Model Checking and Its Applications

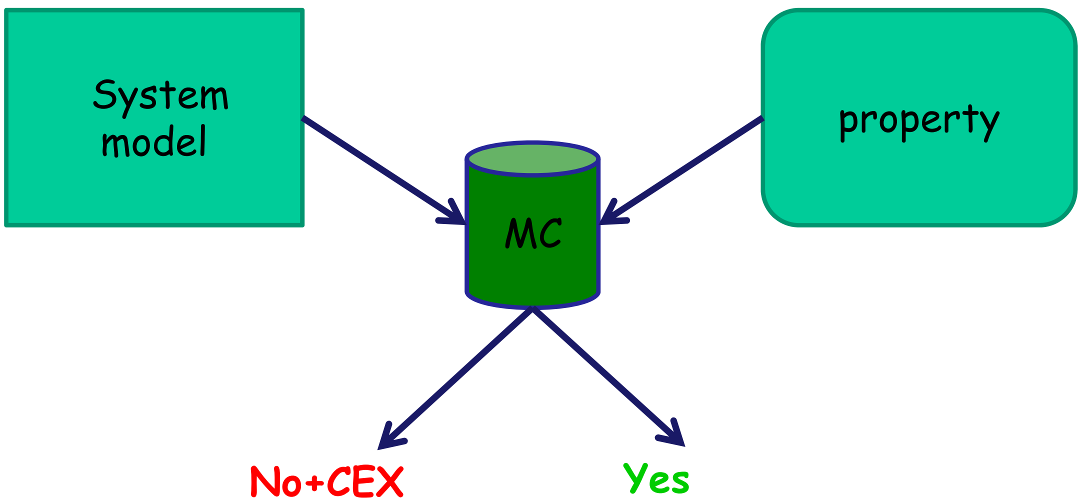
Orna Grumberg
Technion, Israel

Lecture 1

Summer School of Formal Techniques (SSFM) 2019

Model Checking

- Given a system and a specification, does the system satisfy the specification.



Challenges in model checking

Model checking is successfully used, but…

- Scalability
- New types of systems
- New specifications (e.g. security)
- Applications in new areas

Technologies to help

Developed or adapted by the MC community

- SAT and SMT solvers
- Static analysis
- Abstraction - refinement
- Compositional verification
- Machine learning, automata learning

And many more...

In these talks

- We show how to exploit concepts and technologies from model checking to assist in stages of program development
  - Automatic program repair
  - Program difference

Sound and Complete Mutation-Based Program Repair

Bat-Chen Rothenberg and Orna Grumberg

Formal Methods (FM'16)

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

Return all possible repairs

Example

```
int f(int x, int y){
        int z;
        if  (x + y > 8) {
                z = x + y;
        } else {
                z = 9;
        }
        if (z ≥ 9)  z = z − 1;
        assert(z > 8);
        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 ≥
- Replace ≥ with >

Note: Repairs are **minimal**

Repair list:

option 1:

line 7: *replace* $\geq$ *with* $>$

option 2:

line 7: *replace* $-$ *with* $+$

Example

```
int f(int x, int y){
      int z;
      if (x + y > 9) {
            z = x + y;
      } else {
            z = 10;
      }
      if (z ≥ 9) z = z − 1;
      assert(z > 8);
      return z;
}
```

At this point $z \geq 10$

Mutation list:

- Replace + with –
- Replace – with +
- Replace > with ≥
- Replace ≥ 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**

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;
   }
```

Input:
a buggy
program

Translation

Mutation

Repair

line 7: replace operator $\geq$ *with* $>$
line 7: replace operator $-$ *with* $+$
…

