Concolic Program Repair

Yannic Noller | Research Talk
(Automated) Program Repair

Input → (Buggy) Program → Output

- Output: unexpected behavior → bug detected
- Validation
- Program Modification
- Fault / Fix Localization

How to resolve?

Can be very tedious ... and time consuming

→ Automated program repair

yannic.noller@nus.edu.sg
State of the Art

Challenges

- How to provide **high quality but few** patches?
- How to avoid **non-sensical** patches?
- How to produce **less overfitting** patches?
- How to repair bugs in the **absence** of many test cases?
Challenges

Other low-quality patches:

```c
if (((! (image->res_unit == 3)) && (! (image->res_unit == 3))))
if ( (! ((log_level && (! ((- 4) == 0))) && log_level)))
```
Concolic Program Repair
Ruben Shafieefdeen
National University of Singapore
ruben@comp.nus.edu.sg

Yannic Noller
National University of Singapore
yannic.noller@acm.org

Lars Grunske
Humboldt-Universität zu Berlin
Germany
grunske@informatik.hu-berlin.de

Abhik Roychoudhury
National University of Singapore
Singapore
abhik@comp.nus.edu.sg

Abstract
Automated program repair reduces the manual effort in fixing program errors. However, existing repair techniques modify a buggy program such that it passes given tests. Such repair techniques do not facilitate between correct patches and patches that overfit the available tests (breaking unintended but desired functionality). We propose an integrated approach for detecting and discarding overfitting patches via systematic co-exploration of the patch space and input space. We leverage concolic path exploration to systematically traverse the input space (and generate inputs), while ruling out significant parts of the patch space. Given a long enough time budget, this approach allows a significant reduction in the set of patches candidates, as shown by our experiments. We implemented our technique in the form of a tool called CPR and evaluated its efficacy in reducing the patch space by discarding overfitting patches from a pool of plausible patches. We evaluated our approach for fixing real-world software vulnerabilities and defects, for fixing functionality errors in programs drawn from SV-COMP benchmarks used in software verification, as well as for test-suite guided repair. In our experiments, we observed a patch space reduction due to our concolic exploration of up to 74% for fixing software vulnerabilities and up to 63% for SV-COMP programs. Our technique presents the viewpoint of a repair technique for detecting and discarding overfitting patches because they repair the failing test case(s), while not fixing the test data. Such overfitting patches are called plausible patches because they repair the failing test case(s) without fixing the test data. Such overfitting repair provides a practical formulation of the program repair problem, it gives rise to the phenomenon of “overfitting” [30]. The patched program may pass the tests in the given test-suite while failing tests outside the test-suite, thereby overfitting the test data. Such overfitting patches are called plausible patches because they repair the failing test case(s), but they are not guaranteed to be correct, since they may fail tests outside the test-suite guiding the repair. Various solutions to alleviate the patch overfitting issue have been studied to date, including symbolic specification inference [18, 29], machine learning-based prioritization of patches [2, 23, 24], and fuzzing based test-suite augmentation [17]. These works do not guarantee any notion of correctness of the patches, and cannot guarantee even the most basic correctness criteria such as crash freedom.

1 Introduction
Automated Program Repair [18, 26] is an emerging technology which seeks to rectify errors or vulnerabilities in software automatically. There are various applications of automated repair, including improving programmer productivity, reducing exposure to software security vulnerabilities, producing self-healing software systems, and even enabling intelligent tutoring systems for teaching programming.

Since program repair needs to be guided by certain notions of correctness and formal specifications of the program’s behavior are usually not available, it is common to use test-suites to guide repair. The goal of automated repair is then to produce a (minimal) modification of the program so as to pass the tests in the given test-suite. While test-suite driven repair provides a practical formulation of the program repair problem, it gives rise to the phenomenon of “overfitting” [30]. The patched program may pass the tests in the given test-suite while failing tests outside the test-suite, thereby overfitting the test data. Such overfitting patches are called plausible patches because they repair the failing test case(s), but they are not guaranteed to be correct, since they may fail tests outside the test-suite guiding the repair. Various solutions to alleviate the patch overfitting issue have been studied to date, including symbolic specification inference [18, 29], machine learning-based prioritization of patches [2, 23, 24], and fuzzing based test-suite augmentation [17]. These works do not guarantee any notion of correctness of the patches, and cannot guarantee even the most basic correctness criteria such as crash freedom.

