Staged Program Repair with Condition Synthesis

Author: Fan Long and Martin Rinard (MIT)
Problem
DatePeriod (input_arg) {
    char *isostr = NULL; int isostr_len = 0;
    if (print_type(input_arg, 1) == "string") {
        isostr = arg_pointer(input_arg, 1);
        isostr_len = strlen(input_arg);
    }
    DateInterval* interval;
    if (print_type(input_arg, 1) == "dateInterval") {
        interval = arg_pointer(input_arg, 1);
    }
    if (isostr_len) {
        date_period_initialize_by_str(isostr, isostr_len);
    } else {
        date_period_initialize_by_interval(interval);
    }
}
Test the code

DatePeriod (input_arg){
    char *isostr = NULL; int isostr_len = 0;
    if ( print_type(input_arg,1)=="string"){
        isostr = arg_pointer(input_arg,1);
        isostr_len = strlen(input_arg);
    }
    DateInterval* interval;
    if ( print_type(input_arg,1)=="dateInterval"){
        interval = arg_pointer(input_arg,1);
    }
    if(isostr_len){
        date_period_initialize_by_str(isostr,isostr_len);
    }else{
        date_period_initialize_by_interval(interval);
    }
}
One plausible fix

```c
DatePeriod (input_arg){
    char *isostr = NULL; int isostr_len = 0;
    if ( print_type(input_arg,1)=="string"){
        isostr = arg_pointer(input_arg,1);
        isostr_len = strlen(input_arg);
    }
    DateInterval* interval;
    if ( print_type(input_arg,1)=="dateInterval"){
        interval = arg_pointer(input_arg,1);
    }
    if(isostr_len){
        date_period_initialize_by_str(isostr,isostr_len);
    }else{
        date_period_initialize_by_interval(interval);
    }
}
```

Negative test case:

```c```
```

Only fix uninitialized problem

```
```
```
```
```
```
```
```
```
```
```
A correct fix

