Automatically Finding Patches Using Genetic Programming

Westley Weimer
Claire Le Goues
ThanVu Nguyen
Stephanie Forrest

Motivation

- Software Quality remains a key problem
  - Over one half of 1 percent of US GDP each year
  - Programs ship with known bugs
- Reduce debugging costs
  - Bug reports accompanied by patches are addressed more rapidly
- Automated Patch Generation
  - Transform a program with a bug
  - Into a program without the bug
  - By modifying relevant parts of the program

The Cunning Plan

- We can automatically and efficiently repair certain classes of bugs in off-the-shelf, unannotated legacy programs.
- Basic idea: Biased search through the space of all programs until you find a variant that repairs the problem. Key insights:
  - Use existing **test cases** to evaluate variants.
  - Search by perturbing parts of the program **likely** to contain the error.

The Process

- Input:
  - The program source code
  - Some regression test cases passed by the program
  - A test case failed by the program (= the bug)
- Genetic Programming Work:
  - Create variants of the program
  - Run them on the test cases
  - Repeat, retaining and combining variants
- Output:
  - New program source code that passes all tests
  - *or* “no solution found in time”

This Talk

- Genetic Programming
- Weighted Paths
- Our Technique
- Example
- Repair Experiments
- Big Finish

What's In A Name?

- **Genetic programming** is the application of evolutionary or genetic algorithms to program source code.
  - Population of variants
  - Mutation, crossover
  - Fitness function
- Similar in ways to search-based software engineering:
  - Regression tests to guide the search

Two Secret Sauces

- In a large program, not every line is equally likely to contribute to the bug.
- Insight: since we have the test cases, run them and collect coverage information.
- The bug is more likely to be found on lines visited when running the failed test case.
- The bug is less likely to be found on lines visited when running the passed test cases.
- Also: Do not try to invent new code!

The Weighted Path

- We define a weighted path to be a list of <statement, weight> pairs.
- We use this weighted path:
  - The statements are those visited during the failed test case.
  - The weight for a statement S is
    – High (1.0) if S is not visited on a passed test
    – Low (0.1, 0.0) if S is also visited on a passed test

Genetic Programming for Program Repair: Mutation

- Population of Variants:
  - Each variant is an <AST, weighted path> pair
- Mutation:
  - To mutate a variant V = <AST_V, wp_V>, choose a statement S from wp_V biased by the weights
  - Delete S, replace S with S1, or insert S2 after S
    - Choose S1 and S2 from the entire AST
  - Assumes program contains the seeds of its own repair (e.g., has another null check elsewhere).

