

# Sound and Complete Mutation-Based Program Repair

Bat-Chen Rothenberg and Orna Grumberg

Formal Methods (FM'16)

Lecture 2 – 3.8.2017

# Mutation-Based Program Repair

Sequential  
program

Assertions  
in code

Given set  
of  
mutations

Can we use  
these  
mutations to  
make all  
assertions  
hold?

Assignments,  
conditionals,  
loops and  
function calls



Assertion  
violation

operator  
replacement  
( $+$   $\rightarrow$   $-$ ),  
constant  
manipulation  
( $c \rightarrow c + 1$ )

Return  
all  
possible  
repairs

# Example

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```

$x = 5, y = 2$

$z = 9$

$z = 8$



# Example

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z + 1;  
8.   assert(z > 8);  
9.   return z;  
}
```

At  
this  
point  
 $z \geq 9$



Mutation list:

- Replace + with -
- Replace - with +
- Replace > with  $\geq$
- Replace  $\geq$  with >

Repair list:

option 1:

line 7: replace  $\geq$  with >

option 2:

line 7: replace - with +

Note:  
Repairs  
are  
**minimal**

# Example

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 9) {  
3.       z = x + y;  
4.   } else {  
5.       z = 10;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```

At this  
point  
 $z \geq 10$



Mutation list:

Replace + with -

Replace - with +

Replace > with  $\geq$

Replace  $\geq$  with >

Increase constants by 1

# Program Repair

- **Programs:** sequential C programs
- **Specification:** Assertions added in program text
- **Bug:** A program run which violates an assertion
- **Repair:** Changed statement(s) in the program, resulting in a correct program
- **No assumption on the number of faulty expressions.**
- **Goal:** To return a sequence of repaired (correct) programs in increasing number of changes using **techniques and tools of formal methods**

# Program Bug Repair

In existing works:

- Changes allowed for fixing a bug:
  - Insertions, deletions, replacements
- Where?
  - Everywhere, only function calls, only RHS of assignments, etc.
- How many?
  - Single fault assumption

# Program Repair – our work

- We consider:
  - **Expression replacements**
    - I.e. RHS of assignments, conditions and function arguments
  - Replacements from a fixed list of possible changes, called **mutations**.
    - E.g., operator replacement ( $+ \rightarrow -$ ), constant manipulation ( $c \rightarrow c + 1$ ), variable replacement ( $x \rightarrow y$ ).
  - **No assumption** on the number of faulty expressions.

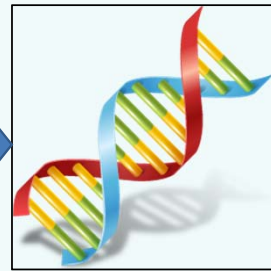


# Overview of our approach

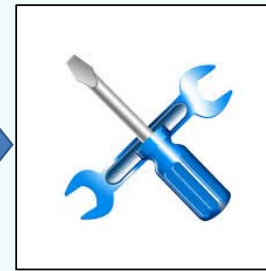
```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```



Translation



Mutation



Repair

```
line 7: replace operator ≥ with >  
line 7: replace operator - with +  
...
```

Output:

All **minimal** repairs,  
sorted by size

Input:  
a buggy  
program

Finding all **unsatisfiable constraint sets**  
from a finite set of **programs**



# First step - Translation

Goal: Translate the program into a **set of constraints** which is **satisfiable iff the program has a bug**  
(i.e. there exists an input for which an assertion fails)

Work by Clarke, Kroening, Lerda (TACAS 2000)  
(CBMC)

- Simplification
- Unwinding of loops
  - a **bounded** number of unwinding
- Conversion to SSA

Correctness  
is bounded

# First step - Translation

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```



```
{ g1 = x1 + y1 > 8,  
  z2 = x1 + y1,  
  z3 = 9,  
  z4 = g1? z2: z3,  
  b1 = z4 ≥ 9 ,  
  z5 = z4 - 1,  
  z6 = b1? z5: z4,  
  z6 ≤ 8  
}
```

[Clarke et al. 2004 (CBMC)]



```
int f(int x, int y){
1.   int z;
2.   if (x + y > 8) {
3.       z = x + y;
4.   } else {
5.       z = 9;
6.   }
7.   if (z ≥ 9) z = z - 1;
8.   assert(z > 8);
9.   return z;
}
```