Output:
All **minimal** repairs,
sorted by size

Finding all correct programs
from a finite set of programs

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

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 20
(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){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9)  z = z − 1;
    assert(z > 8);
    return z;
}
```

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

[Clarke et al. 2004 (CBMC)]

$int\ f(int\ x, int\ y)\{$

1. $int\ z;$
2. $if\ \ (x + y > 8)\ \{$
3. $\quad z = x + y;$
4. $\}\ else\ \{$
5. $\quad z = 9;$
6. $\}$
7. $if\ (z \geq 9)\ \ z = z - 1;$
8. $assert(z > 8);$
9. $return\ z;$

$\}$

simplification

$int\ f(int\ x, int\ y)\{$

1. $int\ z;$

2.1 $bool\ g = x + y > 8;$

2.2 $if\ \ (g)\ \{$

3. $\quad z = x + y;$
4. $\}\ else\ \{$
5. $\quad z = 9;$
6. $\}$

7.1 $bool\ b = z \geq 9$

7.2 $if\ (b)\ \ z = z - 1;$

8. $assert(z > 8);$
9. $return\ z;$

$\}$

Conversion to SSA

$int\ f(int\ x_1, int\ y_1)\{$

1. $int\ z_1;$

2.1 $bool\ g_1 = x_1 + y_1 > 8;$

2.2 $if\ \ (g_1)\ \{$

3. $\quad z_2 = x_1 + y_1;$
4. $\}\ else\ \{$
5. $\quad z_3 = 9;$
6. $\}$

\- $z_4 = g_1 ?\, z_2 : z_3;$

7.1 $bool\ b_1 = z_4 \geq 9$

7.2 $if\ (b_1)\ \ z_5 = z_4 - 1;$

\- $z_6 = b_1 ?\, z_5 : z_4;$

8. $assert(z_6 > 8);$
9. $return\ z_6;$

$\}$

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

$z_2 = x_1 + y_1,$

$z_3 = 9,$

$z_4 = g_1 ?\, z_2 : z_3,$

$b_1 = z_4 \geq 9\,,$

$z_5 = z_4 - 1,$

$z_6 = b_1 ?\, z_5 : z_4,$

$z_6 \leq 8$

$\}$

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);
        }
}
```

$\{ \; z_1 = 0\,,$
$t_1 = x_1 > 0,$
$z_2 = z_1 + x_1,$
$x_2 = x_1 - 1,$
$t_2 = x_2 > 0,$
$z_3 = z_2 + x_2,$
$x_3 = x_2 - 1,$
$t_3 = x_3 > 0,$
$t_1 \wedge t_2 \rightarrow\, !\, t_3,$
$z_4 = t_2 ?\, z_3 : z_2,$
$x_4 = t_2 ?\, x_3 : x_2,$
$z_5 = t_1 ?\, z_4 : z_1,$
$x_5 = t_1 ?\, x_4 : x_1$
$\}$

First step - Translation

```
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9)  z = z − 1;
    assert(z > 8);
    return z;
}
```

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

[Clarke et al. 2004 (CBMC)]

First step - Translation

```
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9)  z = z − 1;
    assert(z > 8);
    return z;
}
```

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

[Clarke et al. 2004 (CBMC)]

First step - Translation

$$int\ f(int\ x, int\ y)\{$$
$$int\ z;$$
$$if\ \ (x + y > 8)\ \{$$
$$z = x + y;$$
$$\}\ else\ \{$$
$$\mathbf{z = 9};$$
$$\}$$
$$if\ (z \geq 9)\ \ z = z - 1;$$
$$assert(z > 8);$$
$$return\ z;$$
$$\}$$

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

[Clarke et al. 2004 (CBMC)]

First step - Translation

```
int f(int x, int y){
    int z;
    if  (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9)  z = z − 1;
    assert(z > 8);
    return z;
}
```

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

[Clarke et al. 2004 (CBMC)]

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){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    while(x > 0){
        z = z + x;
        x = x − 1;
    }
    if (z ≥ 9)  z = z − 1;
    assert(z > 8);
    return z;
}
```

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

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

```
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    while(x > 0){
        z = z - x;
        x = x - 1;
    }
    if (z ≥ 9)  z = z - 1;
    assert(z > 8);
    return z;
}
```

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

Second step- Mutation

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

S_{soft}

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

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

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

$g_1 = x_1 + y_1 > 8$

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;
   }
```

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

$g_1 = x_1 + y_1 > 8$

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

$\{ z_3 = 9\}$

$z_4 = g_1 ? z_2 : z_3$

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

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

$z_6 = b_1 ? z_5 : z_4$

$z_6 \leq 8$

Second step- Mutation

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

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

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** $\boldsymbol{S_i}$ such that the conjunction of all chosen constraints and all constraints in $\boldsymbol{S_{hard}}$ is **unsatisfiable**.