Genetic Programming for Program Repair: Fitness

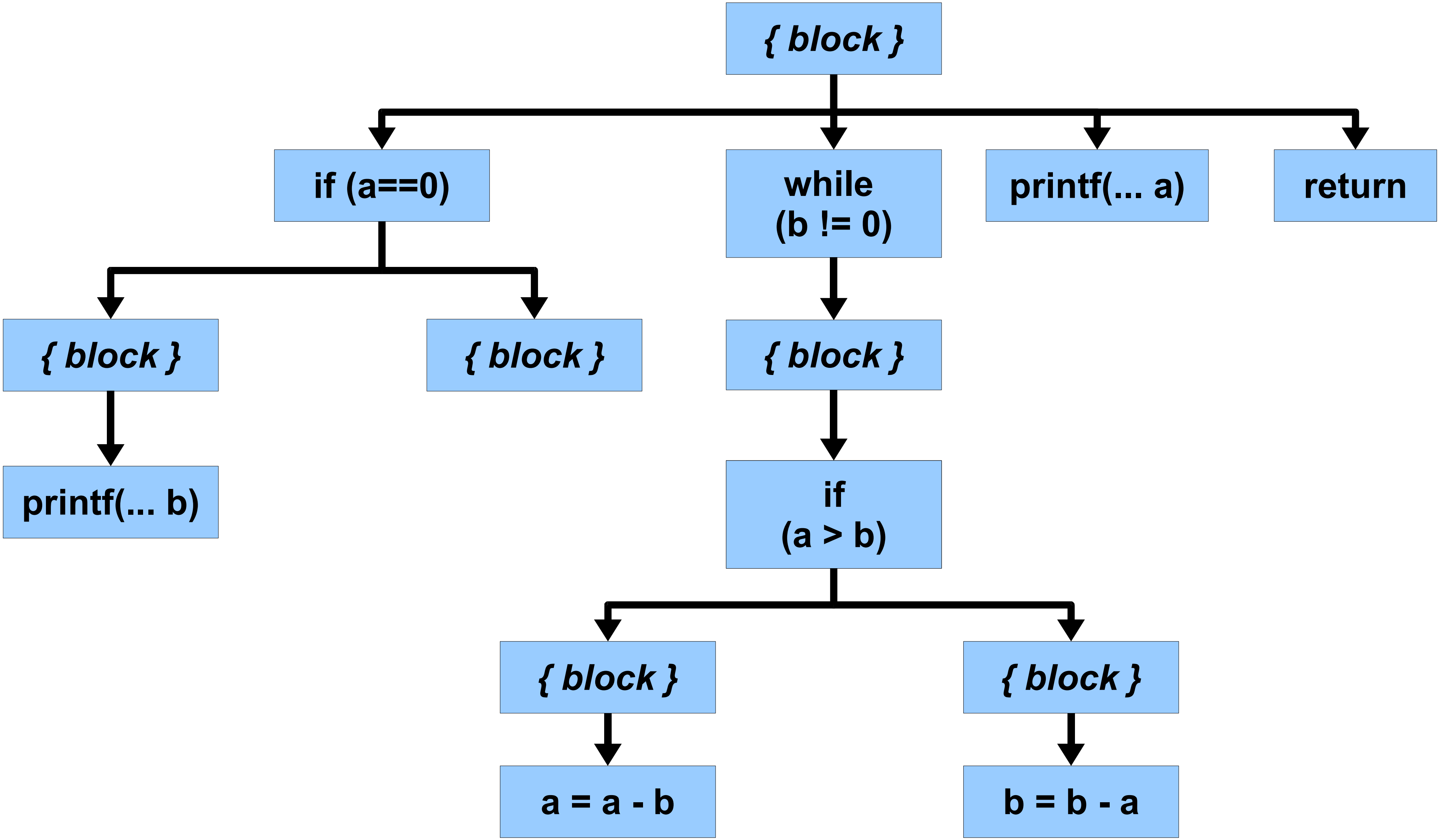
- Compile a variant
  - If it fails to compile, Fitness = 0
  - Otherwise, run it on the test cases
  - Fitness = number of test cases passed
  - Weighted: passing the bug test case is worth more
- Selection and Crossover
  - Higher fitness variants are retained and combined into the next generation
- Repeat until a solution is found

