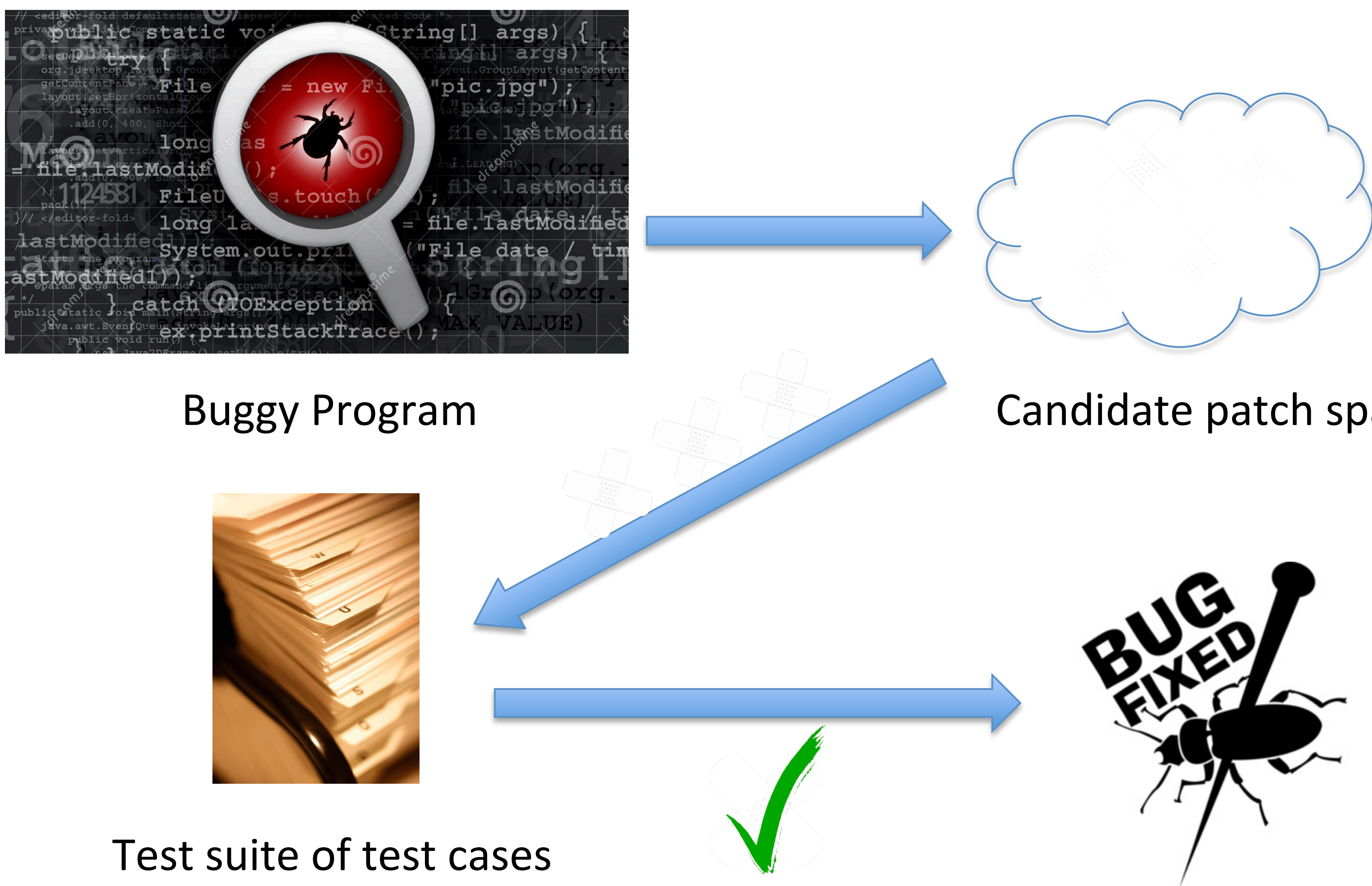
An Analysis of Patch Plausibility and Correctness of Generate-And-Validate Patch Generation System