Conversion  
to SSA

```
int f(int x1, int y1){
1.   int z1;
2.1  bool g1 = x1 + y1 > 8;
2.2  if (g1) {
3.       z2 = x1 + y1;
4.   } else {
5.       z3 = 9;
6.   }
-    z4 = g1?z2:z3;
7.1  bool b1 = z4 ≥ 9
7.2  if (b1) z5 = z4 - 1;
-    z6 = b1?z5:z4;
8.   assert(z6 > 8);
9.   return z6;
}
```

simplification

```
int f(int x, int y){
1.   int z;
2.1  bool g = x + y > 8;
2.2  if (g) {
3.       z = x + y;
4.   } else {
5.       z = 9;
6.   }
7.1  bool b = z ≥ 9
7.2  if (b) z = z - 1;
8.   assert(z > 8);
9.   return z;
}
```

```
{ g1 = x1 + y1 > 8,
z2 = x1 + y1,
z3 = 9,
z4 = g1?z2:z3,
b1 = z4 ≥ 9,
z5 = z4 - 1,
z6 = b1?z5:z4,
z6 ≤ 8
}
```

# First step - Translation

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```



```
{ g1 = x1 + y1 > 8,  
  z2 = x1 + y1,  
  z3 = 9,  
  z4 = g1? z2: z3,  
  b1 = z4 ≥ 9,  
  z5 = z4 - 1,  
  z6 = b1? z5: z4,  
  z6 ≤ 8  
}
```

[Clarke et al. 2004 (CBMC)]

# First step - Translation

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```



```
{ g1 = x1 + y1 > 8,  
  z2 = x1 + y1,  
  z3 = 9,  
  z4 = g1? z2: z3,  
  b1 = z4 ≥ 9,  
  z5 = z4 - 1,  
  z6 = b1? z5: z4,  
  z6 ≤ 8  
}
```

[Clarke et al. 2004 (CBMC)]

# First step - Translation

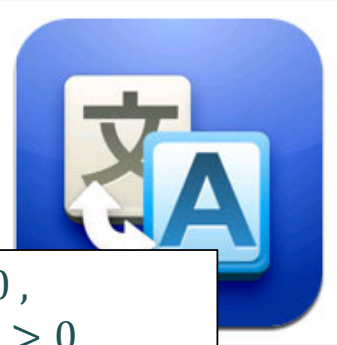
```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   if (z ≥ 9) z = z - 1;  
8.   assert(z > 8);  
9.   return z;  
}
```



```
{  $g_1 = x_1 + y_1 > 8,$   
   $z_2 = x_1 + y_1,$   
   $z_3 = 9,$   
   $z_4 = g_1 ? z_2 : z_3,$   
   $b_1 = z_4 \geq 9,$   
   $z_5 = z_4 - 1,$   
   $z_6 = b_1 ? z_5 : z_4,$   
   $z_6 \leq 8$   
}
```

[Clarke et al. 2004 (CBMC)]

# What about loops?



```
int z = 0;
while(x > 0){
    z = z + x;
    x = x - 1;
}
```

simplification

```
int z = 0;
bool t = x > 0;
while(t){
    z = z + x;
    x = x - 1;
    t = x > 0;
}
```

unwinding  
(for  $b = 2$ )

```
int z = 0;
bool t = x > 0;
if(t){
    z = z + x;
    x = x - 1;
    t = x > 0;
    if(t){
        z = z + x;
        x = x - 1;
        t = x > 0;
        assume(!t);
    }
}
```

```
{ z1 = 0,
  t1 = x1 > 0,
  z2 = z1 + x1},
  x2 = x1 - 1,
  t2 = x2 > 0,
  z3 = z2 + x2},
  x3 = x2 - 1,
  t3 = x3 > 0,
  t1 ∧ t2 → !t3},
  z4 = t2?z3:z2},
  x4 = t2?x3:x2},
  z5 = t1?z4:z1},
  x5 = t1?x4:x1}
}
```



# Translation

- In the translation, loops are unwound a **bounded** number of times
- **Important observation: correctness is bounded.**  
That is, repairs found by our method only guarantee that assertions cannot be violated by inputs going through the loop at most  **$k$  times**

