Trident: Controlling Side Effects in Automated Program Repair

Nikhil Parasaram
Earl Barr
Sergey Mechtaev

Memory Manipulating Patches

A Memory manipulating patch (patch with side effect) is a set of transformations which manipulate the memory locations.

Memory Manipulating Patches

A Memory manipulating patch (patch with side effect) is a set of transformations which manipulate the memory locations.

```
for(i=0;i<q;i++){
    scanf("%d",&a);
-   b[a] = 1;
+   g[a] = 2;
}
```

Example-1: Memory Manipulating Patch

Memory Manipulating Patches

A Memory manipulating patch (patch with side effect) is a set of transformations which manipulate the memory locations.

```
for(i=0;i<q;i++){
    scanf("%d",&a);
-   b[a] = 1;
+   g[a] = 2;
}
```

Example-1: Memory Manipulating Patch

```
    ...
    scanf("%d",&a);
-
+   save_and_incr(&a);
    ...
```

Example-2: Memory Manipulating Patch

Memory Manipulating Patches

A Memory manipulating patch (patch with side effect) is a set of transformations which manipulate the memory locations.

```
for(i=0;i<q;i++){
    scanf("%d",&a);
-   b[a] = 1;
+   g[a] = 2;
}
```

Example-1: Memory Manipulating Patch

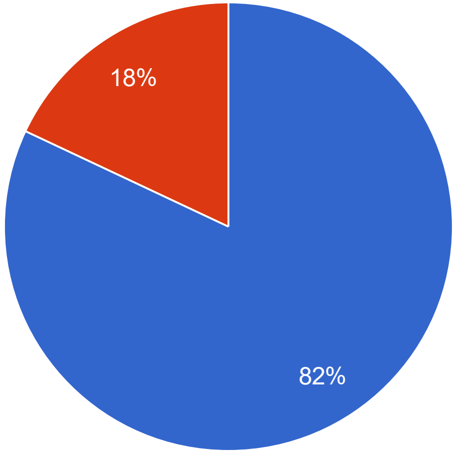
```
    ...
    scanf("%d",&a);
-
+   save_and_incr(&a);
    ...
```

Example-2: Memory Manipulating Patch

```
for(i=0;i<q;i++){
    scanf("%d",&a);
-   b[a] = 1;
+   b[a] = 3;
}
```

Example-3: Non-memory manipulating patch.

Memory Manipulating Patches Are Common



Around 82% of the bugs sampled from GitHub were fixed with memory manipulating patches.

Memory Manipulating Patches

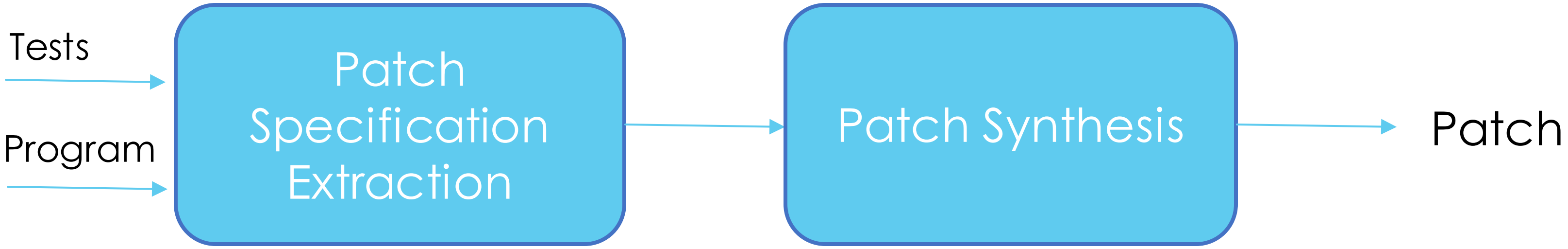
Image Source: https://depositphotos.com/5610300/stock-photo-needle-in-a-haystack.html

Memory Manipulating Patches

Memory Manipulating patches change the existing L-Values and possibly the R-Values of a statement in the program.

```
 r  =  y  +  w;
L-value   R-value   R-Value
```