In this work, we reflect on the problem of patch overfitting [22, 26, 30], in an attempt to produce patches which work...
Our Approach

**semantic approach incl. program synthesis**
- avoids non- compilable patches
- provides symbolic reasoning capabilities

**co-exploration of the input space and patch space**
- prune overfitting patches
- enables gradual improvement

**user-provided specification**
- to reason about additional inputs
- key aspect to handle absence of test cases

---

### Input Space
- initial test case
- explored path (input partition)

### Patch Space
- plausible patches
- correct patch (set)
- refined patch set
- infeasibility checks in both directions
- represented with abstract patches

---

**Concolic Program Repair**

plausible patches
Inputs to Concolic Program Repair

### Research Problem
- **State of the Art**
- **Our Solution**
- **Example**
- **Evaluation**
- **Summary**

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buggy Program</td>
</tr>
<tr>
<td>Failing test case(s)</td>
</tr>
<tr>
<td>Fix Locations</td>
</tr>
<tr>
<td>User Specification</td>
</tr>
</tbody>
</table>

**CVE-2016-3623:** Divide by Zero in LibTIFF v4.0.6

```c
static int cvtRaster(TIFF* tif, uint32* raster, uint32 width, uint32 height) {
    uint32 y;
tstrip_t strip = 0;
tsize_t cc, acc;
unsigned char* buf;
uint32 rwidth = roundup(width, horizSubSampling);
uint32 rheight = roundup(height, vertSubSampling);
uint32 nrows = (rowsperstrip > rheight ? rheight : rowsperstrip);
uint32 rrrows = roundup(nrows, vertSubSampling);
if (CONDITION) return 0;
/* potential divide-by-zero error */
cc = rrrows*rwidth + 2 * ((rrrows*rwidth)
    / (horizSubSampling*vertSubSampling));
```

---

**Example**

- **Observation**
  - `observation`

- **Input**
  - `if (CONDITION) return 0;`

- **User Specification**
  - `formula about correct behavior in SMT format`

- **Fix Locations**
  - `source location, (fix template), synthesis components`

- **Buggy Program**
  - `e.g., exploit as TIFF picture`

- **Failing test case(s)**
  - `Input`
Workflow

Input
- Buggy Program
- Failing test case(s)
- Fix Locations
- User Specification

Concolic Program Repair
- Patch Pool Construction
- Input Generation
- Concolic Execution
- Patch Reduction
  - new input
  - reduced patch pool
  - path constraint $\phi$

Output
- Ranked Patches

Program synthesis

Independent from any test suite

Path exploration

Anytime algorithm (gradual improvement)

Refinement based on explored paths and specification
Patch Representation

.. **concrete** patches

\[
\begin{align*}
    x &> 0 \\
    x &> 1 \\
    x &> 2 \\
    \ldots
\end{align*}
\]

\[
\begin{align*}
    x + 1 &> y \\
    x - 1 &> y \\
    x + 2 &> y \\
    \ldots
\end{align*}
\]

.. **abstract** patches

\[
\begin{align*}
    x &> a, \ a \in [0, 10] \\
    x + a &> y, \ a \in [-10, 10]
\end{align*}
\]

Our notion of an **abstract patch** represents a **patch template** with **parameters**.

- **generate** and **maintain** smaller amount of patch candidates
- allows refinement instead of just discarding
- **subsumes** concrete patches
Abstract Patches

\((\theta_\rho, T_\rho, \psi_\rho)\)

- \(X_\rho\) is the set of program variables
- \(X \subseteq X_\rho\) is the set of input variables
- \(A\) is the set of template parameters

- \(\theta_\rho(X_\rho, A)\) denotes the repaired (boolean or integer) expression

- \(T_\rho(A)\) represents the conjunction of constraints \(\tau_\rho(a_i)\) on the parameters \(a_i \in A\) included in \(\theta_\rho: T_\rho(A) = \bigwedge_{a_i \in A} \tau_\rho(a_i)\)