# More complex example

```
int f(int x, int y){  
1.     int z;  
2.     if (x + y > 8) {  
3.         z = x + y;  
4.     } else {  
5.         z = 9;  
6.     }  
7.     while(x > 0){  
8.         z = z + x;  
9.         x = x - 1;  
10.    }  
11.    if (z ≥ 9) z = z - 1;  
12.    assert(z > 8);  
13.    return z;  
}
```



```
{ g1 = x1 + y1 > 8,  
  z2 = x1 + y1,  
  z3 = 9,  
  z4 = g1?z2:z3,  
  t1 = x1 > 0,  
  z5 = z4 + x1,  
  x2 = x1 - 1,  
  t2 = x2 > 0,  
  z6 = z5 + x2,  
  x3 = x2 - 1,  
  t3 = x3 > 0,  
  t1 ∧ t2 → !t3,  
  z7 = t2?z6:z5,  
  z8 = t1?z7:z4,  
  b1 = z8 ≥ 9,  
  z9 = z8 - 1,  
  z10 = b1?z9:z8,  
  z10 ≤ 8  
}
```

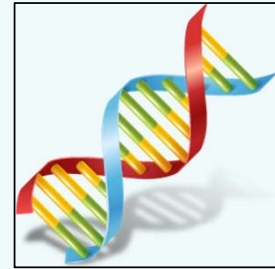
Let's see how mutations to the program affect the set of constraints

```
int f(int x, int y){  
1.     int z;  
2.     if (x + y > 8) {  
3.         z = x + y;  
4.     } else {  
5.         z = 9;  
6.     }  
7.     while(x > 0){  
8.         z = z - x;  
9.         x = x - 1;  
10.    }  
11.    if (z ≥ 9) z = z - 1;  
12.    assert(z > 8);  
13.    return z;  
}
```



```
{ g1 = x1 + y1 > 8,  
  z2 = x1 + y1,  
  z3 = 9,  
  z4 = g1?z2:z3,  
  t1 = x1 > 0,  
  z5 = z4 - x1,  
  x2 = x1 - 1,  
  t2 = x2 > 0,  
  z6 = z5 - x2,  
  x3 = x2 - 1,  
  t3 = x3 > 0,  
  t1 ∧ t2 → !t3,  
  z7 = t2?z6:z5,  
  z8 = t1?z7:z4,  
  b1 = z8 ≥ 9,  
  z9 = z8 - 1,  
  z10 = b1?z9:z8,  
  z10 ≤ 8  
}
```

# Second step- Mutation



$$\begin{cases}
 g_1 = x_1 + y_1 > 8, \\
 z_2 = x_1 + y_1, \\
 z_3 = 9, \\
 z_4 = g_1?z_2:z_3, \\
 t_1 = x_1 > 0, \\
 z_5 = z_4 + x_1, \\
 x_2 = x_1 - 1, \\
 t_2 = x_2 > 0, \\
 z_6 = z_5 + x_2, \\
 x_3 = x_2 - 1, \\
 t_3 = x_3 > 0, \\
 t_1 \wedge t_2 \rightarrow !t_3, \\
 z_7 = t_2?z_6:z_5, \\
 z_8 = t_1?z_7:z_4, \\
 b_1 = z_8 \geq 9, \\
 z_9 = z_8 - 1, \\
 z_{10} = b_1?z_9:z_8, \\
 z_{10} \leq 8 \\
 \}
 \end{cases}$$



$S_{soft}$

$$\begin{cases}
 g_1 = x_1 + y_1 > 8, \\
 z_2 = x_1 + y_1, \\
 z_3 = 9, \\
 t_1 = x_1 > 0, \\
 z_5 = z_4 + x_1, \\
 x_2 = x_1 - 1, \\
 t_2 = x_2 > 0, \\
 z_6 = z_5 + x_2, \\
 x_3 = x_2 - 1, \\
 t_3 = x_3 > 0, \\
 b_1 = z_8 \geq 9, \\
 z_9 = z_8 - 1 \\
 \}
 \end{cases}$$



$$\begin{cases}
 g_1 = x_1 + y_1 > 8, \\
 z_2 = x_1 + y_1, \\
 z_3 = 9, \\
 t_1 = x_1 > 0 \wedge t_2 = x_2 > 0 \wedge t_3 = x_3 > 0, \\
 z_5 = z_4 + x_1 \wedge z_6 = z_5 + x_2, \\
 x_2 = x_1 - 1 \wedge x_3 = x_2 - 1, \\
 b_1 = z_8 \geq 9, \\
 z_9 = z_8 - 1 \\
 \}
 \end{cases}$$

$$\begin{cases}
 z_4 = g_1?z_2:z_3, \\
 t_1 \wedge t_2 \rightarrow !t_3, \\
 z_7 = t_2?z_6:z_5, \\
 z_8 = t_1?z_7:z_4, \\
 z_{10} = b_1?z_9:z_8, \\
 z_{10} \leq 8 \\
 \}
 \end{cases}$$



$$g_1 = x_1 + y_1 > 8$$

## Second step - Mutation

Mutation list:  
 Replace + with -  
 Replace - with +  
 Replace > with ≥  
 Replace ≥ with >

```

int f(int x, int y){
1.   int z;
2.   if (x + y > 8) {
3.       z = x - y;
4.   } else {
5.       z = 9;
6.   }
7.   if (z ≥ 9) {
           z = z - 1;
           }
8.   assert(z > 8);
9.   return z;
}
```

