Staged Program Repair with Condition Synthesis

Author: Fan Long and Martin Rinard (MIT)

slide author names omitted for FERPA compliance

Problem

Pseudocode with a bug

```
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);
    }
}
```

An empty string as the argument?

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);
    }
}
```

Negative test case:

“”

Positive test cases:

“R4/2012-07-01T00:00:00Z/P7D”
DateInterval(“P7D”);
…

Error!
Uninitialized interval

One plausible fix

```
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:

“”

(not check the object after initialized)

```
DateInterval* interval= ...
                new DateInterval(1);
```

Only fix
uninitialized problem

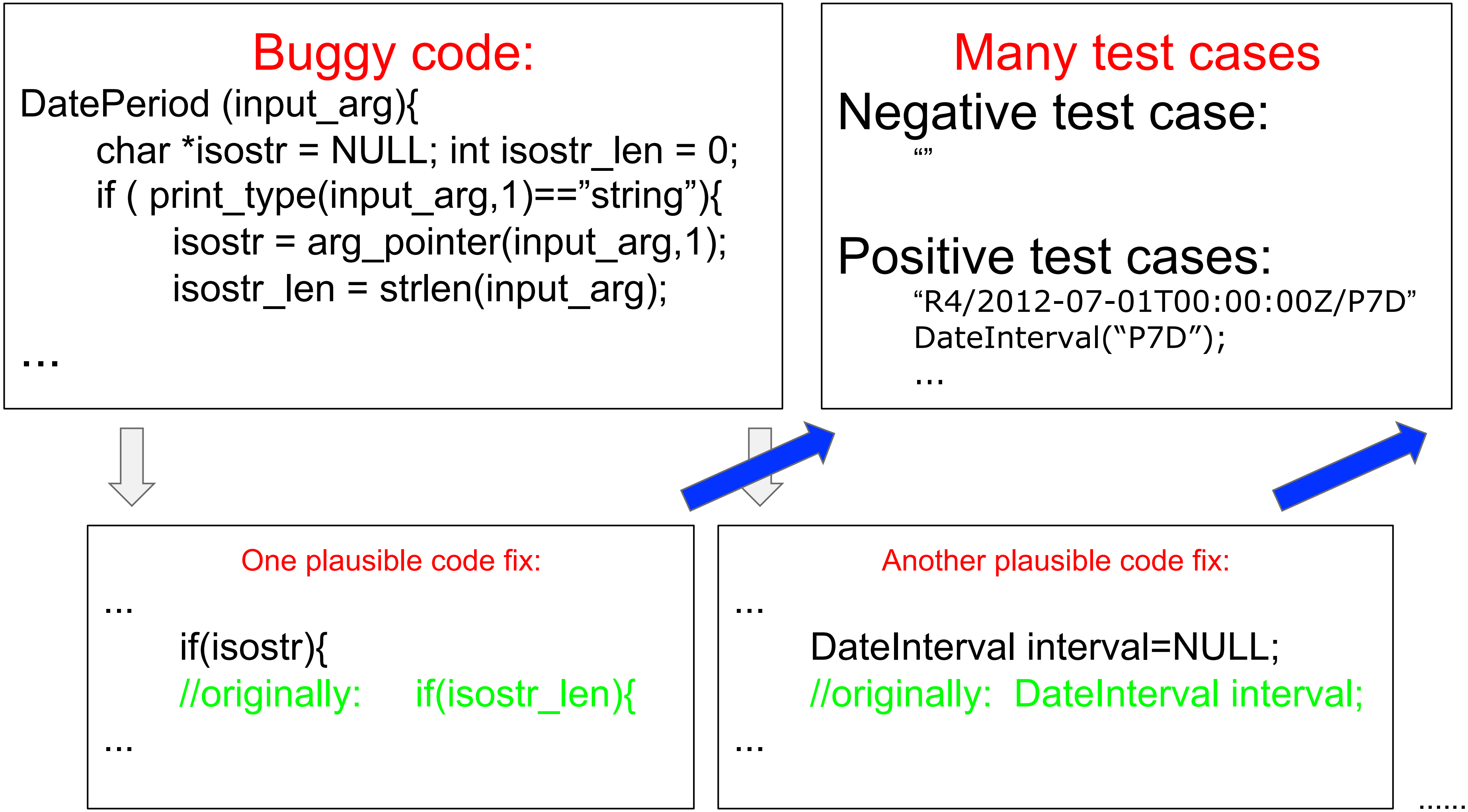
A correct fix

```
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){          →          if(isostr){
		date_period_initialize_by_str(isostr,isostr_len);
	}else{
		date_period_initialize_by_interval(interval);
	}
}
```