Example: GCD

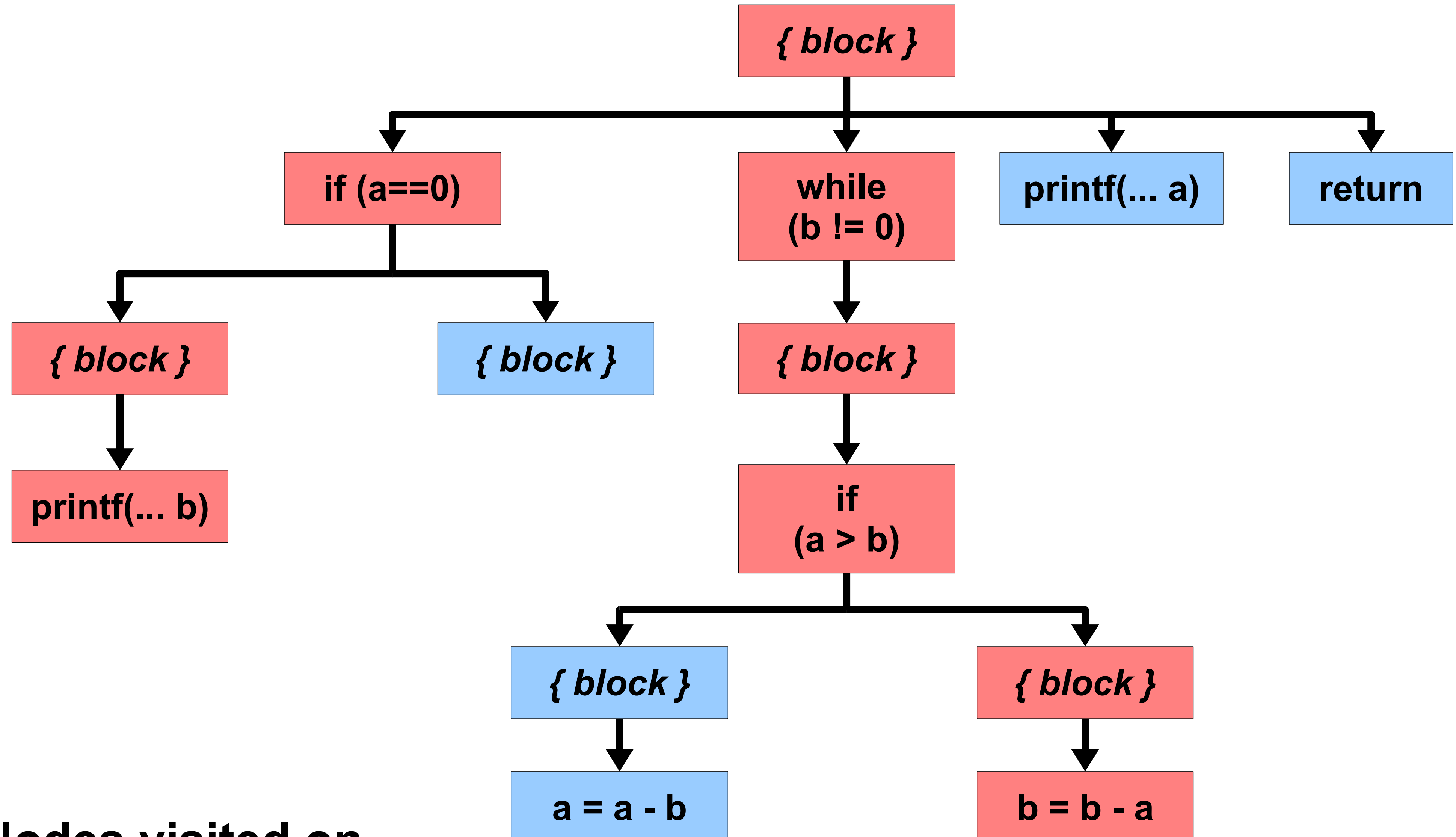
```
/* requires: a >= 0, b >= 0 */
void print_gcd(int a, int b) {
  if (a == 0)
    printf(“%d”, b);
  while (b != 0) {
    if (a > b)
      a = a – b;
    else
      b = b – a;
  }
  printf(“%d”, a);
  return;
}
```

Bug: when a==0 and b>0, it loops forever!

