KATCH: High-Coverage Testing of Software Patches

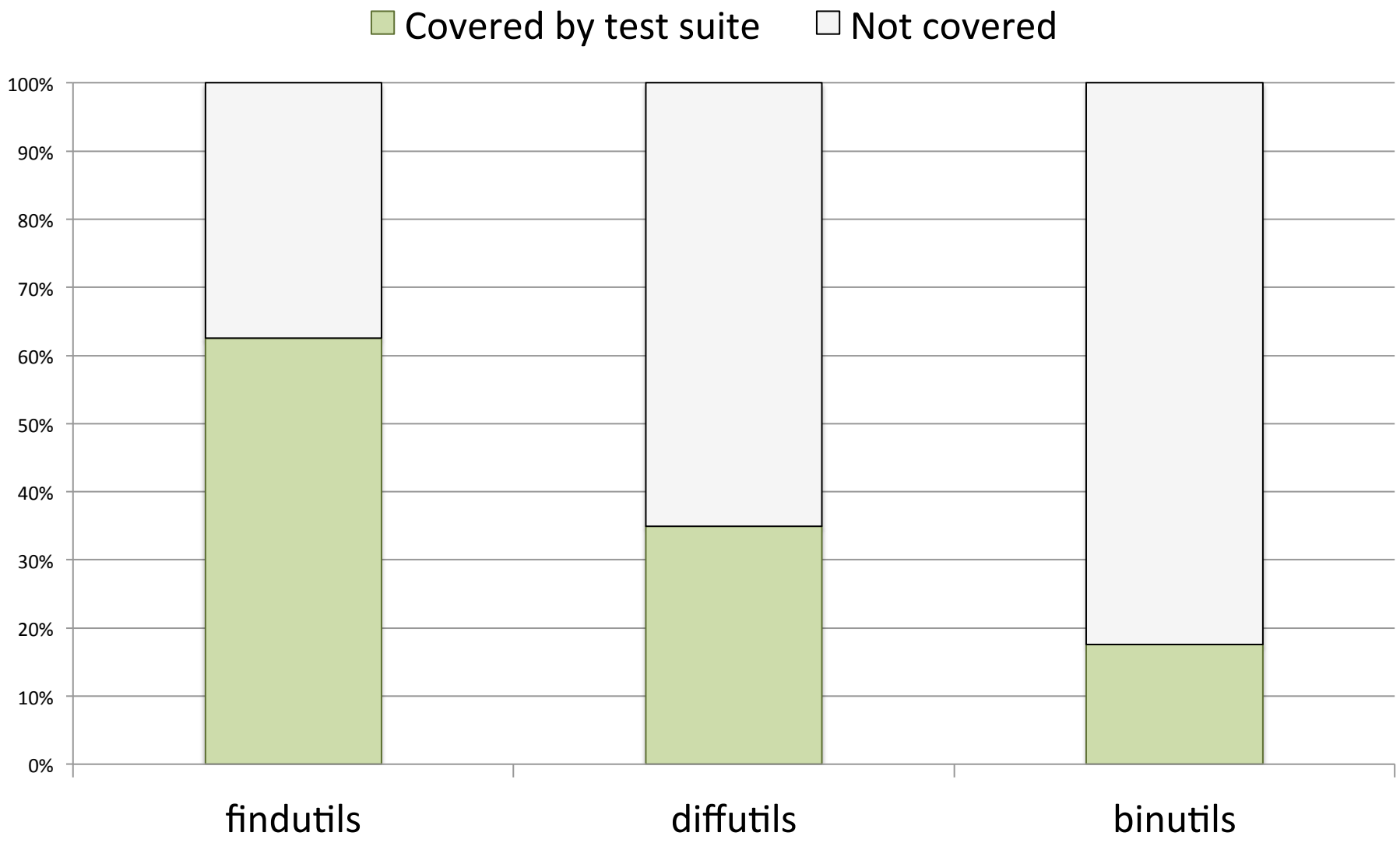
Paul Marinescu and Cristian Cadar

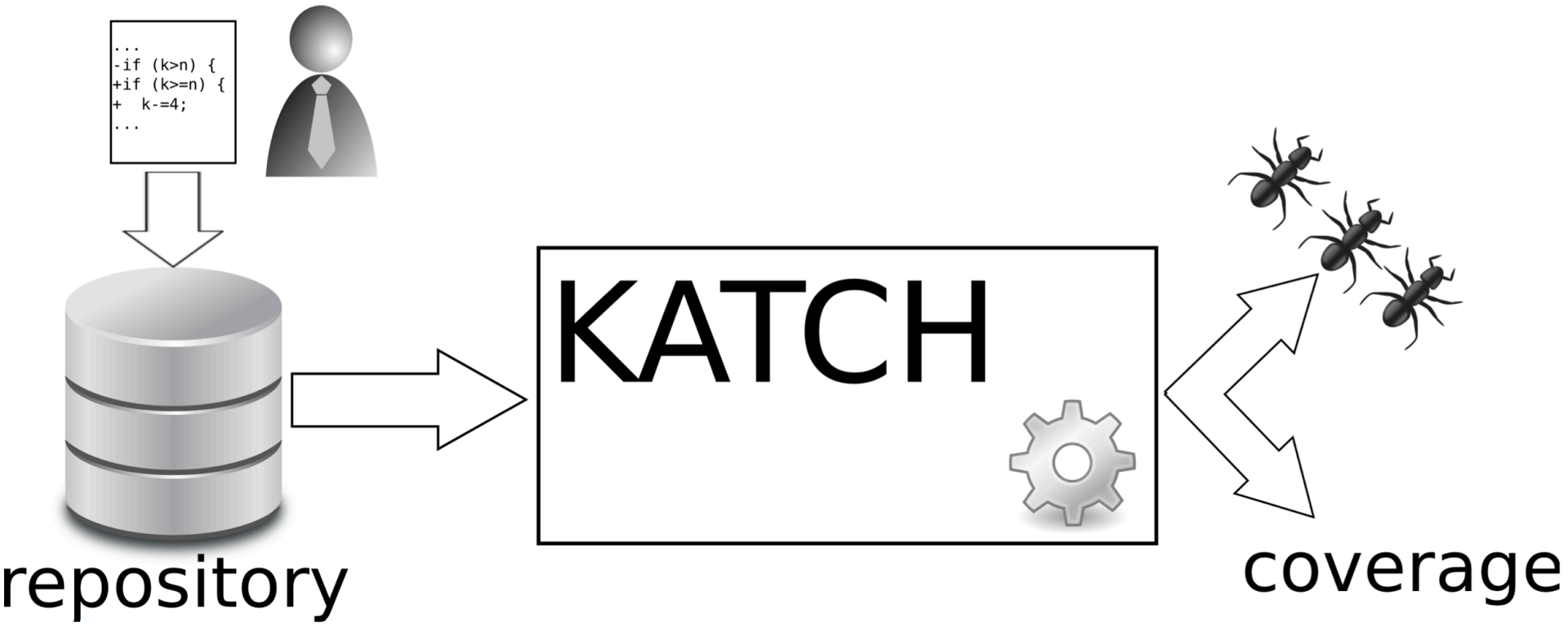
Imperial College London

Intro

- Manual testing is hard
- Supplement it with automatic testing
- We focus on testing software changes

Manual Patch Coverage




...
-if (k>n) {
+if (k>=n) {
+ k-=4;
...
KATCH
repository
coverage

Example: Testing diffutils

```
$ ls diffutils
config.sh  build.sh  regression-test.sh

$ cat config.sh
REPO="git://git.savannah.gnu.org/diffutils.git"
DIFFTARGETS="src lib"
PROGRAMS="src/diff src/diff3 src/sdiff src/cmp"
LIBS="-lrt"

$ katch diffutils 0 100
```