$\{g_1 = x_1 + y_1 > 8, g_1 = x_1 + y_1 \geq 8, g_1 = x_1 + y_1 > 8, g_1 = x_1 + y_1 \geq 8\}$

$\{z_2 = x_1 - y_1, z_2 = x_1 - y_1\}$

$\{z_3 = 9\}$

$z_4 = g_1 ? z_2 : z_3$

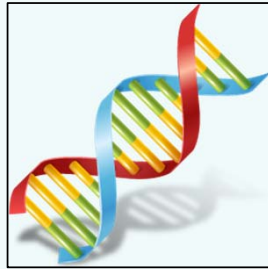
$\{b_1 = z_4 \geq 9, b_1 = z_4 > 9\}$

$\{z_5 = z_4 - 1, z_5 = z_4 + 1\}$

$z_6 = b_1 ? z_5 : z_4$

$z_6 \leq 8$

# Second step- Mutation



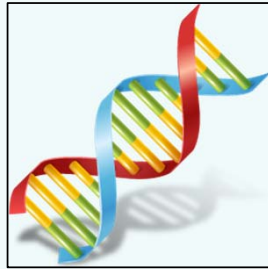
$$\begin{cases}
 g_1 = x_1 + y_1 > 8, \\
 z_2 = x_1 + y_1, \\
 z_3 = 9, \\
 \mathbf{t_1 = x_1 > 0 \wedge t_2 = x_2 > 0 \wedge t_3 = x_3 > 0}, \\
 z_5 = z_4 + x_1 \wedge z_6 = z_5 + x_2, \\
 x_2 = x_1 - 1 \wedge x_3 = x_2 - 1, \\
 b_1 = z_8 \geq 9, \\
 z_9 = z_8 - 1 \\
 \}
 \end{cases}$$



$$\begin{aligned}
 S_1 &= \{ g_1 = x_1 + y_1 > 8, g_1 = x_1 - y_1 > 8, \\
 &\quad g_1 = x_1 + y_1 \geq 8 \} \\
 S_2 &= \{ z_2 = x_1 + y_1, z_2 = x_1 - y_1 \} \\
 S_3 &= \{ z_3 = 9 \} \\
 S_4 &= \{ \mathbf{t_1 = x_1 > 0 \wedge t_2 = x_2 > 0 \wedge t_3 = x_3 > 0}, \\
 &\quad \mathbf{t_1 = x_1 \geq 0 \wedge t_2 = x_2 \geq 0 \wedge t_3 = x_3 \geq 0} \} \\
 S_5 &= \{ z_5 = z_4 + x_1 \wedge z_6 = z_5 + x_2, \\
 &\quad z_5 = z_4 - x_1 \wedge z_6 = z_5 - x_2 \} \\
 S_6 &= \{ x_2 = x_1 - 1 \wedge x_3 = x_2 - 1, \\
 &\quad x_2 = x_1 + 1 \wedge x_3 = x_2 + 1 \} \\
 S_7 &= \{ b_1 = z_8 \geq 9, b_1 = z_8 > 9 \} \\
 S_8 &= \{ z_9 = z_8 - 1, z_9 = z_8 + 1 \}
 \end{aligned}$$