Zichao Qi, Fan Long, Sara Achour, and Martin Rinard
MIT CSAIL

Generate-And-Validate Patch Generation Systems



Generate-And-Validate Patch Generation Systems

- **GenProg** – Genetic Programming
  1. *C. L. Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each.* ***ICSE 2012***
  2. *W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming.* ***ICSE 2009***
  3. *S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues. A genetic programming approach to automated software repair.* ***GECCO 2009***
  4. *C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. Genprog: A generic method for automatic software repair.* ***Software Engineering, IEEE Transactions on 38(1), 2012***
- **AE** – Adaptive Search
  1. *W. Weimer, Z. P. Fry, and S. Forrest. Leveraging program equivalence for adaptive program repair: Models and first results.* ***ASE 2013***
- **RSRepair** – Random Search
  1. *Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang. The strength of random search on automated program repair.* ***ICSE 2014***

All of them report impressive results

	GenProg	AE	RSRepair
Benchmark Defects	**105**	**105**	**24**
Reported Fixed Defects	**55**	**54**	**24**

- Patches generated by these systems are different from human written patch
- No systematic analysis

We analyze the reported patches for these systems

Plausible?
Produce correct outputs for all test cases in the test suite

All generated patches should be plausible

Plausibility

	GenProg	AE	RSRepair
Benchmark Defects	**105**	**105**	**24**
Reported Fixed Defects	**55**	**54**	**24**
Defects With Plausible Patches	**18**	**27**	**10**

- Reason - Weak Proxy
  - Patch evaluation does not check for correct output
  - php, libtiff – check exit code, *not output*
  - Accepted php patch: `main(){ exit(0); }`

Analysis of the reported patches for these systems

Plausible?
Produce correct outputs for all test cases in the test suite

Majority of the patches are not plausible

Correct?
Eliminate the defect

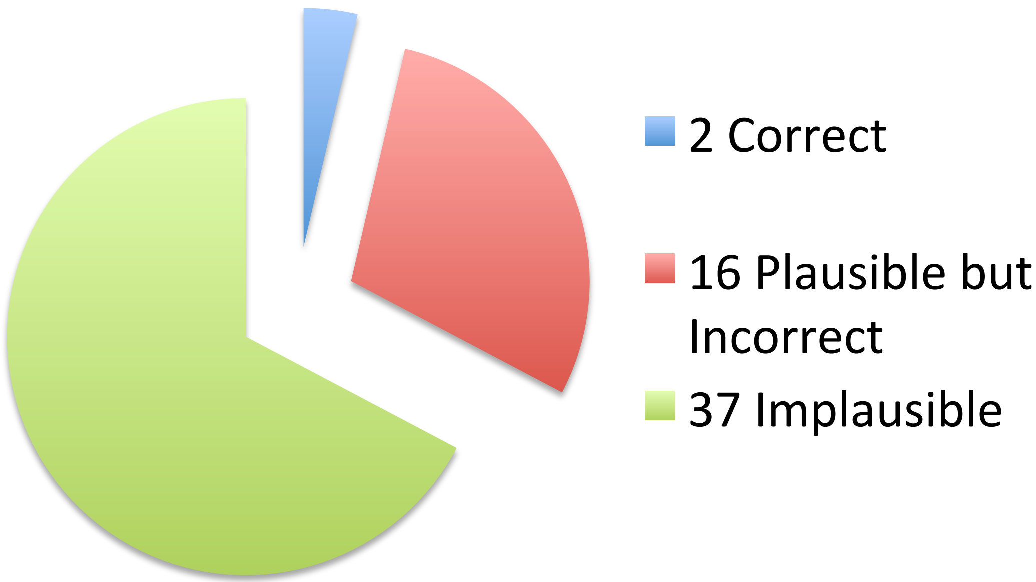
Passing test suite != correctness

Correctness

	GenProg	AE	RSRepair
Benchmark Defects	**105**	**105**	**24**
Reported Fixed Defects	**55**	**54**	**24**
Defects With Plausible Patches	**18**	**27**	**10**
Defects With Correct Patches	**2**	**3**	**2**

Developed new test cases that expose defects for all plausible but incorrect patches

GenProg Statistics



Stronger Test Suites?

Will GenProg generate correct patches given new test cases that eliminate incorrect patches?

Fixed Test Scripts?

Will GenProg generate plausible patches given fixed patch evaluation scripts?

Analysis of the reported patches for these systems

Plausible?
Produce correct outputs for all test cases in the test suite

Majority of the patches are not plausible

Correct?
Eliminate the defect

The overwhelming majority of the patches are not correct

Do stronger test suites help?