Example: Abstract Syntax Tree

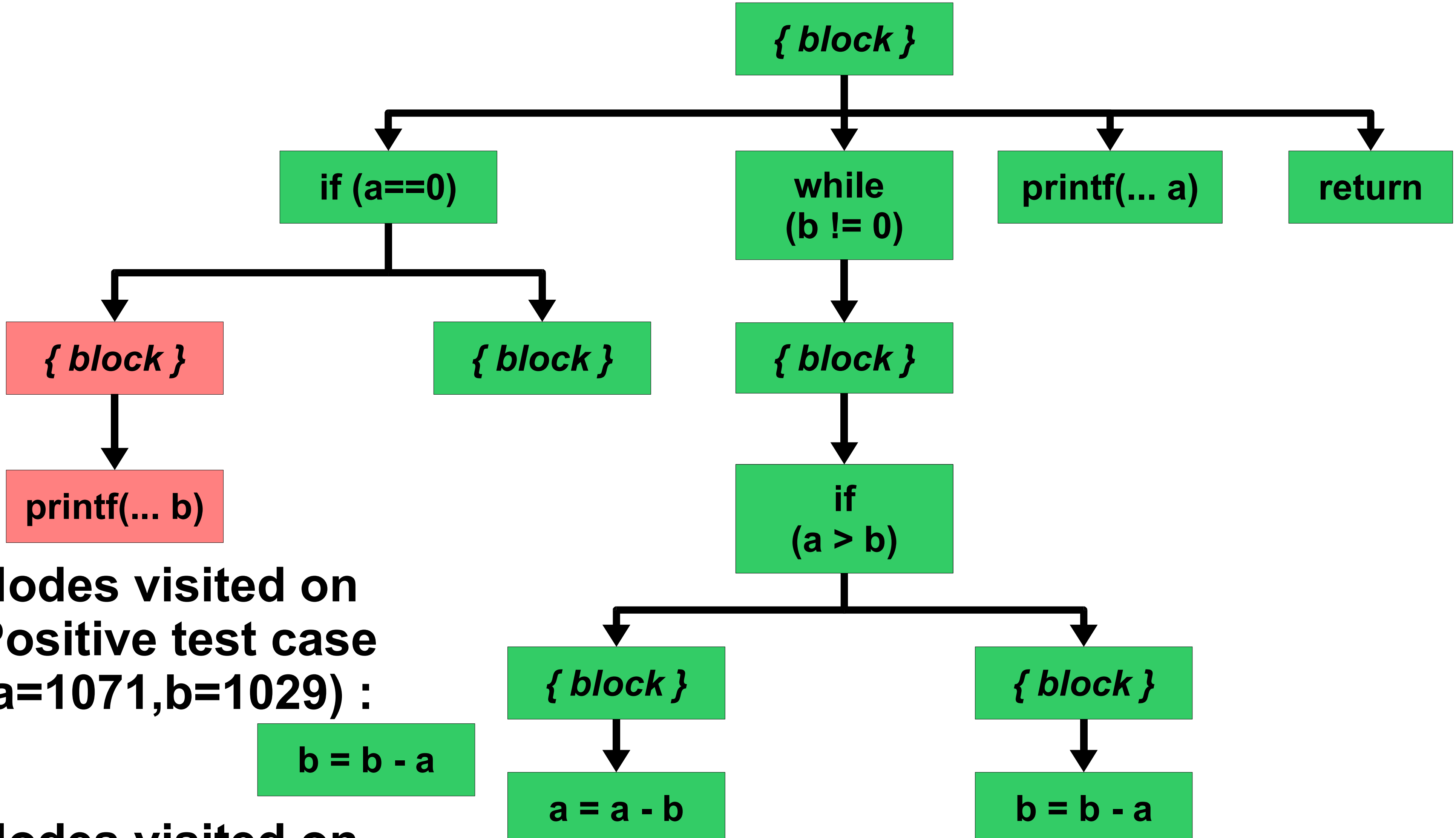


Example: Weighted Path (1/3)



Nodes visited on Negative test case (a=0,b=55) : (printf ...b)

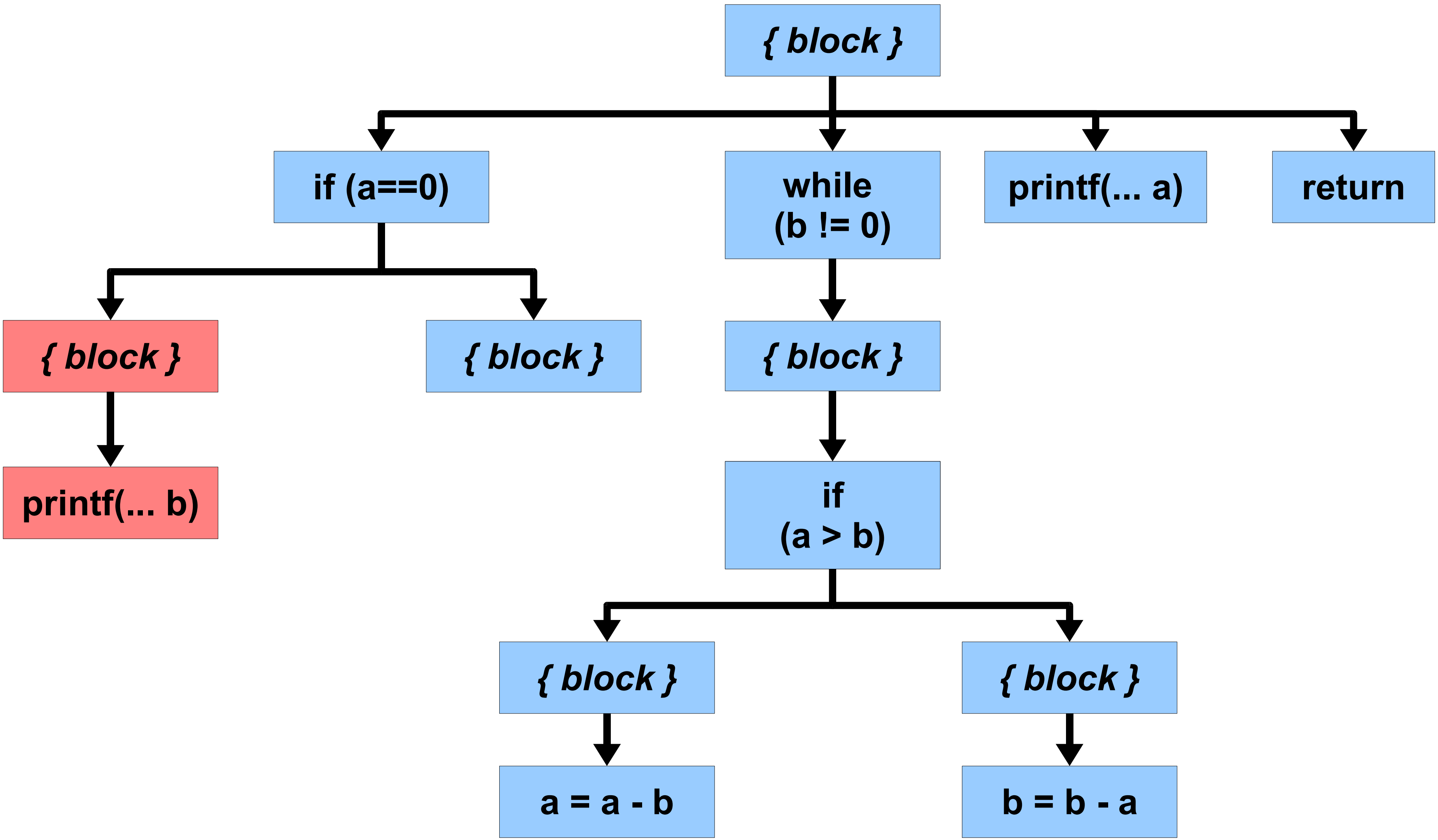
Example: Weighted Path (2/3)