Semantic Program Repair Approaches



Generating R-Values

```
 r =  x + 1;
if (r – y>0)
   r = y;
printf("%d", r);
```

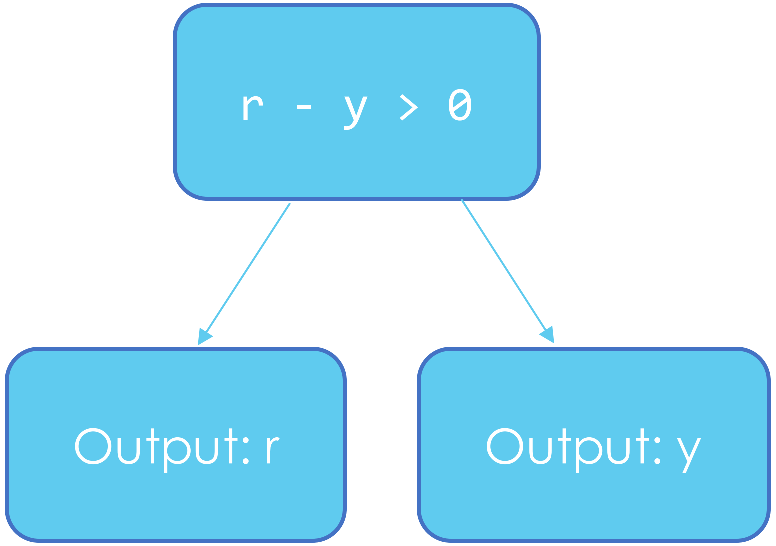
→

```
r = □^R;
if (r – y > 0)
   r = y;
printf("%d", r);
```

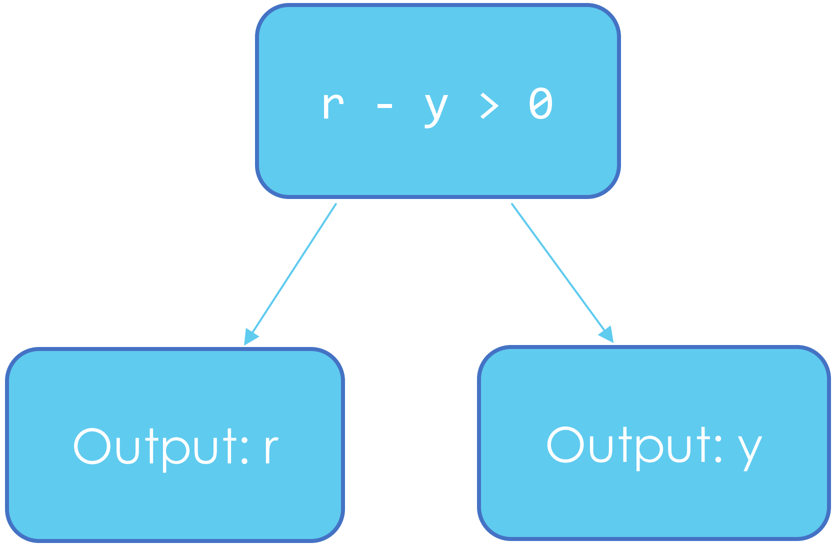
Generating R-Values

```
 r = x + 1;
if (r – y>0)
   r = y;
printf("%d", r);
```

```
r  = □R;
if (r – y > 0)
   r = y;
printf("%d", r);
```



Generating R-Values



r - y > 0

Output: r

Output: y

π_1: (r – y > 0) ^ (output = y)

π_2: (r - y ≤ 0) ^ (output = r)

$\exists\ p \in S.\ \bigvee_i \pi_i[\square^R \rightarrow p] \wedge \text{output} = \text{ExpectedOutput}$

Where S is the search space of patches.