- \(\psi_\rho(X, A)\) is the patch formula induced by inserting the expression \(\theta_\rho\) into the buggy program

Examples

1. patch is a condition

\[
\begin{align*}
\theta_\rho &:= x > a \\
T_\rho &:= \tau_\rho(a) := (a \geq -10) \\
\psi_\rho &:= x > a 
\end{align*}
\]

2. patch is a right hand-side of an assignment

\[
\begin{align*}
\theta_\rho &:= x - a \\
T_\rho &:= \tau_\rho(a) := (a \geq -10) \\
\psi_\rho &:= (y = x - a)
\end{align*}
\]
Infeasability checks

.. in the input space

Path Reduction:
For every generated input, we check that there is one patch that can exercise the corresponding path. Otherwise, the path will not be explored.

For example:
\[
\phi := x > 3 \land y > 5 \land \rho \\
\rho := (x = 0 \lor y = 0)
\]

.. in the patch space

Patch Reduction:
If a patch allows inputs to exercise a path that violates the specification, we identify this as a patch that overfits the valid set of values and attempt to refine it.

For example:
\[
\forall a_1, a_2, \ldots, a_n \forall x_1, x_2, \ldots, x_m : \\
\phi(X) \land \psi_{\rho}(X, A) \land T_{\rho}(A) \implies \sigma(X)
\]
# Patch Refinement

What we **want to have:**

\[ \forall a_1, a_2, \ldots, a_n \forall x_1, x_2, \ldots, x_m : \phi(X) \land \psi_\rho(X, A) \land T_\rho(A) \implies \sigma(X) \]

What we **are checking for:**

\[ \neg (\forall a_1, a_2, \ldots, a_n \forall x_1, x_2, \ldots, x_m : \phi(X) \land \psi_\rho(X, A) \land T_\rho(A) \implies \sigma(X)) \]

\[ \equiv \neg (\forall a_1, a_2, \ldots, a_n \forall x_1, x_2, \ldots, x_m : \neg (\phi(X) \land \psi_\rho(X, A) \land T_\rho(A)) \lor \sigma(X)) \]

\[ \equiv \exists a_1, a_2, \ldots, a_n \exists x_1, x_2, \ldots, x_m : \phi(X) \land \psi_\rho(X, A) \land T_\rho(A) \land \neg \sigma(X) \]

→ use **SMT solver** to retrieve a model \( \mathcal{M} \) to **refine** the parameter constraint
Example

```
......
static int
cvtRaster(TIFF* tif, uint32* raster, uint32 width, uint32 height)
{
  uint32 y;
  tstrip_t strip = 0;
  tsize_t cc, acc;
  unsigned char* buf;
  uint32 rwidth = roundup(width, horizSubSampling);
  uint32 rheight = roundup(height, vertSubSampling);
  uint32 nrows = (rowsperstrip > rheight ?
                 rheight : rowsperstrip);
  uint32 rnrows = roundup(nrows,vertSubSampling);
  if (CONDITION) return 0;
  /* potential divide-by-zero error */
  cc = rnrows*rwidth + 2 * ((rnrows*rwidth)
                  / (horizSubSampling*vertSubSampling));
  ......
}
```

CVE-2016-3623: Divide by Zero in LibTIFF v4.0.6

\[ x \triangleq \text{horizSubSampling} \]
\[ y \triangleq \text{vertSubSampling} \]
Example (2)

Input Space

Initial test input
x=7, y=0

Patch Space

plausible patches

Patch Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x &gt;= a</td>
<td>a &gt;= -10 ∧ a &lt;= 7</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>y &lt; b</td>
<td>b &gt;= 1 ∧ b &lt;= 10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>x == a</td>
<td></td>
<td>y == b</td>
</tr>
</tbody>
</table>

x △ horizSubSampling
y △ vertSubSampling
C △ CONDITION

I

II

P1: x > 3 ∧ y ≤ 5 ∧ ¬C
**Example (2) - Patch 1**

\[ \exists a_1, a_2, \ldots, a_n \exists x_1, x_2, \ldots, x_m : \]

\[ \phi(X) \land \psi_\rho(X, A) \land T_\rho(A) \land \neg \sigma(X) \]

\[ x > 3 \land y \leq 5 \]

\[ \land \neg(x \geq a) \land a \in [-10, 7] \]

\[ \land (x \ast y = 0) \]