Nodes visited on Positive test case (a=1071,b=1029) : b = b - a

Nodes visited on Negative test case (a=0,b=55) : (printf ...b)

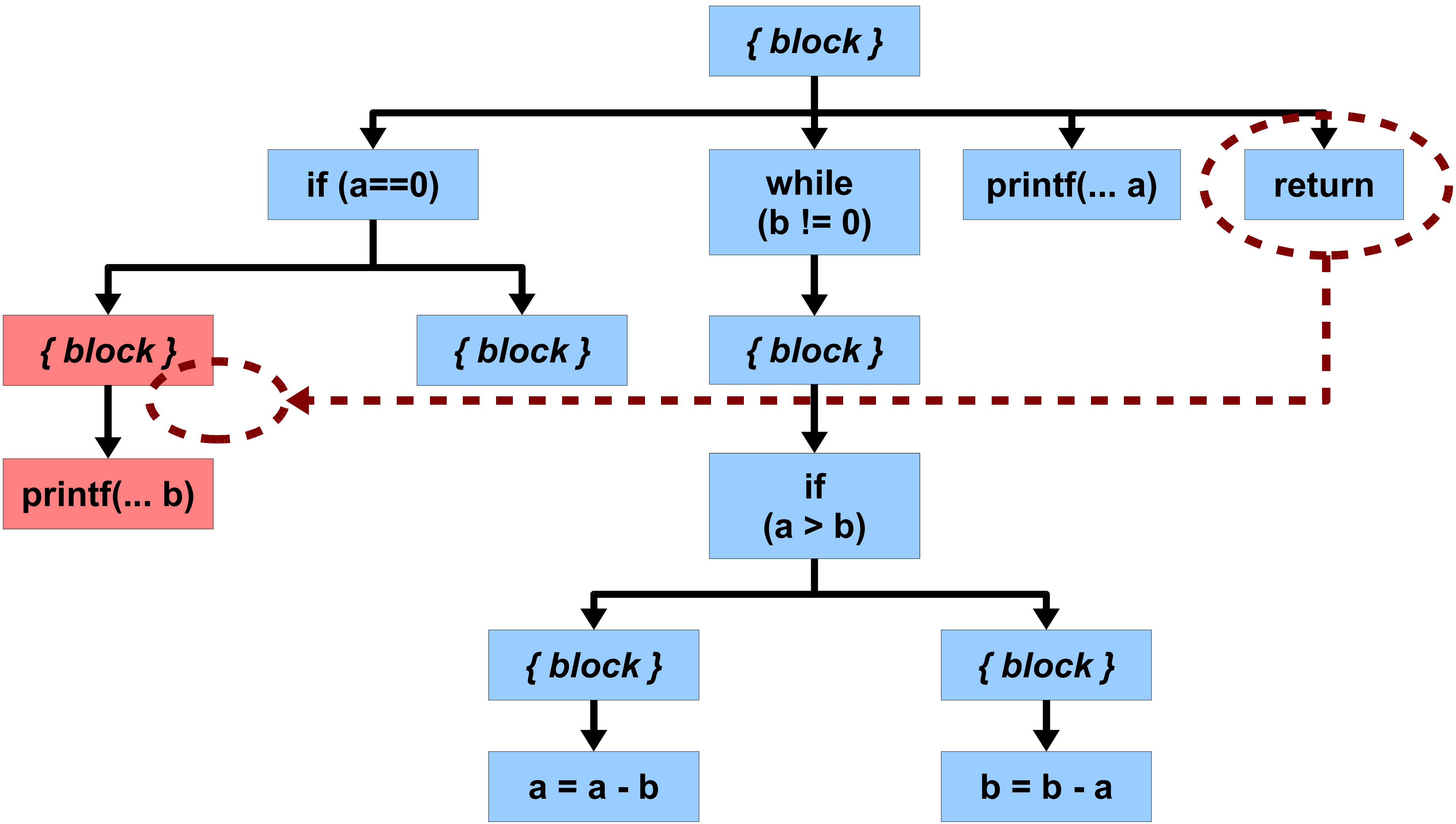
Example: Weighted Path (3/3)