Rerun GenProg with fixed patch evaluation scripts and new test cases that eliminate incorrect patches

Reexecution of GenProg on Remaining 103 Defects

First Reexecution
Fixed patch evaluation
New test cases
Patches for 2 defects

Second Reexecution
2 additional test cases
Patches for 0 defects

Why?

- Developer patches are not in GenProg search space
- GenProg search space may not contain ***any*** correct patch for these 103 defects
- May need richer search space to generate correct patches

Bottom Line For GenProg

- Rerun GenProg with
  - Fixed test scripts
  - Stronger test suites
- GenProg generates patches for only 2 of 105 defects (both patches are correct)

Examples of Correct GenProg Patch(1/2)

Developer

```
-if (y < 1000) {
-     PyObject *accept = PyDict_GetItemString(moddict,
-                                              "accept2dy
-     if (accept != NULL) {
-          int acceptval =  PyObject_IsTrue(accept);
-          if (acceptval == -1)
-               return 0;
-          if (acceptval) {
-               if (0 <= y && y < 69)
-                    y += 2000;
-               else if (69 <= y && y < 100)
-                    y += 1900;
-               else {
-                    PyErr_SetString(PyExc_ValueError,
-                                     "year out of range");
-                    return 0;
-               }
-               if (PyErr_WarnEx(PyExc_DeprecationWarning,
-                    "Century info guessed for a 2-digit ye
-                    return 0;
-          }
-     }
-     else
-          return 0;
-}
  p->tm_year = y - 1900;
  p->tm_mon--;
  p->tm_wday = (p->tm_wday + 1) % 7;
```

GenProg

```
- if (y < 1000) {
-    tmp___0 = PyDict_GetItemString(moddict, "accept2dyear");
-    accept = tmp___0;
-    if ((unsigned int )accept != (unsigned int )((void *)0)) {
-      tmp___1 = PyObject_IsTrue(accept);
-      acceptval = tmp___1;
-      if (acceptval == -1) {
-        return (0);
-      } else {

-      }

-      if (acceptval) {
-        if (0 <= y) {
-          if (y < 69) {
-            y += 2000;
-          } else {
-            goto _L;
-          }
-        } else {
-          _L: /* CIL Label */
-          if (69 <= y) {
-            if (y < 100) {
-              y += 1900;
-            } else {
-              PyErr_SetString(PyExc_ValueError,
                                "year out of range");
-              return (0);
-            }
-          } else {
-            PyErr_SetString(PyExc_ValueError,
                              "year out of range");
-            return (0);
-          }
-        }

-        tmp___2 = PyErr_WarnEx(PyExc_DeprecationWarning,
-                    "Century info guessed for a 2-digit year.", 1);
-        if (tmp___2 != 0) {
-          return (0);
-        } else {

-        }
-      } else {

-      }
-    } else {
-      return (0);
-    }
- } else {

- }
  p->tm_year = y - 1900;
  (p->tm_mon) --;
  p->tm_wday = (p->tm_wday + 1) % 7;
```

Examples of Correct GenProg Patch(2/2)

Developer

```
if (offset >= s1_len) {
        php_error_docref(NULL TSRMLS
        "The start position cannot e
        RETURN_FALSE;
}

-if (len > s1_len - offset) {
-        len = s1_len - offset;
-}

cmp_len = (uint) (len ? len : MAX(s2
```

GenProg

```
if (offset >= (long )s1_len) {
    php_error_docref0((char const   *)((void
    "The start position cannot exceed initia
    while (1) {
      __z___1 = return_value;
      __z___1->value.lval = 0L;
      __z___1->type = (unsigned char)3;
      break;
    }
    return;
  } else {

  }
- if (len > (long )s1_len - offset) {
-   len = (long )s1_len - offset;
- } else {

- }
  if (len) {
    tmp___1 = len;
  } else {
    if ((long )s2_len > (long )s1_len - offs
      tmp___0 = (long )s2_len;
    } else {
      tmp___0 = (long )s1_len - offset;
    }
    tmp___1 = tmp___0;
  }
  cmp_len = (unsigned int )tmp___1;
```

All Correct Patches Simply Delete Code

Semantic Analysis

- Analyze all the plausible patches
- Determine if patch is equivalent to single functionality deletion modification
- Results
  - GenProg: 14/18
  - AE: 22/27
  - RSRepair: 8/10

We found a common scenario

- A negative test case exposes the defect
  - Feature is otherwise unexercised
  - The patch simply deletes the functionality
    - Introduces new security vulnerabilities (buffer overflows)
    - Disables critical functionality (gzip cannot decompress non-zero files)

- Weak test suites
  - May be appropriate for human developers
  - *May not be* appropriate for automatic patch generation systems (at least not by themselves)

If all these patches simply delete functionality

Why not build a patch generation system that ONLY deletes functionality?

