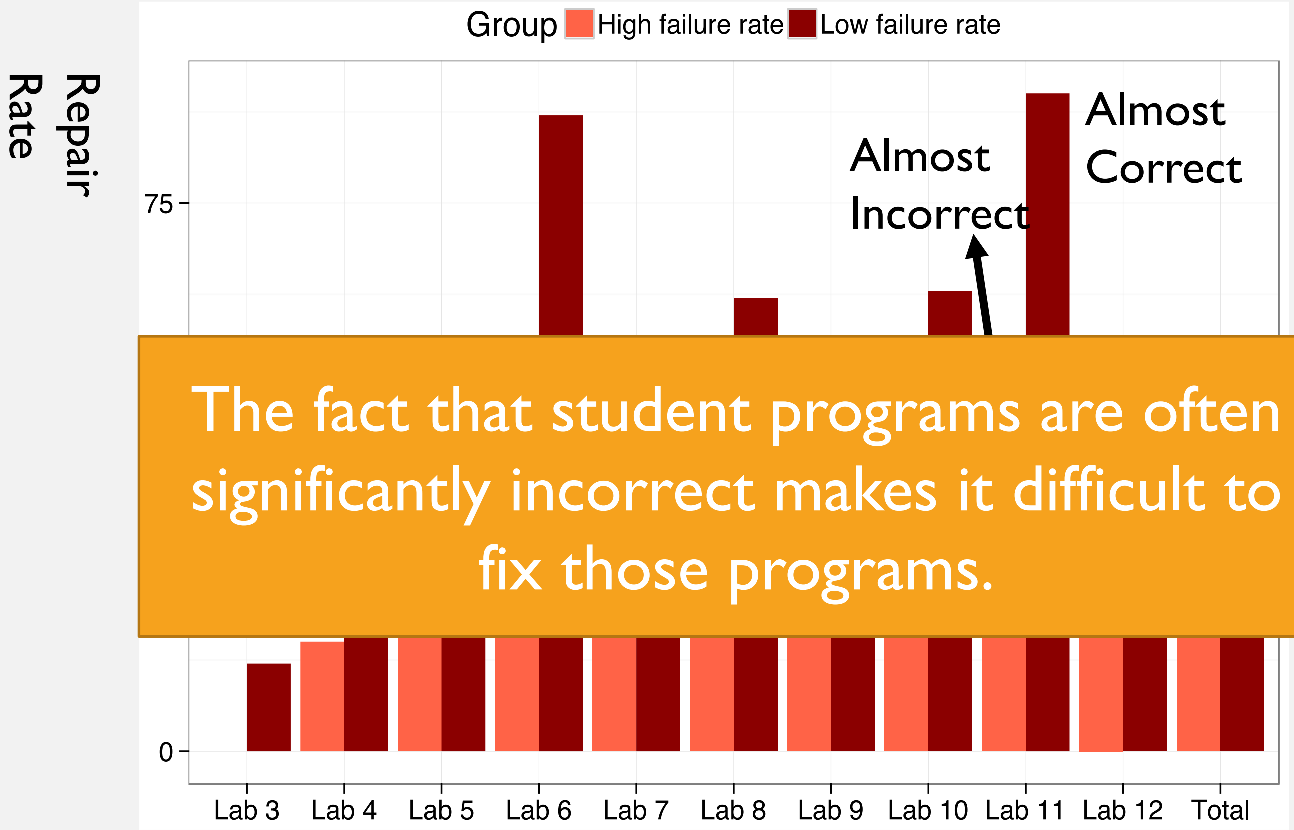
- Lab: Programming assignments

Lab	# Prog	Topic
Lab 3	63	Simple Expressions, printf, scanf
Lab 4	117	Conditionals
Lab 5	82	Loops, Nested Loops
Lab 6	79	Integer Arrays
Lab 7	71	Character Arrays (Strings) and Functions
Lab 8	33	Multi-dimensional Arrays (Matrices)
Lab 9	48	Recursion
Lab 10	53	Pointers
Lab 11	55	Algorithms (sorting, permutations, puzzles)
Lab 12	60	Structures (User-Defined data-types)