Weighted Path:

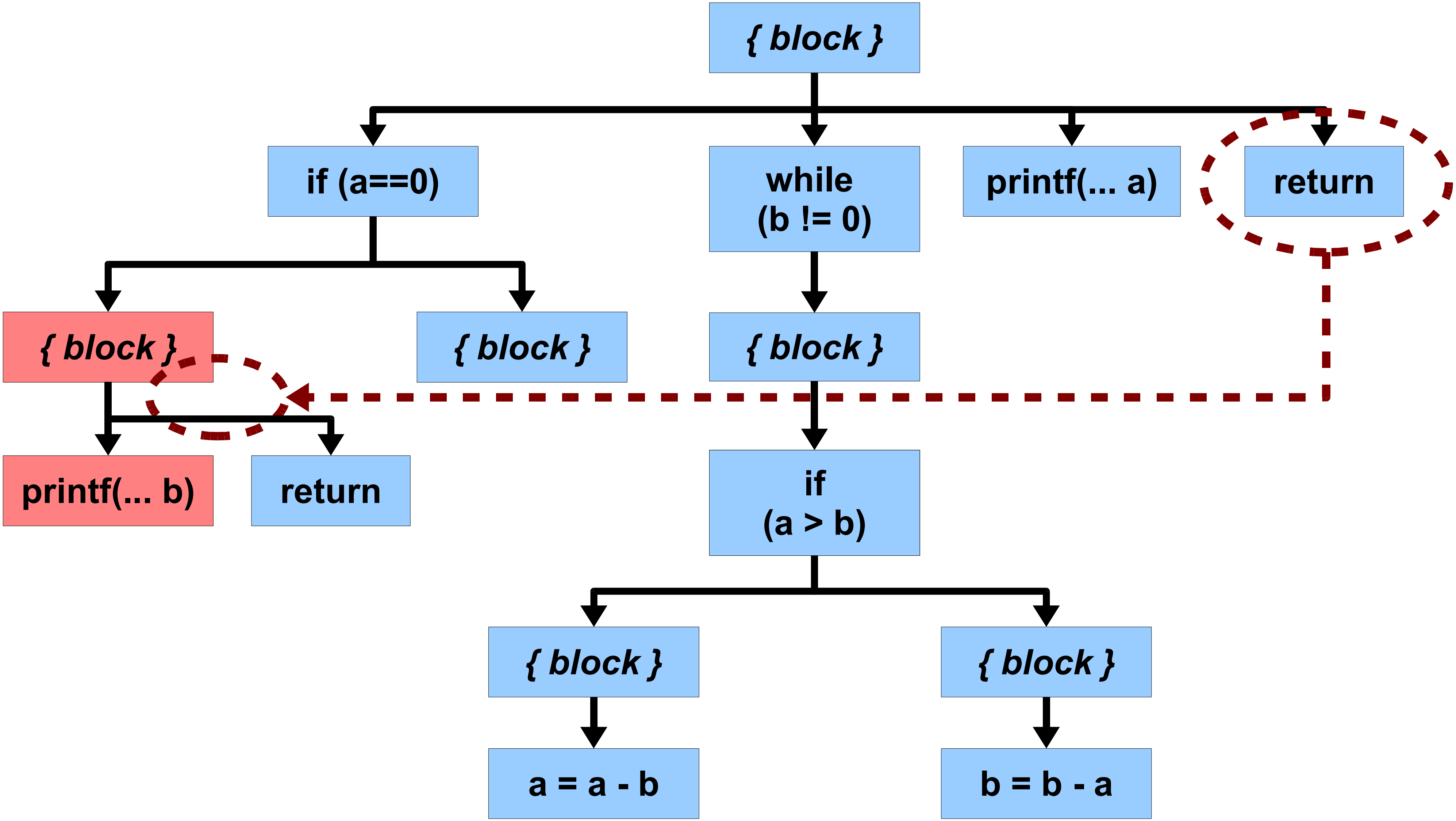
(printf ...b)