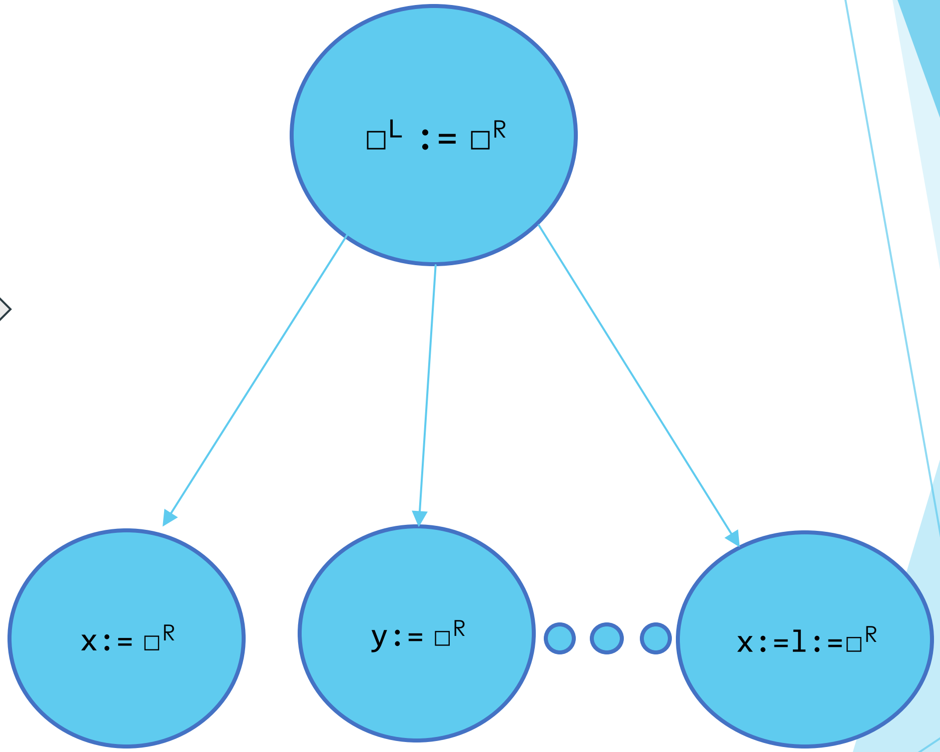
Extending to L-Values

```
int clamp(int x, int l,
int h) {
    int r;
    $\square^L$ := $\square^R$
    if (x < l)
        r = l;
    if (r == x && x > h)
        r = h;
  return r;
}
```

```
int clamp(int x, int l, int
h) {
    int r;
    switch(PATCH_ID) {
      case 0: x := $\square^R$;
      break;
      case 1: y := $\square^R$;
      break;
      case 2: l := $\square^R$;
      break;
      case 3: h := $\square^R$.
      break;
    }
    if (x < l)
        r = l;
    if (r == x && x > h)
        r = h;
    return r;
```

Extending it to L-Values

```
int clamp(int x, int l,
int h) {
    int r;
    switch(PATCH_ID) {
      case 0: x := □^R;
      break;
      case 1: y := □^R;
      break;
      case 2: l := □^R;
      break;
      …
      case n: x :=l:=□^R
      break; }
    if (x < l)
        r = l;
    if (r == x && x > h)
```



State Space Explosion
due to 30 states.