- path constraint P1
- patch 1
- condition specification violation

### ID | Patch Template | Parameter Constraint | # Conc. Patches
--- | --- | --- | ---
1 | x >= a | a >= -10 \land a <= 7 | 18
2 | y < b | b >= 1 \land b <= 10 | 10
3 | x == a || y == b | (a=7 \land b >= -10 \land b <= 10) \lor (b=0 \land a >= -10 \land a <= 10) | 41

### ID | Patch Template | Parameter Constraint | # Conc. Patches
--- | --- | --- | ---
1 | x >= a | a >= -10 \land a <= 4 | 15

**CVE-2016-3623**: Divide by Zero in LibTIFF v4.0.6!

*E.g., exploit as TIFF picture source location, (fix template), synthesis components formula about correct behavior in SMT format*

**Input**

**Failing test case(s)**

**Fix Locations**

**Buggy Program**

```c
static int cvtRaster(TIFF* tif, uint32_t raster, uint32_t width, uint32_t height)
{
    uint32_t y;
    tstrip_t strip = 0;
tsize_t cc, acc;
unsigned char * buf;
    uint32_t rwidth = roundup(width, horizSubSampling);
    uint32_t rheight = roundup(height, vertSubSampling);
    uint32_t nrows = (rowsperstrip > rheight ? rheight : rowsperstrip);
    uint32_t rnrows = roundup(nrows, vertSubSampling);
    if (CONDITION) return 0;
    /* potential divide-by-zero error */
    cc = rnrows * rwidth + 2 * ((rnrows * rwidth) / (horizSubSampling * vertSubSampling));
    ....
}
```

**User Specification**

...
Example (2) - Patch 2

\[ \exists a_1, a_2, \ldots, a_n \exists x_1, x_2, \ldots, x_m : \]
\[ \phi(X) \land \psi(X, A) \land T_p(A) \land \neg \sigma(X) \]

\[ x > 3 \land y \leq 5 \]
\[ \land \neg (y < b) \land b \in [1, 10] \]
\[ \land (x \times y = 0) \]

Path constraint P1

Patch 2

Condition specification violation

ID | Patch Template | Parameter Constraint | # Conc. Patches
---|----------------|-----------------------|-----------------|
1  | \( x \geq a \) | \( a \geq -10 \land a \leq 7 \) | 18
2  | \( y < b \) | \( b \geq 1 \land b \leq 10 \) | 10
3  | \( x \equiv a \lor y \equiv b \) | \((a=7 \land b \geq -10 \land b \leq 10) \lor (b=0 \land a \geq -10 \land a \leq 10)\) | 41

ID | Patch Template | Parameter Constraint | # Conc. Patches
---|----------------|-----------------------|-----------------|
1  | \( x \geq a \) | \( a \geq -10 \land a \leq 4 \) | 15
2  | \( y < b \) | \( b \geq 1 \land b \leq 10 \) | 10
Example (2) - Patch 3

\[ \exists a_1, a_2, \ldots, a_n \exists x_1, x_2, \ldots, x_m : \phi(X) \land \psi_\rho(X, A) \land T_\rho(A) \land \neg \sigma(X) \]

\[ x > 3 \land y \leq 5 \]

path constraint P1

\[ \land \neg (x = a \lor y = b) \]

patch 3

\[ \land (a = 7 \land b \in [-10, 10] \land b = 0 \land a \in [-10, 10]) \land (x \ast y = 0) \]

condition specification violation

Patch Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x &gt;= a</td>
<td>a &gt;= -10 \land a &lt;= 7</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>y &lt; b</td>
<td>b &gt;= 1 \land b &lt;= 10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>x == a \lor y == b</td>
<td>(a = 7 \land b &gt;= -10 \land b &lt;= 10) \lor (b = 0 \land a &gt;= -10 \land a &lt;= 10)</td>
<td>41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x &gt;= a</td>
<td>a &gt;= -10 \land a &lt;= 4</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>y &lt; b</td>
<td>b &gt;= 1 \land b &lt;= 10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>x == a \lor y == b</td>
<td>b = 0 \land a &gt;= -10 \land a &lt;= 10</td>
<td>21</td>
</tr>
</tbody>
</table>
**Example (2)**