Example: Mutation (1/2)



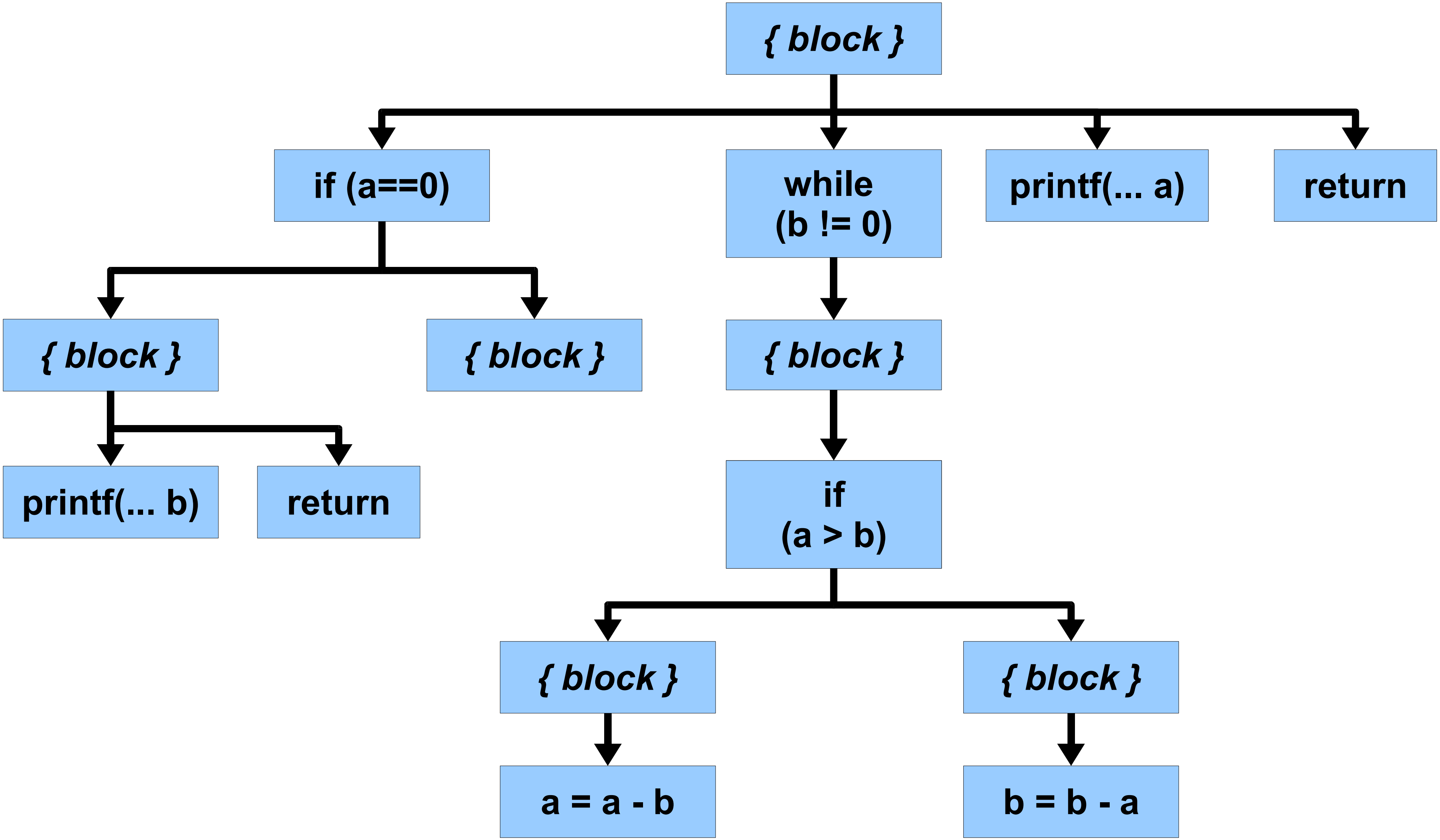
Mutation Source: Anywhere in AST
Mutation Destination: Weighted Path

Example: Mutation (2/2)



Mutation Source: Anywhere in AST
Mutation Destination: Weighted Path

Example: Final Repair



Minimize The Repair

- Repair Patch is a diff between orig and variant
- Mutations may add unneeded statements
  - (e.g., dead code, redundant computation)
- In essence: try removing each line **in the diff** and check if the result still passes all tests
- Delta Debugging finds a 1-minimal subset of the diff in $O(n^2)$ time
  - Removing any single line causes a test to fail
- We use a tree-structured diff algorithm (diffX)
  - Avoids problems with balanced curly braces, etc.