Replace + with -  
 Replace - with +  
 Replace > with ≥  
 Replace ≥ with >

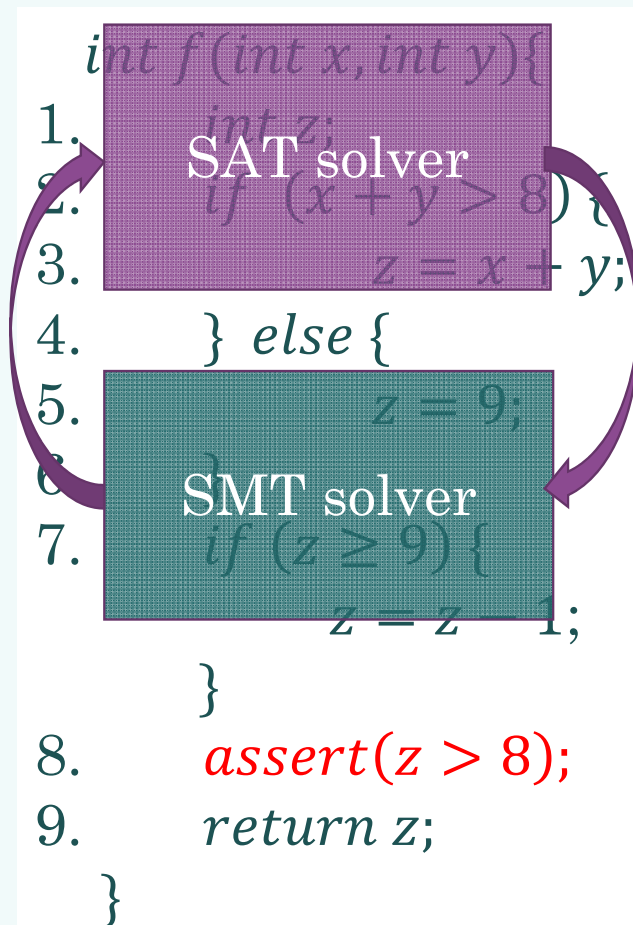
## Second step- Mutation



We have **reduced** the problem of **finding a correct program** from a finite set of mutated programs to the problem of **choosing one constraint from each  $S_i$**  such that the conjunction of all chosen constraints and all constraints in  $S_{hard}$  is **unsatisfiable**.



# Third step - Repair



$\{g_1 = x_1 + y_1 > 8, g_1 = x_1 - y_1 > 8, g_1 = x_1 + y_1 \geq 8\}$

$\{z_2 = x_1 + y_1, z_2 = x_1 - y_1\}$

$\{z_3 = 9\}$

$z_4 = g_1 ? z_2 : z_3$

$\{b_1 = z_4 \geq 9, b_1 = z_4 > 9\}$

$\{z_5 = z_4 - 1, z_5 = z_4 + 1\}$

$z_6 = b_1 ? z_5 : z_4$

$z_6 \leq 8$





# Repair

SAT solver

Choose candidate program of **size = 1**  
 Blocking clause for specific assignment  
 Blocking clause for this assignment  
 And all other supersets of changes

SAT  
 $c_1 = 0$   
 $c_2 = 0$   
 $c_3 = 0$   
 $c_4 = 1$   
 ~~$c_5 = 0$~~   
 $c_6 = 1$   
 $c_7 = 0$   
 $c_8 = 0$   
 $c_9 = 1$   
 $c_{10} = 0$

SMT solver

$g_1 = x_1 + y_1 > 8$   
 $z_4 = g_1? z_2 : z_3$   
 $z_2 = x_1 + y_1$   
 $z_3 = 9$   
 $z_6 = b_1? z_5 : z_4$   
 $b_1 = z_4 \geq 9$   
 $z_6 \leq 8$   
 $z_5 = z_4 - 1$

UNSAT  
 ((repair  
 failed!))

- $c_1$  {  $g_1 = x_1 + y_1 > 8$ ,  $g_1 = x_1 - y_1 > 8$ ,  $g_1 = x_1 + y_1 \geq 8$  }
- $c_2$
- $c_3$
- $c_4$  {  $z_2 = x_1 + y_1$ ,  $z_2 = x_1 - y_1$  }
- $c_5$
- $c_6$  {  $z_3 = 9$  }
- $c_7$  {  $b_1 = z_4 \geq 9$ ,  $b_1 = z_4 > 9$  }
- $c_8$
- $c_9$  {  $z_5 = z_4 - 1$ ,  $z_5 = z_4 + 1$  }
- $c_{10}$

# Final step- Repair



$S_1 = \{1, 2, 3\}$   
 $S_2 = \{4, 5\}$   
 $S_3 = \{6\}$   
 $S_4 = \{7, 8\}$   
 $S_5 = \{9, 10\}$   
 $S_6 = \{11, 12\}$   
 $S_7 = \{13, 14\}$   
 $S_8 = \{15, 16\}$

- Each set  $S_i$  contains a **constraint**  $c_o^i$ , encoding the original (unmutated) statement (marked in **red** in the example)
- A **selection vector (SV)** is a vector of constraints  $[c_1, \dots, c_n]$  where  $c_i$  is taken from  $S_i$  for all  $1 \leq i \leq n$ . For example:  $v' = [2, 4, 6, 7, 9, 12, 13, 15]$
- **Note that each SV “encodes” a program**
- a SV is said to be **correct** if it encodes a (bounded) correct program.