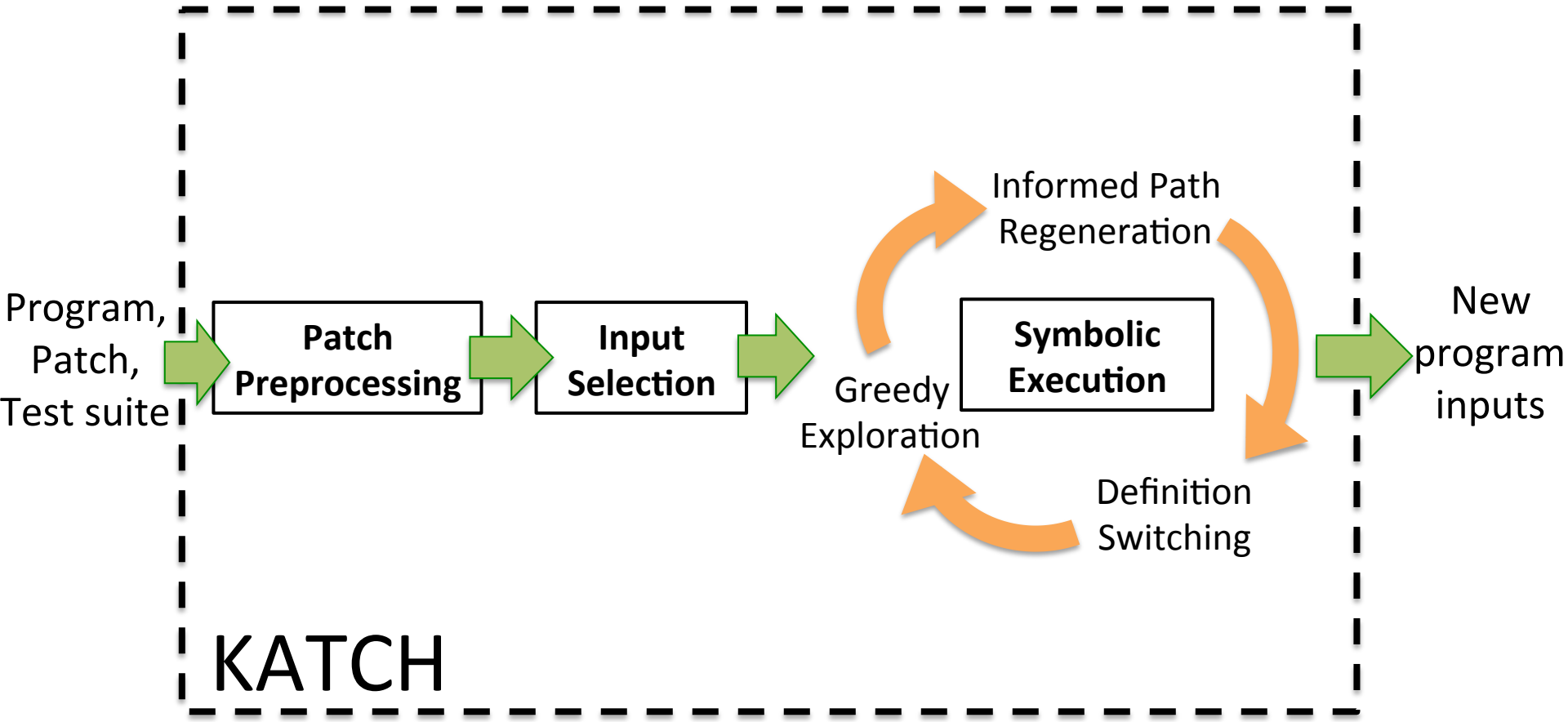
High-Level Idea

- Synthesize inputs which execute the patch code
- Given a program location (e.g. file name, line number), synthesize an input which executes that location