Extending it to L-Values

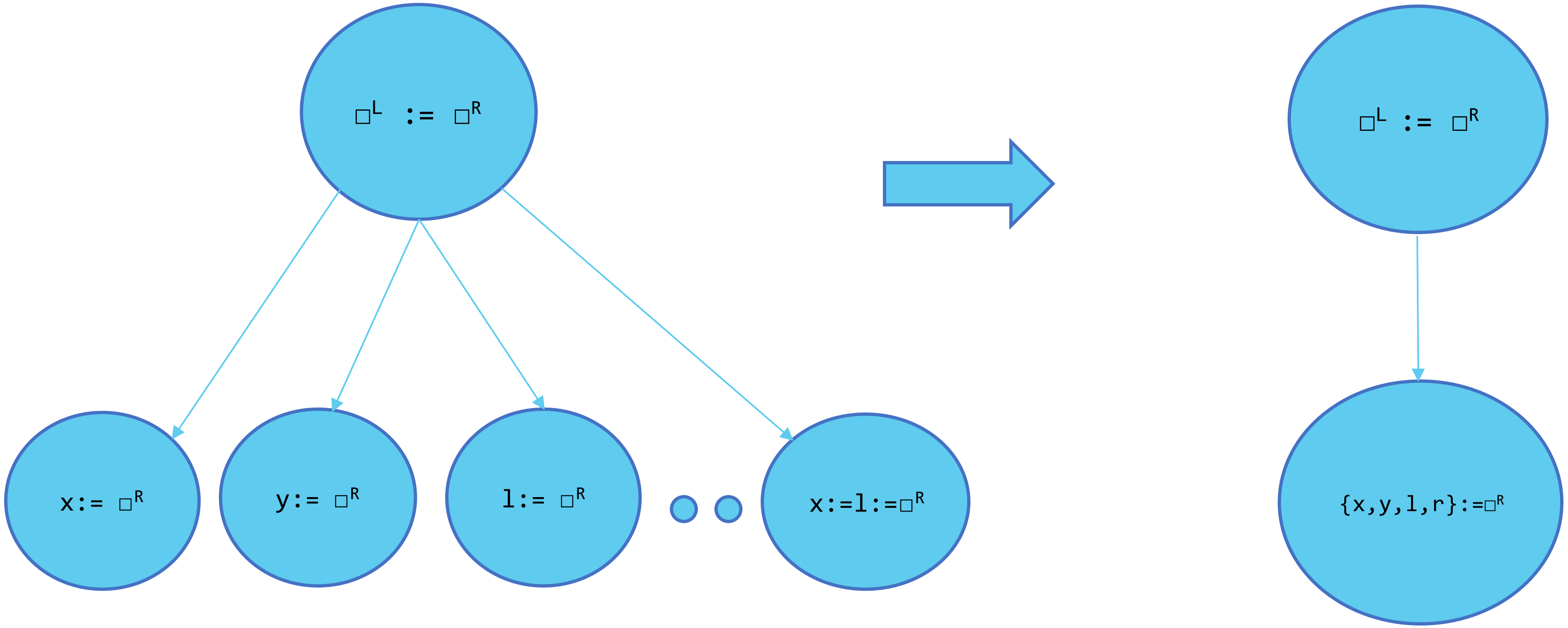
```
int clamp(int x, int l,
int h) {
    int r;

    switch(PATCH_ID) {
        case 0: x := □^R;
        break;
        case 1: y := □^R;
        break;
        case 2: l := □^R;
        …
      }

    if (x < l)
        r = l;
    if (r == x && x > h)
        r = h;
    return r;
}
```

```
int clamp(int x, int l,
int h) {
    int r;
    klee_open_merge();
    switch(PATCH_ID) {
        case 0: x := □^R;
        break;
        case 1: y := □^R;
        break;
        case 2: l := □^R;
        ...
    }
    klee_close_merge();
    if (x < l)
        r = l;
    if (r == x && x > h)
        r = h;
    return r;
}
```

State Merging



State Merging

```
int buggy(int x, int y) {
    //missing call inc_if_zero(&x,&y)

    if (x > 0 && y > 0)
        return 1;
    else
        return 0;
```

```
int buggy(int x, int y, int z) {
    $\square^{L}_{1}, \square^{L}_{2}, \square^{L}_{3} = \square^{R}_{1}, \square^{R}_{2}, \square^{R}_{3};$

    if (x > 0 && y > 0)
        return 1;
    else
        return 0;
```

State Merging

$(s_x \rightarrow x' = \alpha_x \wedge \neg s_x \rightarrow x' = x) \wedge$
$(s_y \rightarrow y' = \alpha_y \wedge \neg s_y \rightarrow y' = y) \wedge$
$(s_z \rightarrow z' = \alpha_z \wedge \neg s_z \rightarrow z' = z) \wedge \mathrm{AtMostK}(k, s_x, s_y, s_z)$

For $\beta \in \{x, y, z\}$
s_β : Selectors, when true, select the corresponding variables
α_β : Fresh variables
β' : Value of the program variable β after executing the statement
AtMostK(k, -): Cardinality constraint that restricts k selectors to be True