Third step - Repair

```
int f(int x, int y){
    int z;
    if (x + y > 8) {
        z = x + y;
    } else {
        z = 9;
    }
    if (z ≥ 9) {
        z = z − 1;
    }
    assert(z > 8);
    return z;
}
```

SAT solver

SMT solver

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

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

$\{ z_3 = 9\}$

$z_4 = g_1 ? z_2 : z_3$

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

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

$z_6 = b_1 ? z_5 : z_4$

$z_6 \leq 8$

Repair

c_1 c_2

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

c_3

c_4 c_5

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

c_6

$\{z_3 = 9\}$

c_7 c_8

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

c_9 c_{10}

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

SAT solver

Choose candidate program of ***size = 1***

Blocking clause for specific assignment

Blocking clause for this assignment

And ***all other supersets of changes***

UNSAT (repair found!)

SAT (not a repair)

SMT solver

$z_4 = g_1 ? z_2 : z_3$

$z_6 = b_1 ? z_5 : z_4$

$z_6 \leq 8$

$g_1 = x_1 + y_1 > 8$

$z_2 = x_1 + y_1$

$z_3 = 9$

$b_1 = z_4 \geq 9$

$z_5 = z_4 - 1$

SAT

$c_1 = 1$

$c_2 = 0$

$c_3 = 0$

$c_4 = 1$

$c_5 = 0$

UNSAT

$c_6 = 1$

$c_7 = 0$

$c_8 = 0$

$c_9 = 1$

$c_{10} = 0$

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 $[c_1, \ldots, c_n]$ 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

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

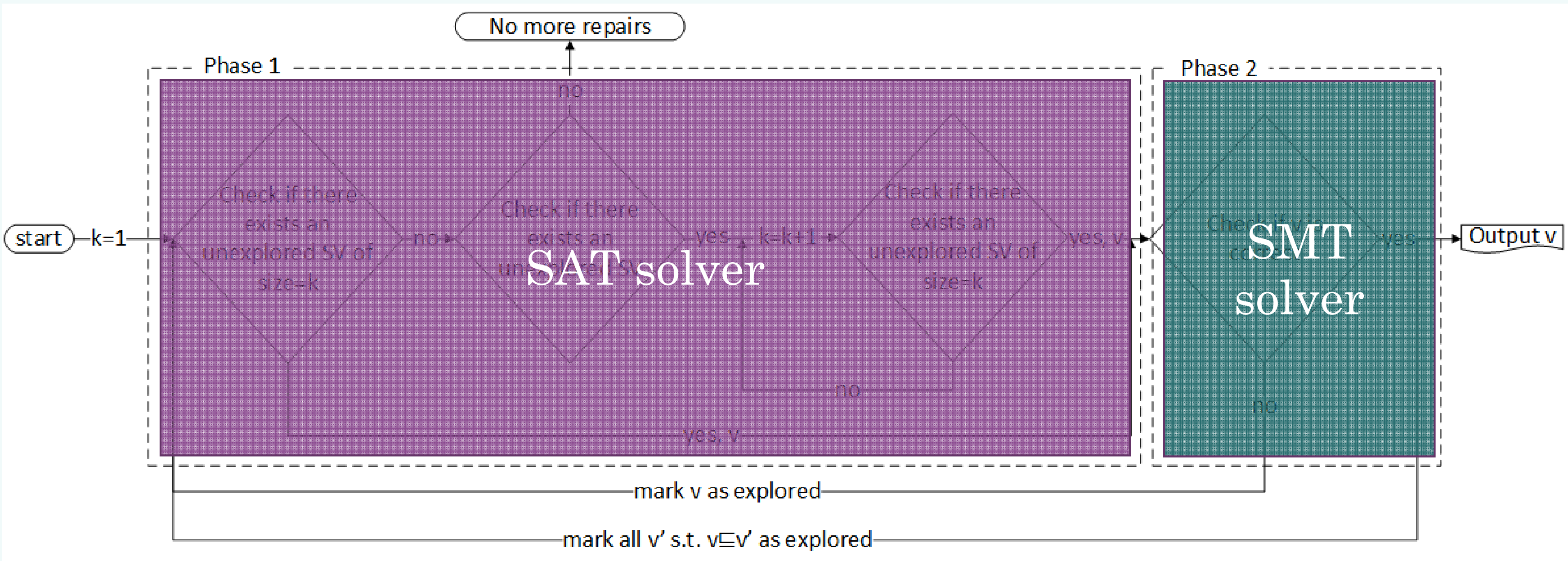
- The **size** of a selection vector v is the number of sets from which the chosen constraint is different than c_o^i, i.e. the number of lines changed in the program encoded by v. For example: size$(v') = 2$