High-Level Approach

- Concrete/Symbolic execution mix + heuristics
- Seeded with existing inputs from the regression test suites

System Overview



Patch Preprocessing

```
Index: src/mod_accesslog.c
===================================================
--- src/mod_accesslog.c   (revision 2659)
+++ src/mod_accesslog.c   (revision 2660)
@@ -156,6 +156,13 @@

void log(char input) {
  int file = open(”access.log”, ...);
+ if (input >= ’ ’ &&
+     input <= ’~’) {
    // printable characters
    write(file, &input, 1);
+ } else {
+   char escinput;
+   escinput = escape(input);
+   write(file, &escinput, 1);
+ }
  close(file);
}
```

TARGET 1
src/mod_accesslog.c:164

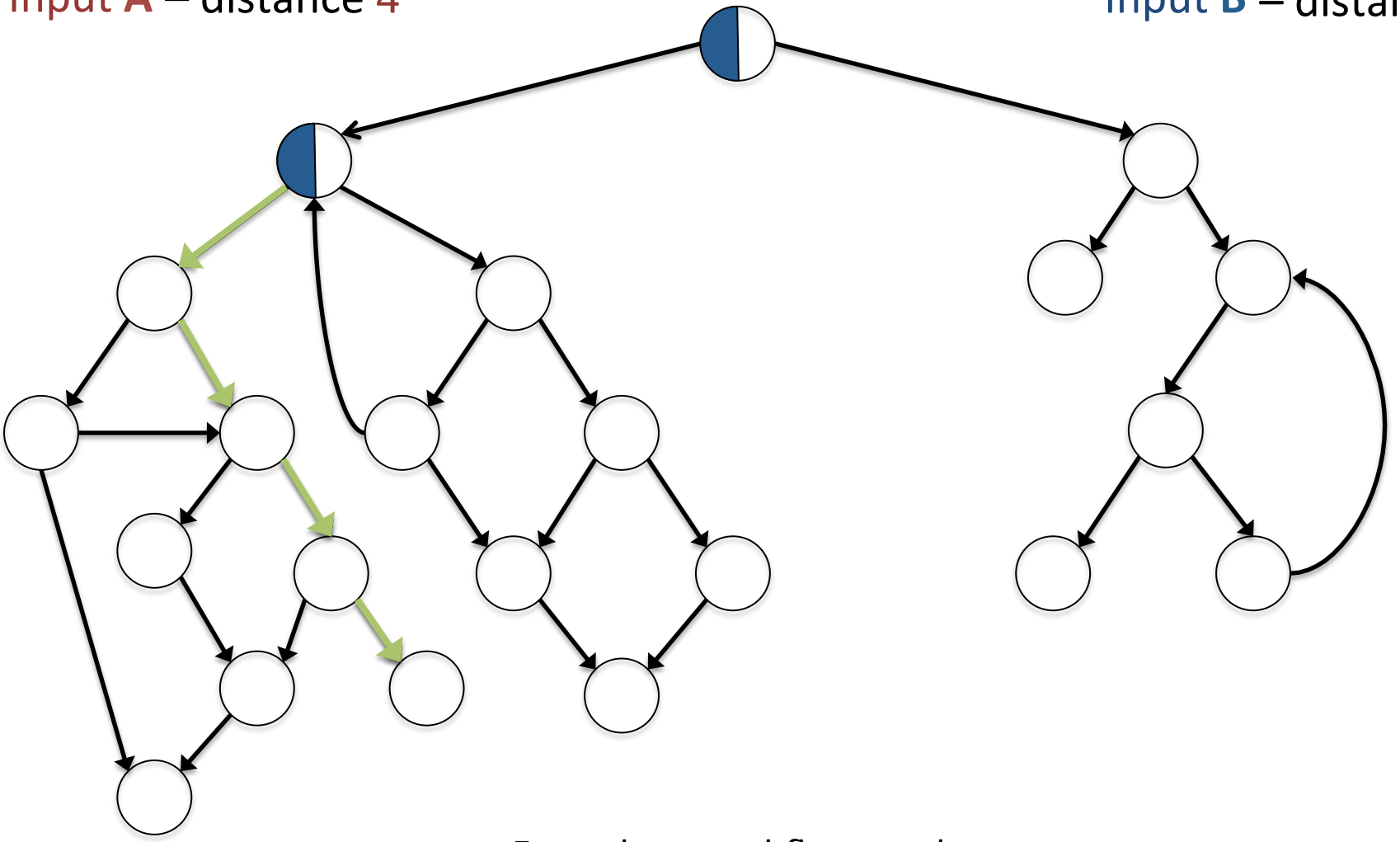
Input Selection

- Rank existing inputs based on how ‘easy’ it is to change them to execute the patch
- Optimization
- Lightweight

Input Selection

Input **A** – distance 4

Input **B** – distance 2



Example control-flow graph

Concrete/Symbolic Execution

- Iterative refinement of the initial input
- Get ‘closer’ to the target at each iteration
- Symbolic execution + path selection heuristics

Greedy Exploration Step

```
void log(char input) {
  int file = open(”access.log”, …);
  if (input >= ’␣’ &&
      input <= ’~’) {
    // printable characters
    write(file, &input, 1);
+ } else {