Negative test case:

“”

Our goal



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);

...
```

Many test cases

Negative test case:

“”

Positive test cases:

“R4/2012-07-01T00:00:00Z/P7D”
DateInterval(“P7D”);
...

One plausible code fix:

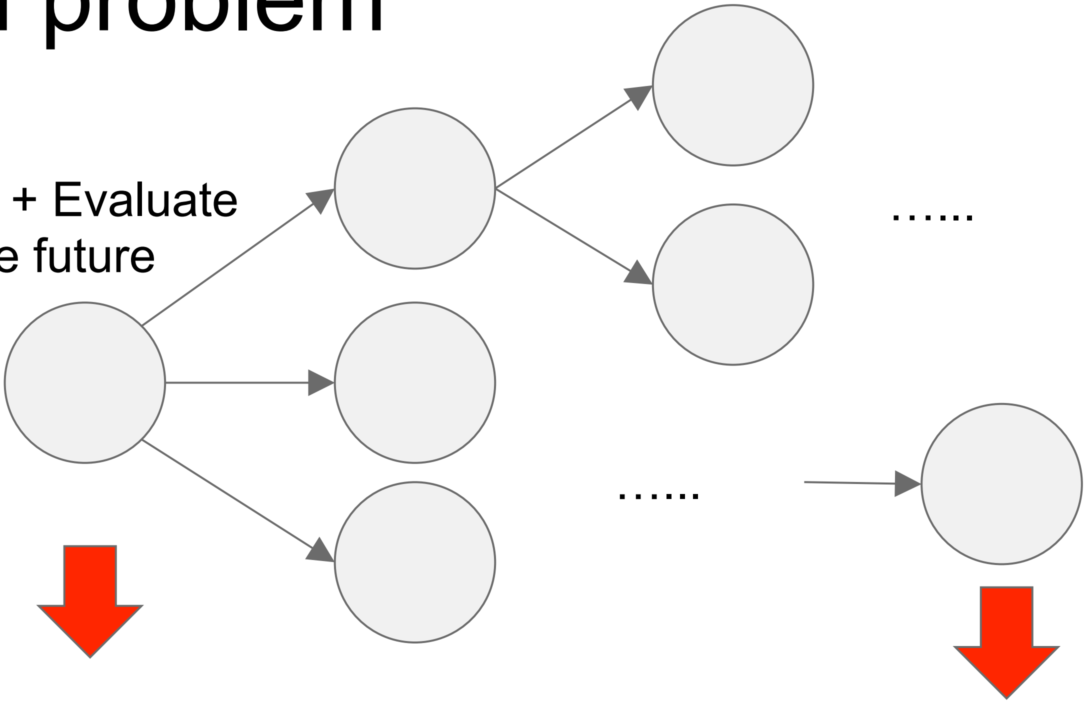
```
...
        if(isostr){
        //originally:     if(isostr_len){
...
```

Another plausible code fix:

```
...
        DateInterval interval=NULL;
        //originally:  DateInterval interval;
...
```

......

Search problem



Enumerate + Evaluate
=Predict the future

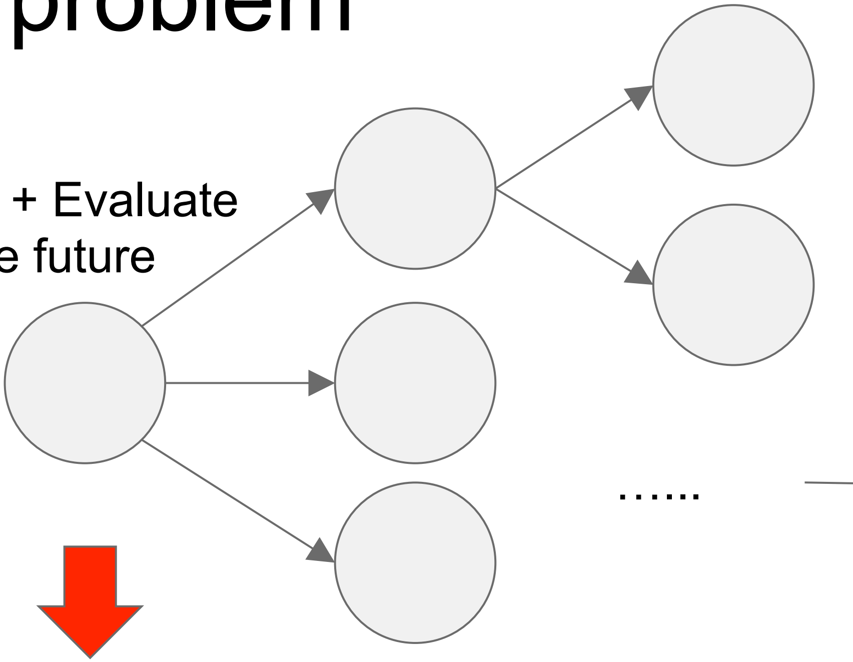
Buggy code:

```
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);
...
```

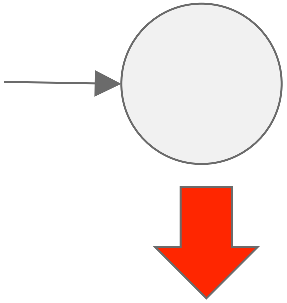
Plausible code:

```
...
        if(isostr){
        //originally:     if(isostr_len){
...
```

An old problem

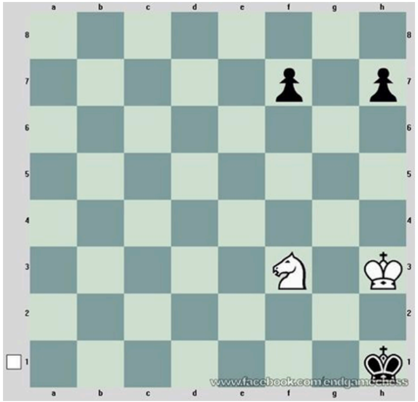


Enumerate + Evaluate
=Predict the future



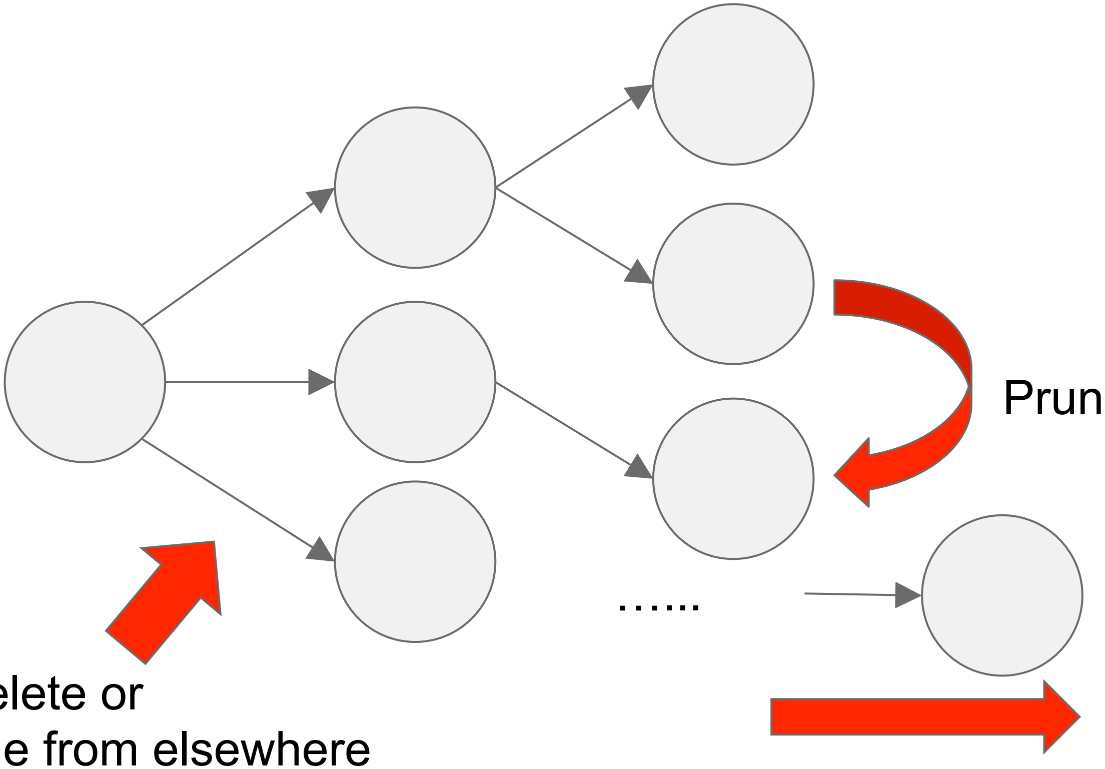
Current state

Win state



Related Work

Baseline approaches



Pruning repeats: AE [3]

Delete or
Copy the code from elsewhere

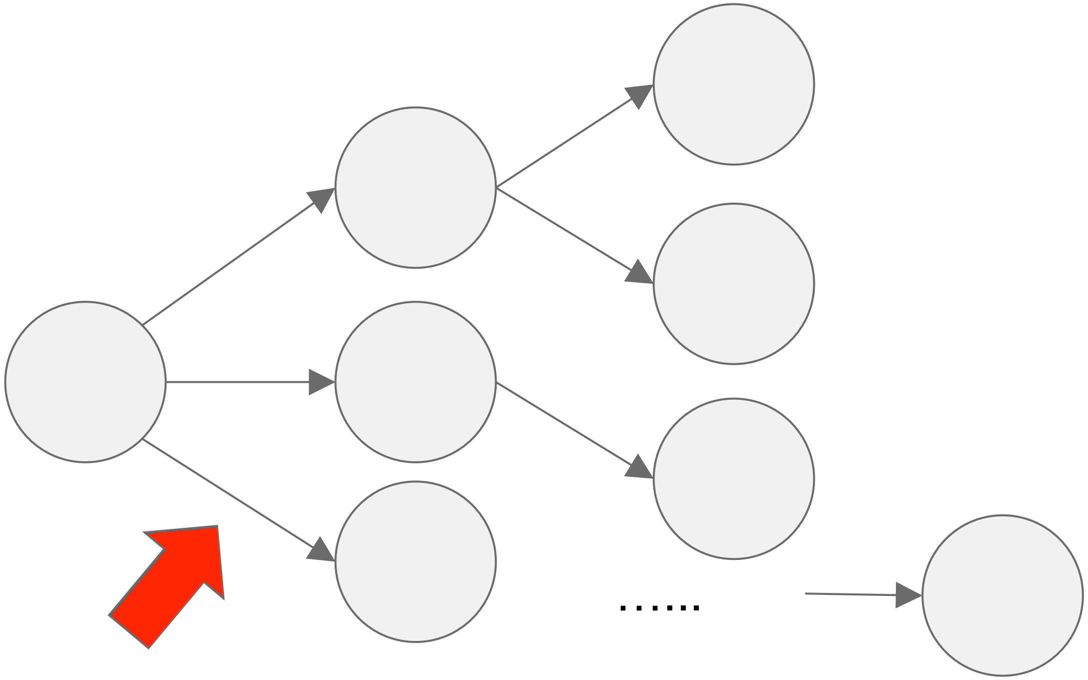
Random search: RSRepair [1]
Genetic programming: GenProg [2]

[1] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang. The strength of random search on automated program repair, ICSE 2014,

[2] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming, ICSE 2009

[3] W. Weimer, Z. P. Fry, and S. Forrest. Leveraging program equivalence for adaptive program repair: Models and first results, ASE 2013

Larger Search Space



Simple transform rules: Debroy and Wong [4]

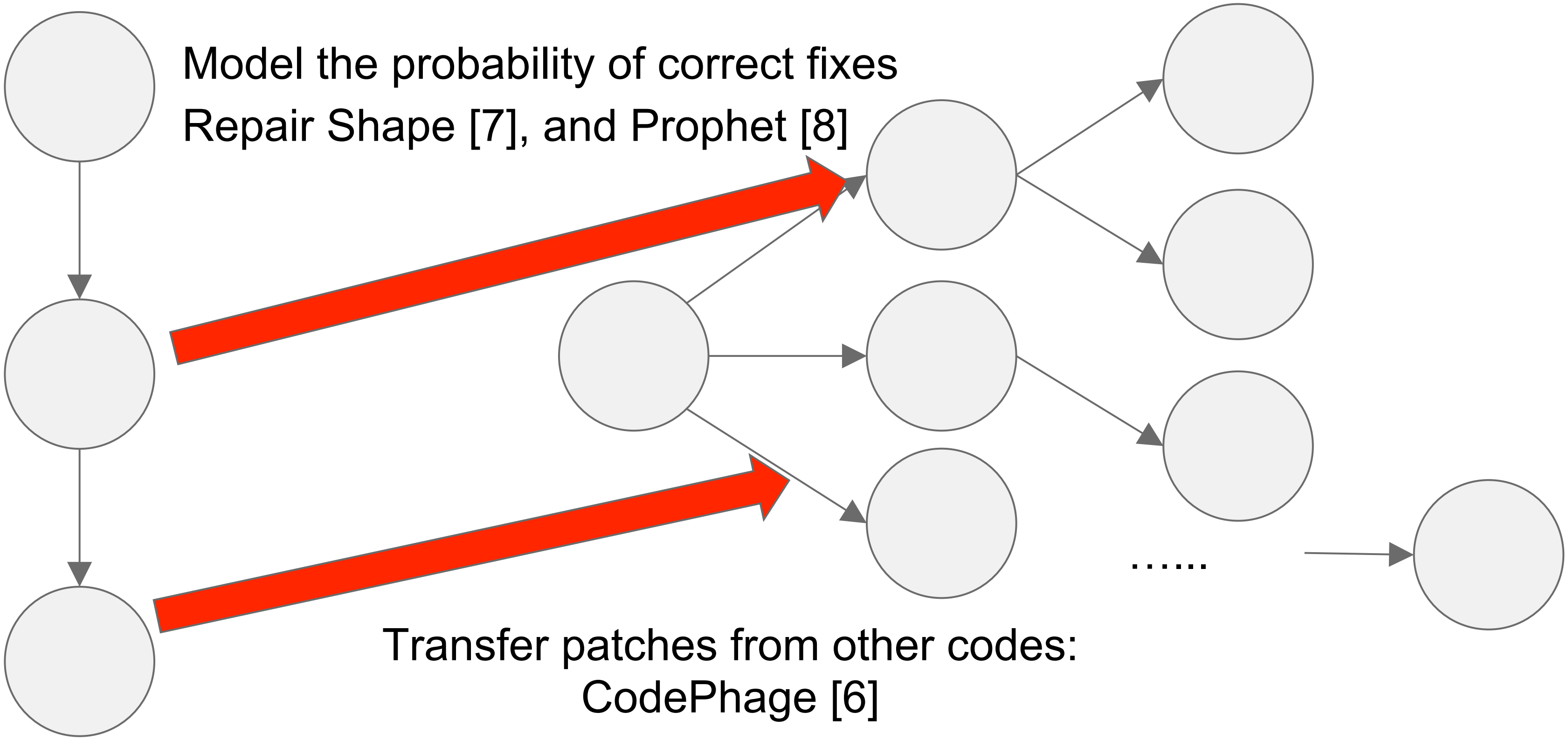
Complex transform rules: PAR [5] (presented last week)

More Complex transform rules: This work