CLOSE TO INCORRECT VS CLOSE TO CORRECT

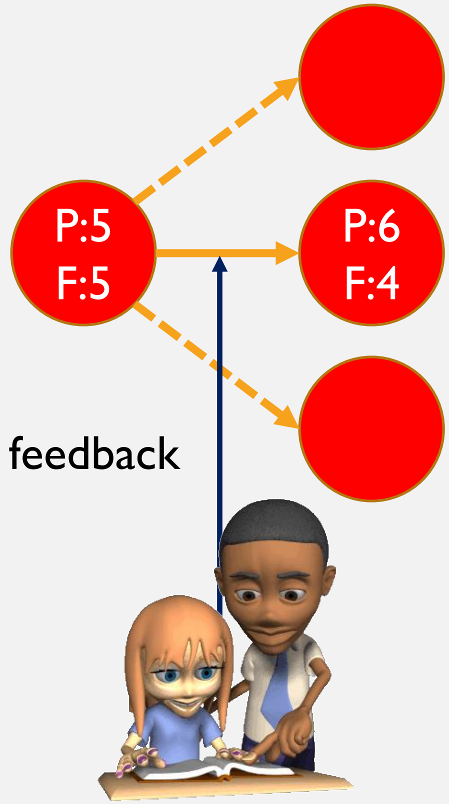


ALMOST INCORRECT VS ALMOST CORRECT



The fact that student programs are often significantly incorrect makes it difficult to fix those programs.

PARTIAL REPAIR AS A HINT



- Control-flow hints
  - change of if-conditionals
  - change of loop-exit conditions
- Data-flow hints
  - adding/deleting statements
- Conditional data-flow hints:

```
if (/* guard condition */) {
    /* a data-flow hint */
}
```

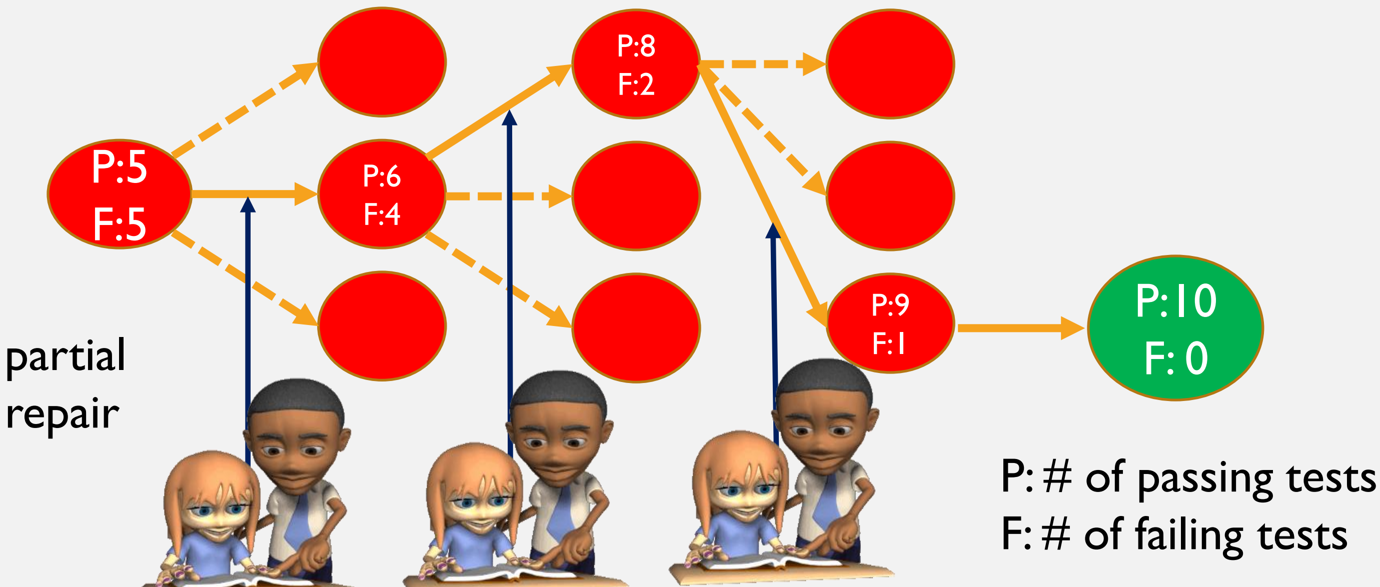
NB: {Conditional data-flow hints} $\supset$ {Data-flow hints}

TAILORED REPAIR STRATEGY

- Look for the following in parallel
  - a control-flow hint
  - a conditional data-flow hint
- Benefits
  - Reduce the search space of each repair tool
  - Combine multiple repair tools in a complementary way
    - A conditional data-flow hint can be composed of
      1. a data-flow hint from search-based repair
      2. a guarded condition from semantic repair

```
if (/* guard condition */) {
    /* a data-flow hint */
}
```

TAILORING REPAIR POLICY



Partial Repair: (all previously passing tests) + (at least one previously failing test)

TWO-STEP REPAIR

Test 1
Test 2
Test 3
Test 4
Test 5

```
+ if (true) {
+     S';
+ } else {
      S;
+ }
```

Test 1
Test 2
Test 3
Test 4
Test 5

```
+ if (E) {
+     S';
+ } else {
      S;
+ }
```

Test 1
Test 2
Test 3
Test 4
Test 5

TWO-STEP REPAIR

Test 1
Test 2
Test 3
Test 4
Test 5

```
+ if (true) {
+     S';
+ } else {
      S;
+ }
```