+   char escinput = escape(input);
+   write(file, &escinput, 1);
+ }
  close(file);
}
```

lighttpd r2660: patch modifies log() to escape sensitive characters

Greedy Exploration Step

```
 void log(char input) {
     int file = open("access.log", …);
     if (input >= '␣' &&
         input <= '~') {
       // printable characters
       write(file, &input, 1);
 + } else {
 +   char escinput = escape(input);
 +   write(file, &escinput, 1);
 + }
     close(file);
 }
```

Available input: "t" (or any printable char)

1. Greedy step: choose the symbolic branch whose unexplored side is closest to the patch.
2. Explore this side!

Informed Path Regeneration

```
void log(char input) {
  if (input >= '␣' &&
        input <= '~') {
    . . .
  } else {
+   . . .
  }
}

if (0 == strcmp(request, "GET")
    . . .
for (char* p = request; *p; p++)
    log(*p);
```

Available input: "GET"

Greedy step fails!

1. Backtrack to the symbolic branch that disallows this side to be executed
2. Explore the other side of that branch

request[2] ≠ 'T'

Definition Switching

```
enum escape_t escape;
void log(char input) {
  if (escape == ESCAPE_ALL) {
+       . . .
  }
}

opt = getopt_long(argc, argv, ...)
switch (opt) {
  case 'a': escape = ESCAPE_SPACE;
  break;
  case 'b': escape = ESCAPE_ALL;
. . .
log(. . .);
```