# Final step- Repair

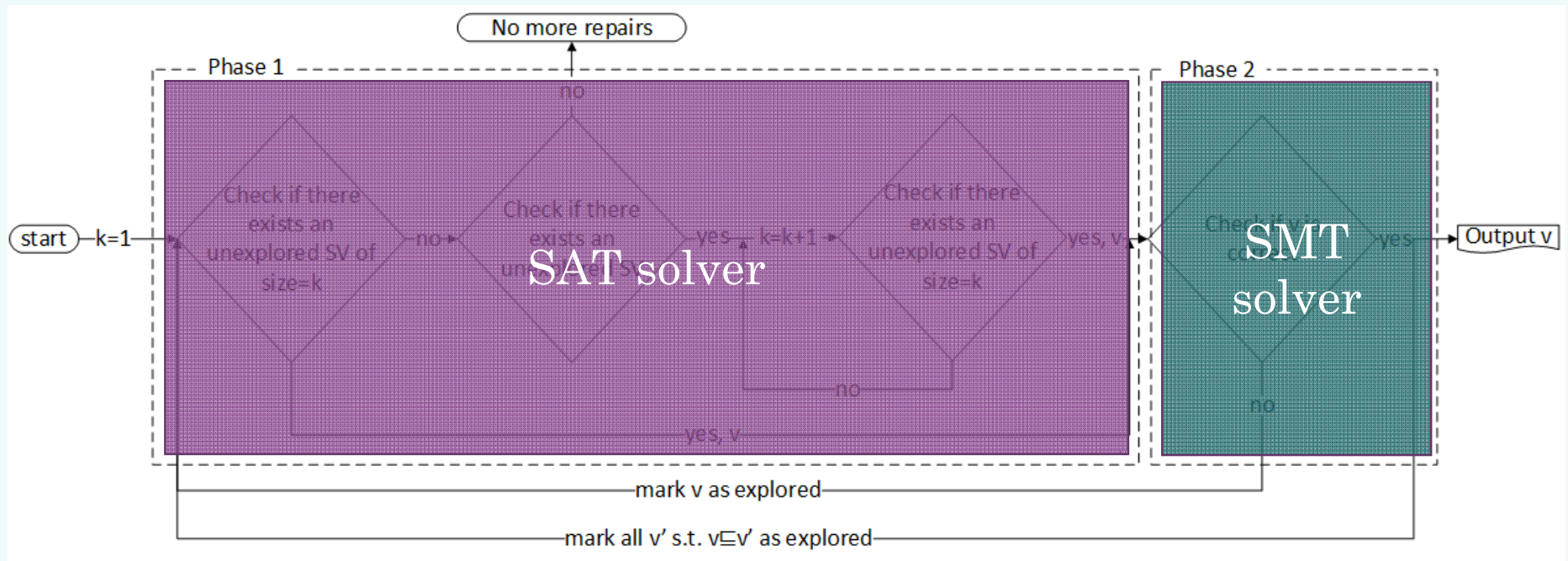


$$\begin{aligned} S_1 &= \{1, 2, 3\} \\ S_2 &= \{4, 5\} \\ S_3 &= \{6\} \\ S_4 &= \{7, 8\} \\ S_5 &= \{9, 10\} \\ S_6 &= \{11, 12\} \\ S_7 &= \{13, 14\} \\ S_8 &= \{15, 16\} \end{aligned}$$

- The **size** of a selection vector  $v$  is the number of sets from which the chosen constraint is different than  $c_o^i$ , i.e. the number of lines changed in the program encoded by  $v$ . For example:  $\text{size}(v') = 2$
- Size order:  $[2, 4, 6, 7, 9, 12, 13, 15] \sqsubseteq [2, 4, 6, 7, 9, 12, 14, 15]$   
but  $[2, 4, 6, 7, 9, 12, 13, 15] \not\sqsubseteq [3, 4, 6, 7, 9, 12, 14, 15]$
- A SV  $v$  is a **Minimal correct SV (MCSV)** if  $v$  is a correct SV and there does not exist a SV  $v'$  s.t.  $v' \sqsubseteq v$ .
- **We would like to return all minimal SVs in increasing size order.**