### Input Space

- **I**:
  - Initial test input: $x=7$, $y=0$
  - $P_3$, $P_2$, $P_4$
  - $P_1$: $x > 3 \land y \leq 5 \land \neg C$

- **II**:
  - $P_3$, $P_2$, $P_4$
  - $P_1$: $x \geq a$ , $y$ , $x = a \lor y = b$

### Patch Space

- **I**: $P_3$, $P_2$, $P_4$
  - Plausible patches
  - Correct patch (set)

- **II**: $P_3$, $P_2$, $P_4$
  - Plausible patches

### Patch Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x \geq a$</td>
<td>$a \geq -10 \land a \leq 7$</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>$y &lt; b$</td>
<td>$b \geq 1 \land b \leq 10$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$x = a \lor y = b$</td>
<td>$(a=7 \land b \geq -10 \land b \leq 10) \lor (b=0 \land a \geq -10 \land a \leq 10)$</td>
<td>41</td>
</tr>
</tbody>
</table>

**Example**

```
uint32 rows = roundup(prows, vertSubSampling);
uint32 cols = roundup(pcols, horizSubSampling);

if (CONDITION) return 0;
/* potential divide-by-zero error */
```

**CVE-2016-3623**: Divide by Zero in LibTIFF v4.0.6

- E.g., exploit as TIFF picture source location, (fix template), synthesis components
- Formula about correct behavior in SMT format

---

`x ≜ horizSubSampling`
`y ≜ vertSubSampling`
`C ≜ CONDITION`
Example (3)

**Input Space**

**Patch Space**

**Patch Details**

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x &gt;= a</td>
<td>a ≥ -10 ∧ a ≤ 4</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>y &lt; b</td>
<td>b ≥ 1 ∧ b ≤ 10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>x == a</td>
<td></td>
<td>y == b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x &gt; 3 ∧ y ≤ 5 ∧ ¬C</td>
<td>a ≥ -10 ∧ a ≤ 0</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>x ≤ 3 ∧ y &gt; 5 ∧ ¬C</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>x == a</td>
<td></td>
<td>y == b</td>
</tr>
</tbody>
</table>

\[
\text{if (CONDITION) return 0;}
\]

\[
\begin{align*}
\text{PARAMETERS:} & \quad \text{CONDITION} \\
\text{P1:} & \quad x > 3 \land y \leq 5 \land \neg C \\
\text{P2:} & \quad x \leq 3 \land y > 5 \land \neg C \\
\end{align*}
\]

\[
\text{PARAMETERS:} & \quad \text{CONDITION} \\
\text{P1:} & \quad x > 3 \land y \leq 5 \land \neg C \\
\text{P2:} & \quad x \leq 3 \land y > 5 \land \neg C \\
\]

\[
\begin{align*}
\text{PARAMETERS:} & \quad \text{CONDITION} \\
\text{P1:} & \quad x > 3 \land y \leq 5 \land \neg C \\
\text{P2:} & \quad x \leq 3 \land y > 5 \land \neg C \\
\end{align*}
\]

\[
\begin{align*}
\text{PARAMETERS:} & \quad \text{CONDITION} \\
\text{P1:} & \quad x > 3 \land y \leq 5 \land \neg C \\
\text{P2:} & \quad x \leq 3 \land y > 5 \land \neg C \\
\end{align*}
\]
Example (4)

Input Space

- P1
- P2
- P3
- P4

P2: $x \leq 3 \land y > 5 \land \neg C$

P3: $x \leq 3 \land y \leq 5 \land \neg C$

Patch Space

- 12

Patch Details

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x \geq a$</td>
<td>$a \geq -10 \land a \leq 0$</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>$y &lt; b$</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$x = a \lor y = b$</td>
<td>$a = 0 \land b = 0$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x \geq a$</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$x = a \lor y = b$</td>
<td>$a = 0 \land b = 0$</td>
<td>1</td>
</tr>
</tbody>
</table>