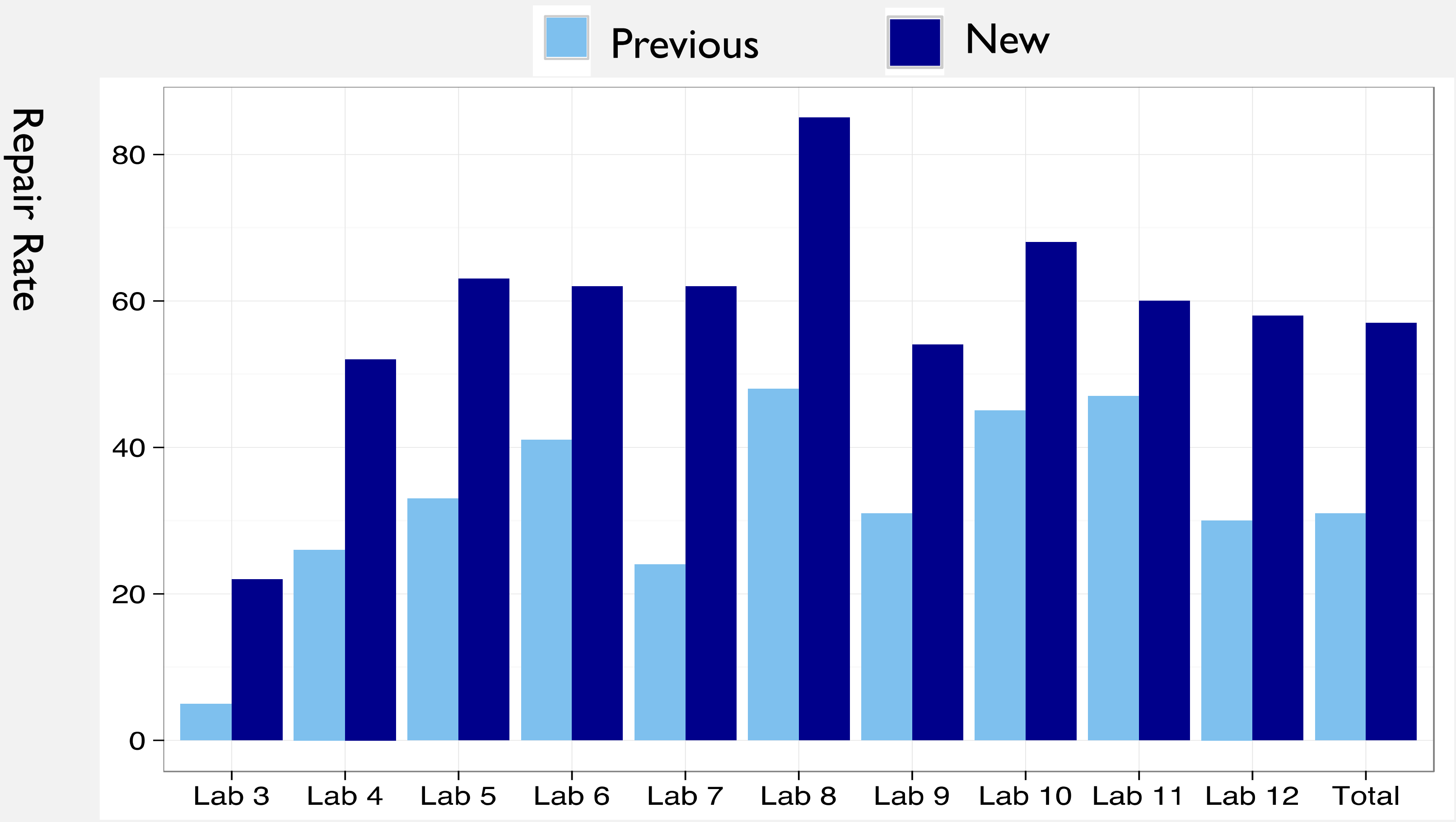
Test 1
Test 2
Test 3
Test 4
Test 5

```
+ if (E) {
+     S';
+ } else {
      S;
+ }
```

Test 1
Test 2
Test 3
Test 4
Test 5

NEW RESULT

- 84% improvement



CONCLUSION

- Out-of-the-box application of APR tools to ITS is infeasible
- Positive result after adopting
  - a new repair policy accepting partial repairs
  - a new repair strategy
- Further improvement seems possible by refining repair operators (e.g., strings and arrays)

- Reading
- https://www.comp.nus.edu.sg/~abhik/pdf/FSE17.pdf

CONCLUSION

- User study:
  - TA's grading performance improves.
  - Novice students do not seem to know how to effectively make use of repairs.
- Future work:
  - How to transform repairs into hints more comprehensible to novice students?
  - We share our dataset and toolset
    - https://github.com/jyi/ITSP

TUTORING – BEYOND REPAIR