```c
DatePeriod (input_arg){
    char *isostr = NULL; int isostr_len = 0;
    if ( print_type(input_arg,1)=="string"){
        isostr = arg_pointer(input_arg,1);
        isostr_len = strlen(input_arg);
    }
    DateInterval* interval;
    if ( print_type(input_arg,1)=="dateInterval"){
        interval = arg_pointer(input_arg,1);
    }
    if(isostr_len){
        date_period_initialize_by_str(isostr,isostr_len);
    }else{
        date_period_initialize_by_interval(interval);
    }
}
```
Our goal

**Buggy code:**
```c
DatePeriod (input_arg){
    char *isostr = NULL; int isostr_len = 0;
    if ( print_type(input_arg,1)==”string”){
        isostr = arg_pointer(input_arg,1);
        isostr_len = strlen(input_arg);
    ...
```

**Many test cases**
- **Negative test case:**
  ```
  “”
  ```
- **Positive test cases:**
  ```
  “R4/2012-07-01T00:00:00Z/P7D”
  DateInterval(“P7D”);
  ```

**One plausible code fix:**
```c
... if(isostr){
    //originally:     if(isostr_len){
...
```

**Another plausible code fix:**
```c
... DateInterval interval=NULL;
    //originally:  DateInterval interval;
...
```

......
Search problem

Enumerate + Evaluate
=Predict the future

---

**Buggy code:**
DatePeriod (input_arg){
    char *isostr = NULL; int isostr_len = 0;
    if ( print_type(input_arg,1)=="string"){
        isostr = arg_pointer(input_arg,1);
        isostr_len = strlen(input_arg);
    ...

---

**Plausible code:**
...
    if(isostr){
        //originally:   if(isostr_len){
    ...
    ...
An old problem

Enumerate + Evaluate
=Predict the future

Current state

Win state
Related Work
Baseline approaches

Delete or Copy the code from elsewhere

Random search: RSRepair [1]
Genetic programming: GenProg [2]

Pruning repeats: AE [3]

Larger Search Space

Simple transform rules: Debroy and Wong [4]

Complex transform rules: PAR [5] (presented last week)

More Complex transform rules: This work

Given Training Data

Model the probability of correct fixes
Repair Shape [7], and Prophet [8]

Transfer patches from other codes:
CodePhage [6]

Successful fix before

Main contributions of this work

More Complex transform rules
Better error localization

Prune the branch if we know it is not possible to reach our goal from here

Experiment:
1. Pruning speeds up the tool from 3x to 120x in a benchmark.
2. Fix 12x more bugs than GenProg [2].

Methods
Staged Program Repair

- Input:
  1. A program
  2. A test suite
     a. Positive test cases - program produces correct output
     b. Negative test cases - program produces incorrect output
        (exposes the defect)

- Goal Output:
  - A modified program that produces correct output for all tests
Algorithm

1) Fault Localization

2) Transformation Schema

3) Condition Synthesis
   a) Target Value Search
   b) Condition Generation
Error Localizer

Modifies source code to have a callback before each statement that records the time of execution.
Source code is recompiled on all test cases.
This allows us to identify and prioritize target statements that:

a. are executed with more negative test cases
b. are executed with fewer positive test cases
c. are executed later during executions with negative test cases
PHP Example

if (isostr_len) {

    // Handle (string) case
    date_period_initialize(&(dpobj->start), &(dpobj->end),
    &(dpobj->interval), &recurrences, isostr, isostr_len); ...
}

else {

    // Handle (DateTime,...) cases
    /* pass uninitialized ‘interval’ */

    intobj = (php_interval_obj *)
    zend_object_store_get_object(interval); ...  

} Always executed in negative test cases, rarely executed in positive test cases!
PHP Example

if (isostr_len) {
    // Handle (string) case
    date_period_initialize(&(dpobj->start), &(dpobj->end),
    &(dpobj->interval), &recurrences, isostr, isostr_len); ...
} else {
    // Handle (DateTime,...) cases
    /* pass uninitialized 'interval' */
    intobj = (php_interval_obj *)
    zend_object_store_get_object(interval); ...
}
Transformation Schemas

Now, we’ve obtained a set of target statements on which we can apply transformation schemas on

Example schemas:

*Condition Refinement:*
Given a target “if” statement, conjoin or disjoin an abstract condition to the original if condition.

*Condition Introduction:*
Given a target statement, transform it so that it executes only if an abstract condition is true
PHP Example

if (isostr_len) {

    // Handle (string) case

    date_period_initialize(&(dpobj->start), &(dpobj->end),
    &(dpobj->interval), &recurrences, isostr, isostr_len); ...

} else {

    // Handle (DateTime,...) cases

    /* pass uninitialized 'interval' */

    intobj = (php_interval_obj *)

    zend_object_store_get_object(interval); ...

}
PHP Example

if (isostr_len || abstract_cond() ) {

    // Handle (string) case
    date_period_initialize(&(dpobj->start), &(dpobj->end),
    &(dpobj->interval), &recurrences, isostr, isostr_len); ...
}

} else {

    // Handle (DateTime,...) cases
    /* pass uninitialized ‘interval’ */
    intobj = (php_interval_obj *)
    zend_object_store_get_object(interval); ...
}
Target Condition Value Search

Performed when a statement contains an abstract condition
SPR searches for a value of `abstract_cond()` that
produces a correct output for the negative test case
Done by repeatedly generating a different sequence of 0/1
return values from `abstract_cond()` on each
execution

Record a mapping from the current “environment” (i.e.
any variables in the surrounding context) to the return
value of `abstract_cond()`.
Condition Generation

Use the recorded mappings to generate a symbolic condition that approximates the mappings.

In other words, the abstract condition is instantiated with a symbolic condition given by the associated environment.
if (isostr_len || abstract_cond() ) {
    // Handle (string) case
    date_period_initialize(&(dpobj->start), &(dpobj->end),
    &(dpobj->interval), &recurrences, isostr, isostr_len); ...
} else {
    // Handle (DateTime,...) cases
    /* pass uninitialized ‘interval’ */
    intobj = (php_interval_obj *)
    zend_object_store_get_object(interval); ...
}
PHP Example

```
if (isostr_len || abstract_cond() ) {

<table>
<thead>
<tr>
<th>test cases</th>
<th>abstract_cond() target value</th>
<th>isostr_len variable value</th>
<th>isostr variable value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval object</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>not empty string</td>
<td>1</td>
<td>1</td>
<td>000</td>
</tr>
<tr>
<td>empty string</td>
<td>1</td>
<td>0</td>
<td>XXX</td>
</tr>
</tbody>
</table>