Example

```c
uint32 cvtRaster(TIFF* tif, uint32* raster, uint32 width, uint32 height)
{
    uint32 y;
tstrip_t strip = 0;
tsize_t cc, acc;
unsigned char* buf;
uint32 rwidth = roundup(width, horizSubSampling);
uint32 rheight = roundup(height, vertSubSampling);
uint32 nrows = ((rowsperstrip > rheight) ? rheight : rowsperstrip);
uint32 rnrows = roundup(nrows, vertSubSampling);
if (CONDITION) return 0;
/* potential divide-by-zero error */
cc = rnrows * rwidth + 2 * (((rnrows * rwidth) / (horizSubSampling * vertSubSampling)));
```
Example (5)

### Inputs to Concolic Program Repair

- **Example**: 
  
  ```
  if (CONDITION) return 0;
  /* potential divide-by-zero error */
  cc = rnrows * rwidth + 2 * ((rnrows * rwidth) / (horizSubSampling * vertSubSampling));
  ```

### Inputs to Concolic Program Repair

- **CVE-2016-3623**: Divide by Zero in LibTIFF v4.0.6!
- **e.g., exploit as TIFF picture source location, (fix template), synthesis components formula about correct behavior in SMT format**

### Example Evaluation

- **Input Space**
  - P3: \( x \leq 3 \land y \leq 5 \land \neg C \)
  - P4: \( x > 3 \land y > 5 \land C \)
- **Patch Space**
  - ID 1
- **Patch Details**

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( x \geq a )</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>( x = a \lor y = b )</td>
<td>( a = 0 \land b = 0 )</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Input Space**
  - P3: \( x \leq 3 \land y \leq 5 \land \neg C \)
  - P4: \( x > 3 \land y > 5 \land C \)
- **Patch Space**
  - ID 1
- **Patch Details**

<table>
<thead>
<tr>
<th>ID</th>
<th>Patch Template</th>
<th>Parameter Constraint</th>
<th># Conc. Patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( x = a \lor y = b )</td>
<td>( a = 0 \land b = 0 )</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \phi := x > 3 \land y > 5 \land \rho \)
\( \rho := (x = 0 \lor y = 0) \)
**Research Problem**

P1: \[ x > 3 \land y \leq 5 \land \neg C \]

P2: \[ x \leq 3 \land y > 5 \land \neg C \]

P3: \[ x \leq 3 \land y \leq 5 \land \neg C \]

**State of the Art**

---

**Our Solution**

---

**Example**

---

**Evaluation**

---

**Summary**

---

**Patch space refinement** based on the exploration of input space.

**Rule out** parts of the input space, which **contradicts** with the patch space.

**Abstract** patches vs. **concrete** patches

**Gradual improvement**

---
Evaluation

Tools/Techniques
- CEGIS
- ExtractFix
- Angelix
- Prophet

Benchmarks
- ExtractFix
- ManyBugs
- SV-COMP

Repair Areas
- Security Vulnerability Repair
- General Test-based Repair
- Fixing Logical Errors

https://cpr-tool.github.io
http://doi.org/10.5281/zenodo.4668317
Comparison with existing APR

```c
static int jpc_dec_parseopts (..) {
    -----------------------------
    - return 0;
    + return opts->maxlyrs;
}
```

```c
static int jpc_dec_process_siz(..) {
    -----------------------------
    - if (!(dec->cmpts = jas_malloc(dec->numcomps *
        sizeof(jpc_dec_cmpt_t))))) {
        + if (!(dec->cmpts = jas_malloc(dec->numcomps *
            sizeof(jpc_dec_cmpt_t))))) || (1)) {
        -----------------------------
    }
}
```

Patches generated by existing APR

- Overfitting patches
- Non-sensical patches
Comparison with existing APR (2)

**CVE-2016-8691**

```c
static int jpc_siz_getparms(...) {
  -----------------------------
  + if (siz->comps[i].hsamp == 0)
    return -1;
  -----------------------------
}
```

CPR generates correct Patch

- **Initial Patch Space**: 260
- **Refined Patch Space**: 96
- **Refinement**: 63%
- **Rank of Correct Patch**: 1
## Evaluation Insights