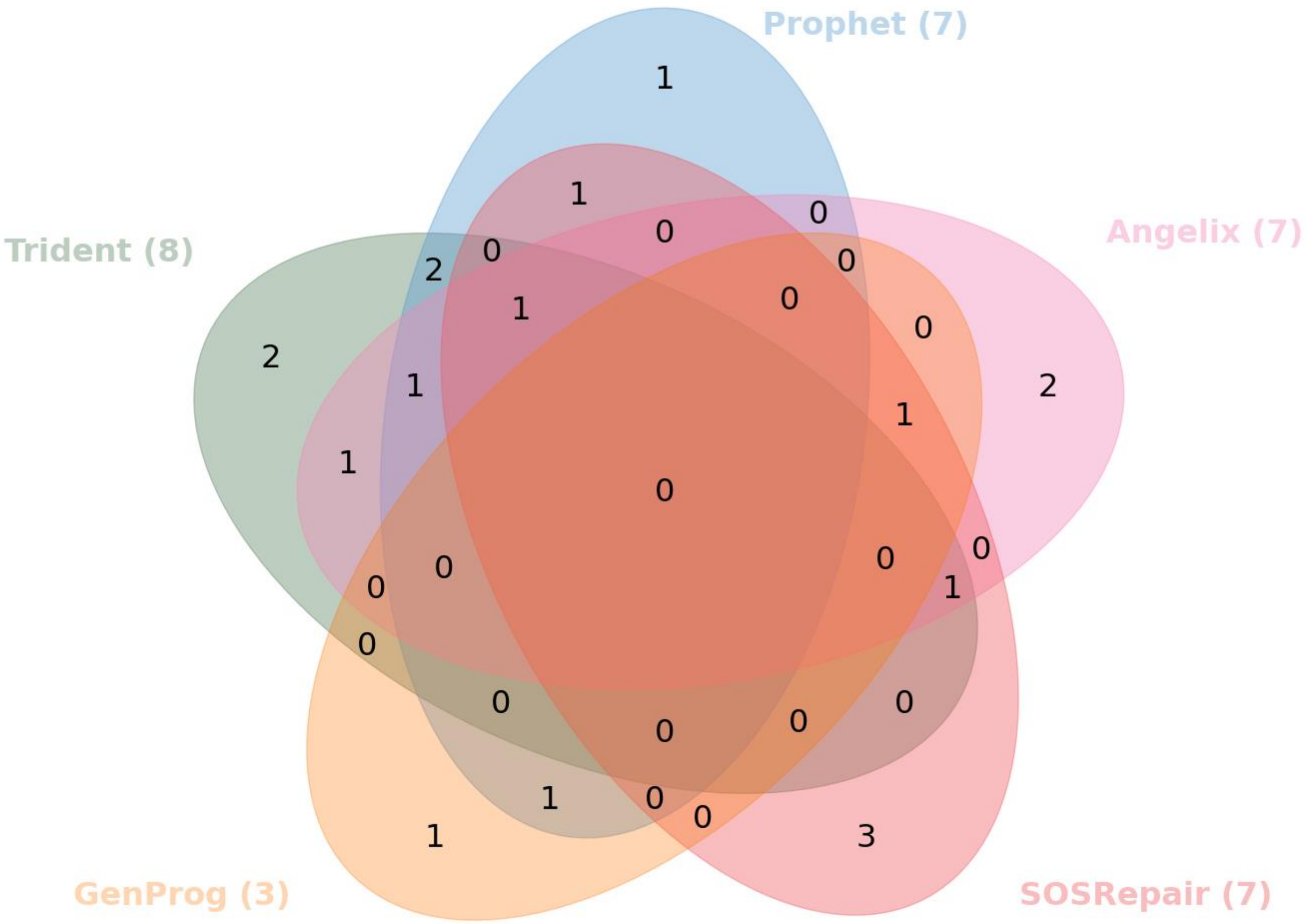
OSS10 Results

Bug	Trident	SKL	GenProg
match_lines=match_words = 0;	✕	✕	✕
strip_trailing_slashes (optarg);	✓	✓	✕
preserve_xattr = true	✓	✕	✕
if (! nfiles) fstatus[0].failed = 1;	Overfit	✕	✕
if (line_width < 0) line_width = 0	✕	✕	✕
relative_to = relative_base	✕	✕	✕
f[i].fd = -1;	✓	✕	✕
end = key->range[3];	✕	✕	✕
flags = option_mask32;	✓	✓	✕
xtc &= ~TC_UOPPOST	Overfit	Overfit	✕
Total	**4 + 2**	**2 + 1**	**0 + 0**

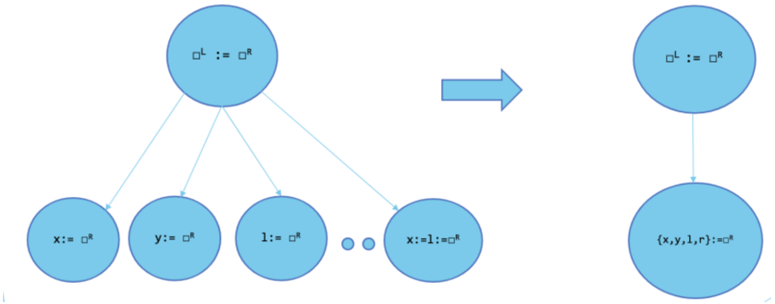
Classes	Paths Average		
	Trident	SKL	SL
191a84a	5.4	13.2	22.5
7585d81	95.0	96.0	96.0
c160afe	**10.0**	**268.0**	**1602.5**
9944e47	**22.0**	**490.5**	**616.0**
ca99c52	5688.0	6024.2	6352.0
9f5aa48	1175.0	1320.0	9000.0
2a80912	**8.0**	**39.0**	**104.0**
0506e29	12922.0	16807.0	16807.0
5c13ab4	**88.0**	**317.0**	**936.0**