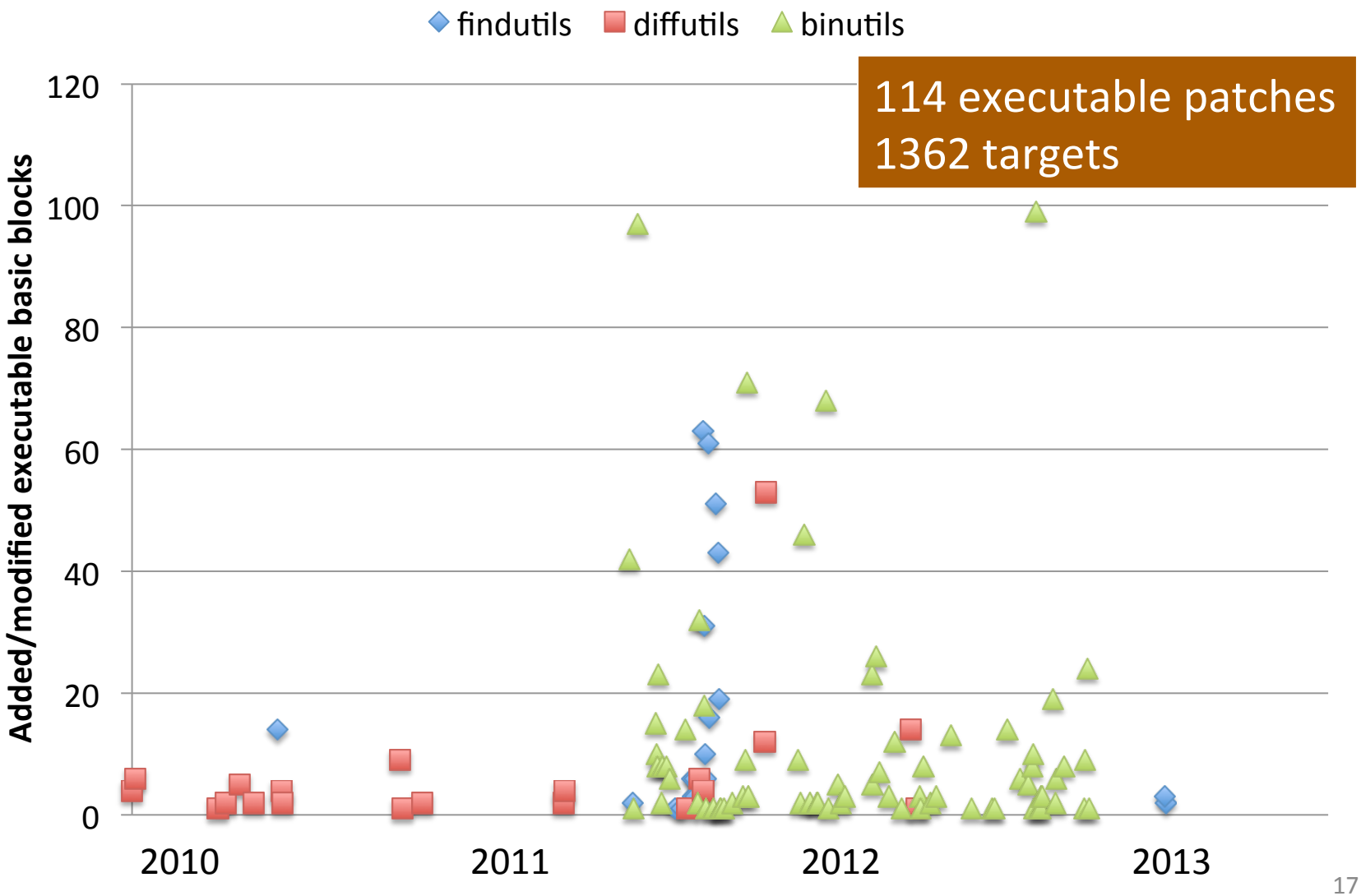
Available test: opt = **'a'**

Patch guarded by concrete branch

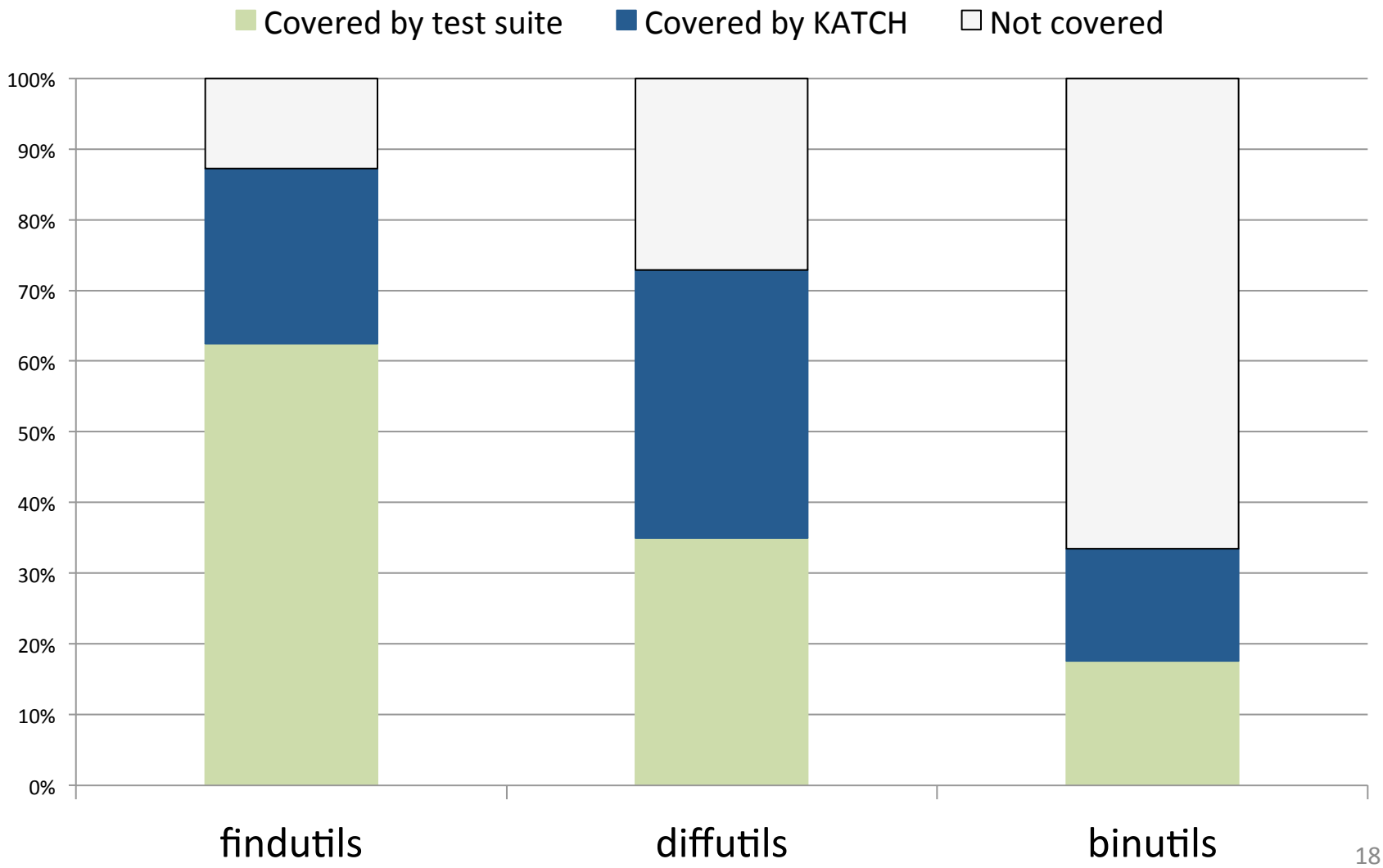
Backtracking step fails!

1. Find all reaching definitions for the variables involved and try to cover another one
2. Favors definitions that can be statically shown to satisfy target, or unexecuted definitions

Evaluation


findutils
diffutils
binutils
114 executable patches
1362 targets
Added/modified executable basic blocks
120
100
80
60
40
20
0
2010
2011
2012
2013