<table>
<thead>
<tr>
<th>ID</th>
<th>Buggy Program</th>
<th>Project</th>
<th>Bug ID</th>
<th>Components</th>
<th>Our CEGIS Implementation</th>
<th>CPR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>General</td>
<td>Custom</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Libtiff</td>
<td>CVE-2016-5321</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Libtiff</td>
<td>CVE-2014-8128</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Libtiff</td>
<td>CVE-2016-3186</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Libtiff</td>
<td>CVE-2016-5314</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Libtiff</td>
<td>CVE-2016-9279</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Libtiff</td>
<td>bugzilla 2633</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Libtiff</td>
<td>CVE-2016-10094</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Libtiff</td>
<td>CVE-2017-7601</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Libtiff</td>
<td>CVE-2016-3623</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Libtiff</td>
<td>CVE-2017-7595</td>
<td>4</td>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Libtiff</td>
<td>bugzilla 2611</td>
<td>4</td>
<td>3</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Binutils</td>
<td>CVE-2018-10372</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Binutils</td>
<td>CVE-2017-15025</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Libxml2</td>
<td>CVE-2016-1834</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Libxml2</td>
<td>CVE-2016-1838</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Libxml2</td>
<td>CVE-2016-1839</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Libxml2</td>
<td>CVE-2012-5134</td>
<td>4</td>
<td>3</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Libxml2</td>
<td>CVE-2017-5969</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Libhpeg</td>
<td>CVE-2018-14498</td>
<td>4</td>
<td>3</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Libhpeg</td>
<td>CVE-2018-19664</td>
<td>4</td>
<td>3</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Libhpeg</td>
<td>CVE-2017-15232</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Libhpeg</td>
<td>CVE-2012-2806</td>
<td>4</td>
<td>3</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>FFmpeg</td>
<td>CVE-2017-9992</td>
<td>6</td>
<td>3</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>FFmpeg</td>
<td>Bugzilla-1404</td>
<td>4</td>
<td>2</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Jasper</td>
<td>CVE-2016-8692</td>
<td>4</td>
<td>3</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Jasper</td>
<td>CVE-2016-9387</td>
<td>5</td>
<td>3</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Coreutils</td>
<td>Bugzilla 2654</td>
<td>5</td>
<td>3</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Coreutils</td>
<td>GNUBug 25003</td>
<td>4</td>
<td>4</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Coreutils</td>
<td>GNUBug 25025</td>
<td>4</td>
<td>2</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Coreutils</td>
<td>Bugzilla 19784</td>
<td>4</td>
<td>3</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**CPR is more effective than CEGIS wrt input and patch space exploration.**

**Up 74% Patch Space Reduction**

**CPR can gradually refine the patch space via concolic exploration.**

**CPR can be used for test-guided general-purpose repair and security repair.**

**CPR provides highly ranked patches.**
Concolic Program Repair

Infeasability checks

.. in the input space
Path Reduction:
For every generated input, we check that there is one patch that can exercise the corresponding path. Otherwise, the path will not be explored.

For example:

\[ \phi := x > 3 \land y > 5 \land \rho \]
\[ \rho := (x = 0 \lor y = 0) \]

.. in the patch space

Patch Reduction:
If a patch allows inputs to exercise a path that violates the specification, we identify this as a patch that overfits the valid set of values and attempt to refine it.

Fresh look on program repair

Input Space
- initial test case
- explored path (input partition)
- plausible patches
- represented with abstract patches

Patch Space
- refined patch set
- correct patch (set)

Workflow

Input
- Buggy Program
- Failing test cases
- Fix Locations
- User Specification

Concolic Program Repair
- Patch Pool Construction
- Input Generation
- Concolic Execution
- Patch Reduction
- ranking (input space)
- ranked patches

anytime algorithm (gradual improvement)

Evaluation

Summary

Our Solution
Predict patch
- Learning-inference
- Concolic Program Repair

Challenges
- How to provide high quality but few patches?
- How to avoid non-sensical patches?
- How to produce less overfitting patches?
- How to repair bugs in the absence of many test cases?

State of the Art
- Patch synthesis
- Buggy Program
- based repair
- constraints
- Extract