6f7a009	12647.0	15966.0	21378.0
Average	**3266.1**	**4133.9**	**5691.3**

Path Count

Correct Fixes For Various Tools



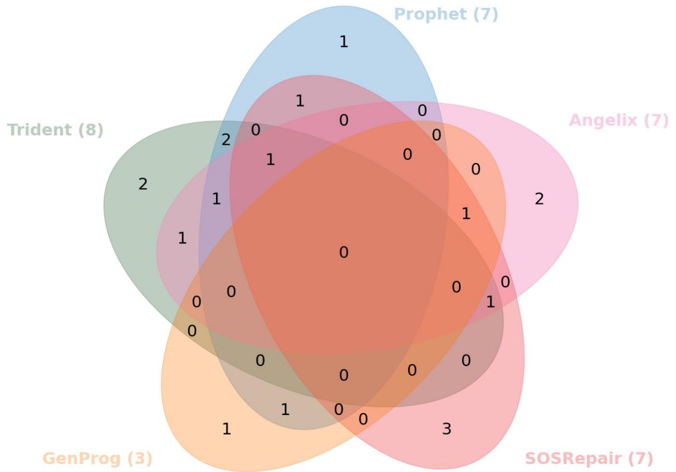
Questions



State Merging Semantics

$(s_x \rightarrow x' = \alpha_x \wedge \neg s_x \rightarrow x' = x) \wedge$
$(s_y \rightarrow y' = \alpha_y \wedge \neg s_y \rightarrow y' = y) \wedge$
$(s_z \rightarrow z' = \alpha_z \wedge \neg s_z \rightarrow z' = z) \ \wedge \text{AtMostK}(k, s_x, s_y, s_z)$

For $\beta \in \{x, y, z\}$
s_β : Selectors, when true, select the corresponding variables
α_β : Fresh variables
β' : Value of the program variable β after executing the statement
AtMostK(k, -): Cardinality constraint that restricts k selectors to be True


Prophet (7)
Trident (8)
Angelix (7)
GenProg (3)
SOSRepair (7)