[4] V. Debroy and W. E. Wong. Using mutation to automatically suggest fixes for faulty programs. ICST, 2010

[5] D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. ICSE 2013

Given Training Data



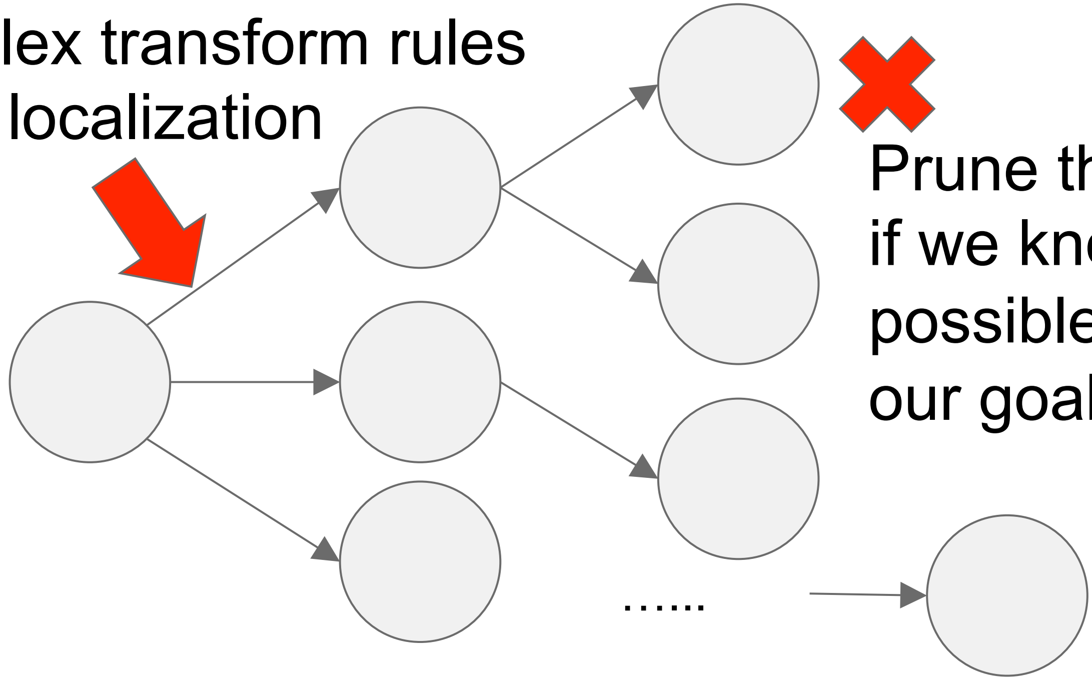
Model the probability of correct fixes
Repair Shape [7], and Prophet [8]

Transfer patches from other codes:
CodePhage [6]

Successful fix before

[6] S. Sidiroglou-Douskos et. al., Automatic error elimination by horizontal code transfer across multiple applications. ACM SIGPLAN 2015
[7] M. Martinez and M. Monperrus. Mining software repair models for reasoning on the search space of automated program fixing. ESE 2015.
[8] F. Long and M. Rinard. Prophet: Automatic patch generation via learning from successful patches. Technical Report MIT-CSAIL, 2015.

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].

[2] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming, ICSE 2009

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); ...
}
```