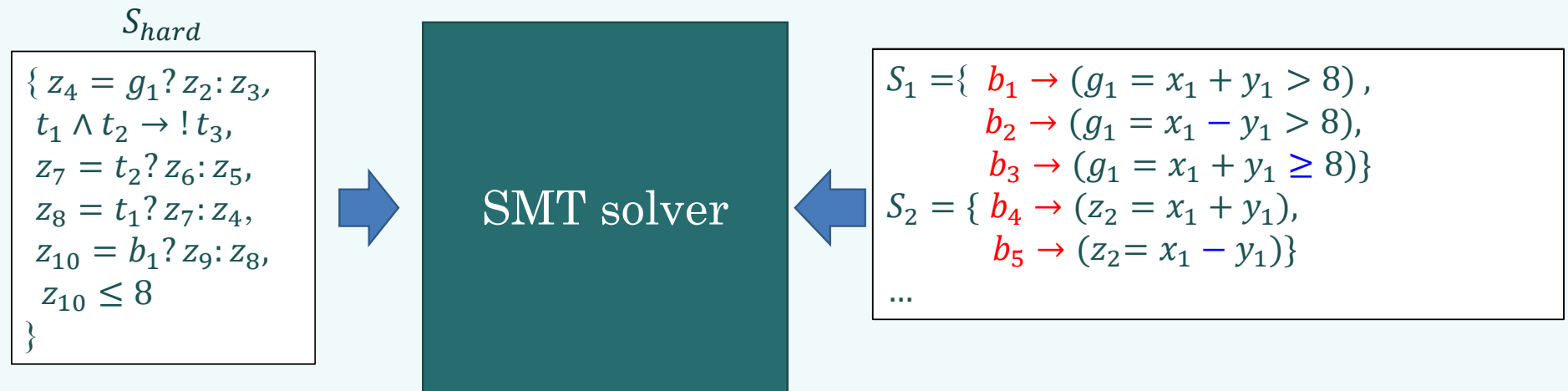


# Repair scheme





# Initialization of the SMT solver



The formula solved by the SAT solver will be over the Boolean variables we added.



# Initialization of the SAT solver

Every satisfying assignment corresponds to a SV

$$\begin{aligned} b_1 + b_2 + b_3 = 1 \wedge \\ b_4 + b_5 = 1 \wedge \\ \dots \end{aligned}$$