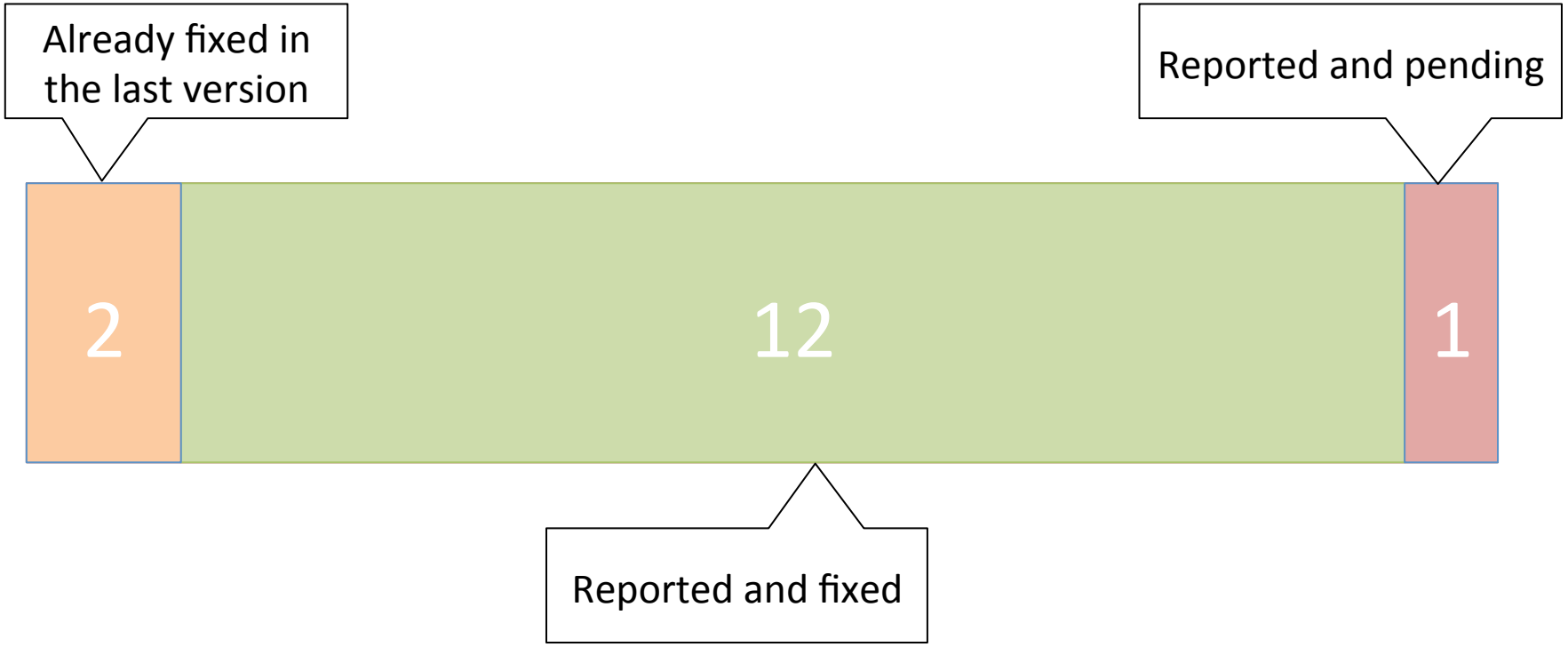
Coverage Improvement



Bugs Found


6 bugs in patch code
15 Crash Bugs
5 bugs close to patch code
4 bugs
unknown
causal relation

Bugs Found


Already fixed in the last version
Reported and pending
2
12
1
Reported and fixed

Automatic Patch Testing

Practical autonomous testing system

Coverage improvement and bug finding

Short artifact* presentation on Friday

http://srg.doc.ic.ac.uk/projects/katch/

*Successfully evaluated by the ESEC/FSE artifact evaluation committee

Selected Related Work

- Directed Test Suite Augmentation (APSEC'09, FSE'10)
- Directed Symbolic Execution (SAS'11)
- Differential Symbolic Execution (FSE'08)
- Directed Incremental Symbolic Execution (PLDI'11)

Heuristic Contribution

Suite	Greedy	Greedy+IPR	Greedy+DS	KATCH
findutils	74	85	78	85
diffutils	25	29	49	63
binutils	70	121	76	135
Total	**169**	**235**	**203**	**283**

IPR = Informed Path Regeneration
DS = Definition Switching