High priority repair target statement

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); ...
}
```

Apply condition refinement on target statement

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.

PHP Example

Notice that **isostr** is never 0 in the negative test cases, but always 0 when abstract_cond() is invoked in the positive test case

```
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() ) {
```

test cases	abstract_cond() target value	isostr_len variable value	isostr variable value
Interval object	0	0	0
not empty string	1	1	OOO
empty string (negative case)	1	0	XXX

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

Replace abstract_cond() so now all tests now pass, making this is a successful repair!

```
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

[2] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming, ICSE 2009

Main results

		This work (SPR)	GenProg [2]	AE [3]
Search range		**All codes**	A specific file	A specific file
# Plausible	fixes in PHP	**16/31**	5/31	7/31
	fixes in others	**22/38**	11/38	18/38
	feature changes	**3/36**	2/36	2/36
# Correct	fixes in PHP	**9/31**	1/31	2/31
	fixes in others	**2/38**	0/38	0/38
	feature changes	0/36	**1/36**	**1/36**
Average time per bug		86 m	???	???

[2] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming, ICSE 2009

[3] W. Weimer, Z. P. Fry, and S. Forrest. Leveraging program equivalence for adaptive program repair: Models and first results, ASE 2013

Larger search space

		This work (SPR)	GenProg [2]	AE [3]
In search space	# Correct in PHP	**13/44**	~2/44	~6/44
	# Correct in others	**7/61**	~1/61	~3/61
	# First correct	11/20	???	???

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