All SV's returned are of size  $\leq k$

$$b_1 + b_4 + \dots \geq n - k$$

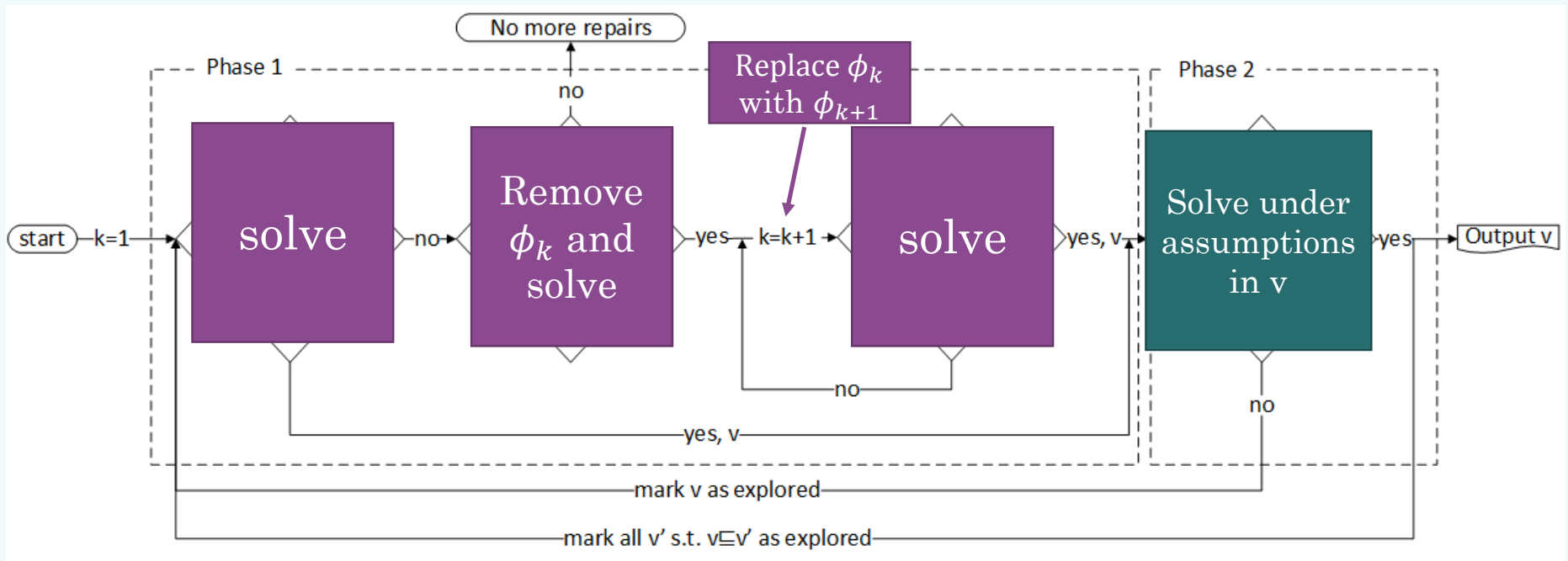


This constraint, called **cardinality constraint**, will be referred to as  $\phi_k$

$$\begin{aligned} S_1 = \{ & b_1 \rightarrow (g_1 = x_1 + y_1 > 8), \\ & b_2 \rightarrow (g_1 = x_1 - y_1 > 8), \\ & b_3 \rightarrow (g_1 = x_1 + y_1 \geq 8) \} \\ S_2 = \{ & b_4 \rightarrow (z_2 = x_1 + y_1), \\ & b_5 \rightarrow (z_2 = x_1 - y_1) \} \\ & \dots \end{aligned}$$



# Repair scheme



# Results on our example

Program

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y > 8) {  
3.       z = x + y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   while(x > 0){  
8.       z = z + x;  
9.       x = x - 1;  
10.  }  
11.  if (z ≥ 9) z = z - 1;  
12.  assert(z > 8);  
13.  return z;  
}
```

List of mutations

Replace + with -  
Replace - with +  
Replace > with ≥  
Replace ≥ with >

Results

line 11: replace operator ≥ with >  
line 11: replace - with +

Unwinding bound (not including)

3



# Results on our example

## Program

```
int f(int x, int y){  
1.   int z;  
2.   if (x + y ≥ 8) {  
3.       z = x - y;  
4.   } else {  
5.       z = 9;  
6.   }  
7.   while(x > 0){  
8.       z = z + x;  
9.       x = x - 1;  
10.  }  
11.  if (z ≥ 9) z = z - 1;  
12.  assert(z > 8);  
13.  return z;  
}
```

## List of mutations

Replace + with -  
Replace - with +  
Replace > with ≥  
Replace ≥ with >



## Results

line 2: replace ≥ with > *and*  
line 3 replace - with + *and*  
line 11 replace - with +  
line 2: replace ≥ with > *and*  
line 3 replace - with + *and*  
line 11 replace ≥ with >

## Unwinding bound (not including)

3

Theorem:

Our algorithm is sound and complete.

That is, for a given bound  $\mathbf{b}$ :

A program is returned by our algorithm  
iff

it is **minimal** and  **$\mathbf{b}$ -bounded correct**

- Minimal number of changes
- Every assertion reachable along a computation of bounded length  $\mathbf{b}$  is correct

Ver.	Method of [11]		Method of [12]		Our method			
	Fixed?	Time[s]	Fixed?	Time[s]	Fixed?	Time[s]	Fixed?	Time[s]

		Level 1	Level 2
Op. replacement	Arithmetic	{+, -}, {*, /, %}	{+, -, *, /, %}
	Relational	{>, >=}, {<, <=}	{>, >=, <, <=}, {==, !=}
	Logical	{  , &&}	
	Bit-wise	{>>, <<}, {&,  , ^}	
Constant manipulation			C → C+1, C → C-1, C → -C, C → 0

28	+	34	+	35		+	93.678
31					+	1.246	4.661
32					+	1.902	85.349
35	+	41	+	46		+	92.866
36	+	8	+	6		+	94.599
39	+	82	+	101	+	2.558	16.393
40							1.890

implemented in the tool FoREnSiC.

16 (39%)	38	15 (36.6%)	38	11 (26.83%)	2.278	18 (43.9%)	48.151
----------	----	------------	----	-------------	-------	------------	--------

# Summary

- We suggest a repair method which returns **all minimal** (bounded) correct programs, in **increasing order**
  - Based on a given set of mutations
- If no repaired program is returned then the given mutations **cannot repair** the program
- SAT solver handles the search of mutated programs
- SMT solver checks if a mutated program is correct
- Both solvers are used **incrementally**

# Summary

- Minimal mutations: No change is made to the original program unless necessary
- The method can assist a programmer in debugging in initial stages of development
  - When bug are simple, but many

Questions?