Experimental Results

Program	LOC	| Path |	Time (s)	Success	Bug
gcd	22	1.3	149	54%	inf loop
uniq	1146	81.5	32	100%	segfault
ultrix look	1169	213.0	42	99%	segfault
svr4 look	1363	32.4	51	100%	inf loop
units	1504	2159.7	107	7%	segfault
deroff	2236	251.4	129	97%	segfault
nullhttpd	5575	768.5	502	36%	buffer overrun
indent	9906	1435.9	533	7%	inf loop
flex	18775	3836.6	233	5%	segfault
atris	21553	34.0	69	82%	buffer overrun
average		881.4	184.7	58.7%	

Average minimization time: 12 seconds.
Total: **10** repaired programs, over **63,000** lines of code.

Repair Quality

- Repairs are typically *not* what a human would have done
  - Example: our technique adds bounds checks to one particular network read, rather than refactoring to use a safe abstract string class in multiple places
- Recall: any proposed repair must pass **all** regression test cases
  - When POST test is omitted from nullhttpd, the generated repair eliminates POST functionality
  - Tests ensure we do not sacrifice functionality
  - Minimization prevents gratuitous deletions
  - Adding more tests helps rather than hurting

Technique Limitations

- May not handle nondeterministic faults
  - Difficult to test for race conditions, etc.
  - Long term: put scheduler constraints into the variant representation.
- Assumes bug test case visits different lines than normal test cases
- Assumes existing statements can form repair
  - Current work: repair templates
  - Hand-crafted and mined from CVS repositories
- Slower on large test suites: test case selection

Want to hear more?

ICSE 2009

- Formal algorithm, crossover, mutation
- Representation, parsing, stmt details
- Test cases used
- Sensitivity
- Repair quality
- "Does it work?"

GECCO 2009

- Evolutionary questions
  - nonstandard crossover
  - really evolutionary?
  - operator frequency
- Effect of more test cases
- Scaling behavior
- "Why did it work?"

Conclusions

- We can automatically and efficiently repair certain classes of bugs in off-the-shelf legacy programs.
  - Ten programs totaling 63kloc in about 6 minutes each, on average
- We use regression tests to encode desired behavior.
  - Existing tests encode required behavior
- The genetic programming search focuses attention on parts of the program visited during the bug but not visited during passed test cases.

Questions

- I encourage difficult questions.

Bonus Slide: Test Cases

```
1 #!/bin/sh
2 # Positive Test Case for nullhttpd (POST data)
3 ulimit -t 5
4 /usr/bin/wget --tries=1 --post-data 'name=my_name&submit=submit'
5   "http://localhost:\$PORT/cgi-bin/hello.pl"
6 if diff hello.pl ../known-good-hello.pl-result ; then
7   # if the current output matches the known-good output
8   echo "passed hello.pl test case" >> ../list-of-tests-passed
9 fi
```

Figure 2: Positive test case for `nullhttpd`. `wget` is a command-line HTTP client; `ulimit` cuts the test off after five seconds. The test assumes that the sandboxed webserver is accepting connections on `PORT` and has its own copy of `htdocs`, including `cgi-bin/hello.pl`. Note the oracle comparison using `diff` against `known-good-hello.pl-result` on line 6.

```
1 #!/bin/sh
2 # Negative Test Case for nullhttpd
3 ulimit -t 5
4 ../nullhttpd-exploit -h localhost -p $PORT -t2
5 /usr/bin/wget --tries=1 "http://localhost:$PORT/index.html"
6 if diff index.html ../known-good-index.html-result ; then
7   # if the current output matches the known-good output
8   echo "passed exploit test case" >> ../list-of-tests-passed
9 fi
```

Figure 3: Negative test case for `nullhttpd`. If the exploit (line 4) disables the webserver then the request for `index.html` (line 5) will fail.