- 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

S_{hard}

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

SMT solver

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

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

$$b_1 + b_2 + b_3 = 1 \ \wedge$$
$$b_4 + b_5 = 1 \ \wedge$$
$$\ldots$$

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

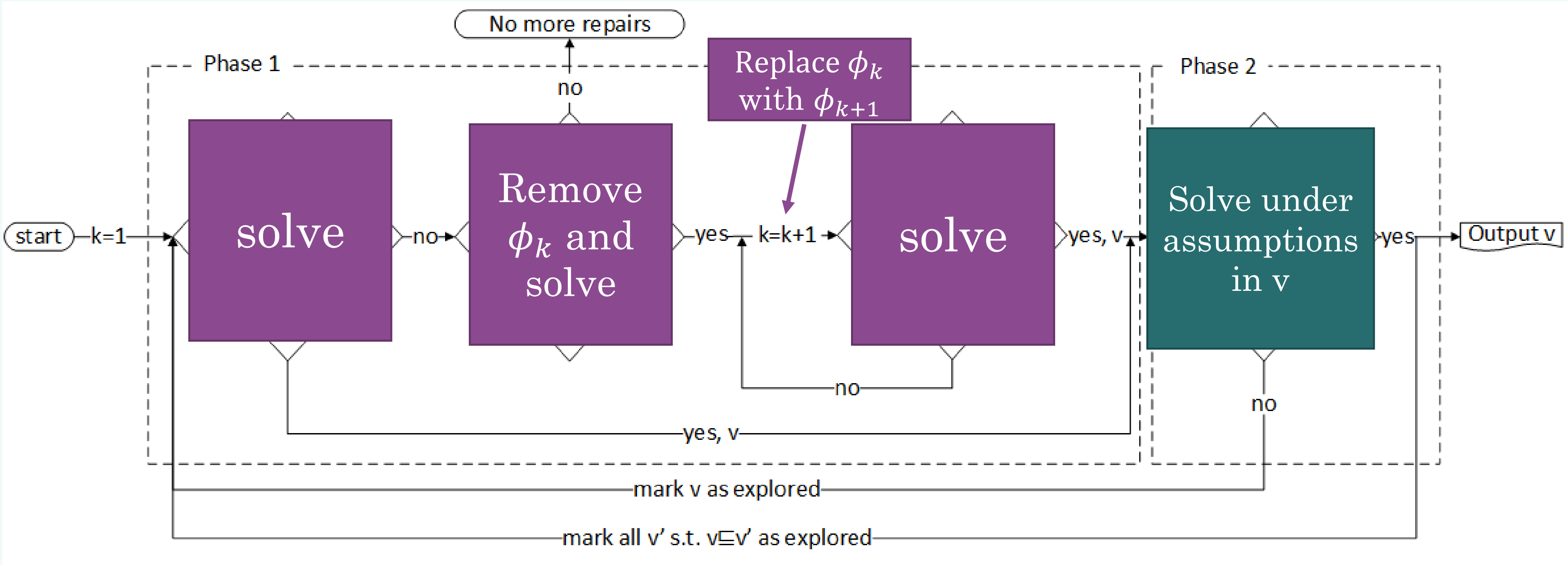
$$b_1 + b_4 + \cdots \geq n - k$$

SAT solver

This constraint, called cardinality constraint, will be referred to as ϕ_k

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

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 11: *replace operator* $\geq$ *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 >

Unwinding bound (not including)

3

Results

line 2: *replace* $\geq$ *with* $>$ *and*
line 3 *replace* $-$ *with* $+$ *and*
line 11 *replace* $-$ *with* $+$
line 2: *replace* $\geq$ *with* $>$ *and*
line 3 *replace* $-$ *with* $+$ *and*
line 11 *replace* $\geq$ *with* $>$

Theorem:
Our algorithm is sound and complete.

That is, for a given bound **b**:

A program is returned by our algorithm

iff

it is minimal and **b-**bounded correct

- Minimal number of changes
- Every assertion reachable along a computation of bounded length **b** is correct

Ver.	Method of [11]		Method of [12]		Our method: Mutation level 1		Our method: Mutation level 2	
	Fixed?	Time[s]	Fixed?	Time[s]	Fixed?	Time[s]	Fixed?	Time[s]
3					+	1.725	+	68.651
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
	16 (39%)	38	15 (36.6%)	38	11 (26.83%)	2.278	18 (43.9%)	48.151

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

implemented in the tool FoREnSiC.

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 bugs are simple, but many

Questions?