Step 1: exist target values which can pass all cases

Step 2: synthesize condition based on variable values

if (isostr_len || (isostr != 0) ) {
```
PHP Example

if (isostr_len || (isostr != 0)) {
    // Handle (string) case
    date_period_initialize(&(dpobj->start), &(dpobj->end),
                           &(dpobj->interval), &recurrences, isostr, isostr_len); ...
}
else {
    // Handle (DateTime,...) cases
    /* pass uninitialized ‘interval’ */
    intobj = (php_interval_obj *)
    zend_object_store_get_object(interval); ...
}
Summary

Input: A program, positive test cases, negative test cases

1. Fault Localization
2. Transformation Schema
3. Condition Synthesis
   - Target Value Search
   - Condition Generation

Output: A repaired program that passes all test cases
Experiments
Benchmark

Proposed by GenProg [2]

8 different applications

Average lines of code: ~642000

Average number of test cases: ~1234

Total number of bugs: 69

Total number of feature changes: 36

Main results

<table>
<thead>
<tr>
<th>Search range</th>
<th>This work (SPR)</th>
<th>GenProg [2]</th>
<th>AE [3]</th>
</tr>
</thead>
<tbody>
<tr>
<td># Plausible</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixes in PHP</td>
<td><strong>16/31</strong></td>
<td>5/31</td>
<td>7/31</td>
</tr>
<tr>
<td>fixes in others</td>
<td><strong>22/38</strong></td>
<td>11/38</td>
<td>18/38</td>
</tr>
<tr>
<td>feature changes</td>
<td><strong>3/36</strong></td>
<td>2/36</td>
<td>2/36</td>
</tr>
<tr>
<td># Correct</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixes in PHP</td>
<td><strong>9/31</strong></td>
<td>1/31</td>
<td>2/31</td>
</tr>
<tr>
<td>fixes in others</td>
<td><strong>2/38</strong></td>
<td>0/38</td>
<td>0/38</td>
</tr>
<tr>
<td>feature changes</td>
<td><strong>0/36</strong></td>
<td>1/36</td>
<td>1/36</td>
</tr>
<tr>
<td>Average time per bug</td>
<td><strong>86 m</strong></td>
<td>???</td>
<td>???</td>
</tr>
</tbody>
</table>

## Larger search space

<table>
<thead>
<tr>
<th>In search space</th>
<th>This work (SPR)</th>
<th>GenProg [2]</th>
<th>AE [3]</th>
</tr>
</thead>
<tbody>
<tr>
<td># Correct in PHP</td>
<td>13/44</td>
<td>~2/44</td>
<td>~6/44</td>
</tr>
<tr>
<td># Correct in others</td>
<td>7/61</td>
<td>~1/61</td>
<td>~3/61</td>
</tr>
<tr>
<td># First correct</td>
<td>11/20</td>
<td>???</td>
<td>???</td>
</tr>
</tbody>
</table>

Why does every algorithm fix more bugs in PHP?

PHP contains more easy bugs in the benchmark

More test cases in PHP (8471) than in others (avg. 200)
Target value search (pruning)

Among 11 first plausible and correct repairs

Average pruning successful rate: ~98.7%

This means we only need to synthesize ~1% of abstract condition

Average speed up: 44.5x

More candidate repairs need to consider without condition value search
Discussion
Discussion 1

When will the method work well?
Discussion 1

When will the method work well?

Possible answers:
Many test cases, simple bugs (only 1 transform schema), shorter code length...
Discussion 2

Why does this work repair much more bugs than the previous approaches?
Discussion 2

Why does this work repair much more bugs than the previous approaches?

Possible answers:
Larger search space, more effective pruning, overfitting...
Discussion 3

Do you think there is an overfitting problem in the experiment of the work?
Discussion 3

Do you think there is an overfitting problem in the experiment of the work?

Possible answers:

Both the design of the transformation schemas and prioritization of applying the schemas might not generalize well.
Discussion 4

This method is known to be state of the art. Can you find any limitations in this work?
Discussion 4

This method is known to be state of the art. Can you find any limitations in this work?

Possible answers:

Only performs a single fix at a time. Does not consider the process of human interactions while debugging
Discussion 5

What are some ways in which we can improve the work? (e.g., more advanced AI search techniques)
Discussion 5

What are some ways in which we can improve the work? (e.g., more advanced AI search techniques)

Possible answers:

We might evaluate intermediate states by fixing partial bugs and search promising states deeper.
Thank you