We present Kali

- Automatic patch generation system
- Consider the search space that consists of only patches that remove functionality

Experimental Results of Kali

	GenProg	AE	RSRepair	Kali
Benchmark Defects	**105**	**105**	**24**	**105**
Reported Fixed Defects	**55**	**54**	**24**	
Defects With Plausible Patches	**18**	**27**	**10**	**27**
Defects With Correct Patches	**2**	**3**	**2**	**3**

- Kali is as good as previous systems
  - Much simpler
  - Not need to know the source code file to repair
- Can pinpoint the defective code
- Can provide insight into important defect characteristics.

Experimental Results of Kali

	GenProg	AE	RSRepair	Kali
Benchmark Defects	**105**	**105**	**24**	**105**
Reported Fixed Defects	**55**	**54**	**24**	
Defects With Plausible Patches	**18**	**27**	**10**	**27**
Defects With Correct Patches	**2**	**3**	**2**	**3**

Is Automatic Patch Generation A Total Failure?

NO!

Path To Success

- Richer search spaces
- More efficient search algorithms
- Incorporate additional sources of information
  – Correct code from other applications
  – Learned characteristics of human patches
  – Learned invariants
  – Specifications

Promising directions

- Learn invariant from correct execution
  - ClearView: *J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, et al. Automatically patching errors in deployed software. SOSP 2009.*
  - Patches security vulnerabilities in 9 of 10 defects
  - At least 4 patches are correct
- Solvers
  - NOPOL: F. *DeMarco, J. Xuan, D. Le Berre, and M. Monperrus. Automatic repair of buggy if conditions and missing preconditions with smt. CSTVA 2014*
  - SemFix: *H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra. Semfix: Program repair via semantic analysis. ICSE 2013*

Promising directions

- Specifications
  - Autofix-E: *Yu Pei, Carlo A. Furia, Martin Nordio, Yi Wei, Andreas Zeller, and Bertrand Meyer. Automated Fixing of Programs with Contracts. IEEE Transactions on Software Engineering, 2014.*
  - *Etienne Kneuss, Manos Koukoutos and Viktor Kuncak. Deductive Program Repair. CAV 2015*
- Correctness evaluation
  - *Thomas Durieux, Matias Martinez, Martin Monperrus, Romain Sommerard, Jifeng Xuan. Automatic Repair of Real Bugs: An Experience Report on the Defects4J Dataset. Technical report 1505.07002, Arxiv, 2015*
- Code from another application
  - CodePhage: *S. Sidiroglou, E. Lahtinen, F. Long, and M. Rinard. Automatic error elimination by multi-application code transfer. PLDI 2015*

Promising Directions

	SPR Staged condition synthesis	**Prophet** Learn from successful patches
Benchmark Defects	**105**	**105**
Defects With Plausible Patches	**41**	**42**
Defects With Correct Patches	**11**	**15**

SPR: *F. Long and M. Rinard. Staged program repair in SPR. To appear in ESEC-FSE 2015*

Prophet: *F. Long and M. Rinard. Prophet: Automatic patch generation via learning from successful human patches. Under submission*

Take Aways

- Facts about GenProg/AE/RSRepair
  - These systems fix 2/3/2 of 105 bugs (not 55/54/24)
  - Errors in test scripts and weak test suites
  - Fixed test scripts and stronger test suites do not help
- Paths to success
  - Richer search spaces
  - More efficient search algorithms
  - Incorporate additional sources of information
    - Correct code from other applications (CodePhage)
    - Learned characteristics of human patches (Prophet)
    - Learned invariants (ClearView)
    - Specifications (AutoFixE, Deductive Repair)

Summary

- Evaluation of GenProg, AE and RSRepair
  - Incorrect results
  - Equivalent to functionality elimination
  - Stronger test suites do not help
- Kali
  - Functionality elimination system
  - Help developer better understand the bug

Path to Success for the automatic patch generation systems

- Richer search spaces
- More efficient search algorithms
- Incorporate additional sources of information
  - Correct code from other applications
  - Learned characteristics of human patches
  - Learned invariants
- Better patch evaluation
  - Correctness
  - Understand the negative effects

Questions?