

Symbolic Execution

(Chapter 6)

Symbolic execution

- Middle way between testing and analysis
- Executes the program on symbolic values
- One symbolic execution corresponds to many actual executions

Motivation

Test cases for this program ?

```
read(x);  
a := x - 1;  
b := 1 - x;  
if (a + b == 0) then  
    write("error");  
else  
    write("OK");
```

Motivation

Test cases for this program ?

read(x);	$\langle \{x = X\} \rangle$
a := x - 1;	$\langle \{x = X, a = X - 1\} \rangle$
b := 1 - x;	$\langle \{x = X, a = X - 1, b = 1 - X\} \rangle$
if (a + b == 0) then	
write("error");	$\langle \{x = X, a = X - 1, b = 1 - X\}, \text{True} \rangle$
else	
write("OK");	$\langle \{x = X, a = X - 1, b = 1 - X\}, \text{False} \rangle$

Symbolic states

- When control reaches the conditional, symbolic values do not allow execution to select a branch
- One can choose a branch, and record the choice in a *path condition*

- Result:

$\langle \{a = A, y = Y + 5, x = 2 * Y + A + 7\}, \langle 1, 3, 4 \rangle, Y + 2 \leq A \rangle$

The diagram shows three components of a symbolic state: a set of variable values, an execution path, and a path condition. Blue brackets are drawn under each component. The first bracket spans the entire set of variable values and is labeled 'variable values'. The second bracket spans the execution path and is labeled 'execution path'. The third bracket spans the path condition and is labeled 'path condition'.

variable values execution path path condition

Symbolic execution rules (1)

symbolic state:

`<symbolic_variable_values, execution_path, path_condition>`

- **read (x)**
 - removes any existing binding for x and adds binding $x = X$, where X is a *newly introduced* symbolic value
- **write (expression)**
 - $\text{output}(n) = \text{computed_symbolic_value}$ (n counter initialized to 1 and automatically incremented after each output statement)

Symbolic execution rules (2)

- **$x := \text{expression}$**
 - construct symbolic value of expression, SV ;
replace previous binding for x with $x = SV$
- After execution of the last statement of a sequence that corresponds to an **edge of control graph**, append the edge to execution path

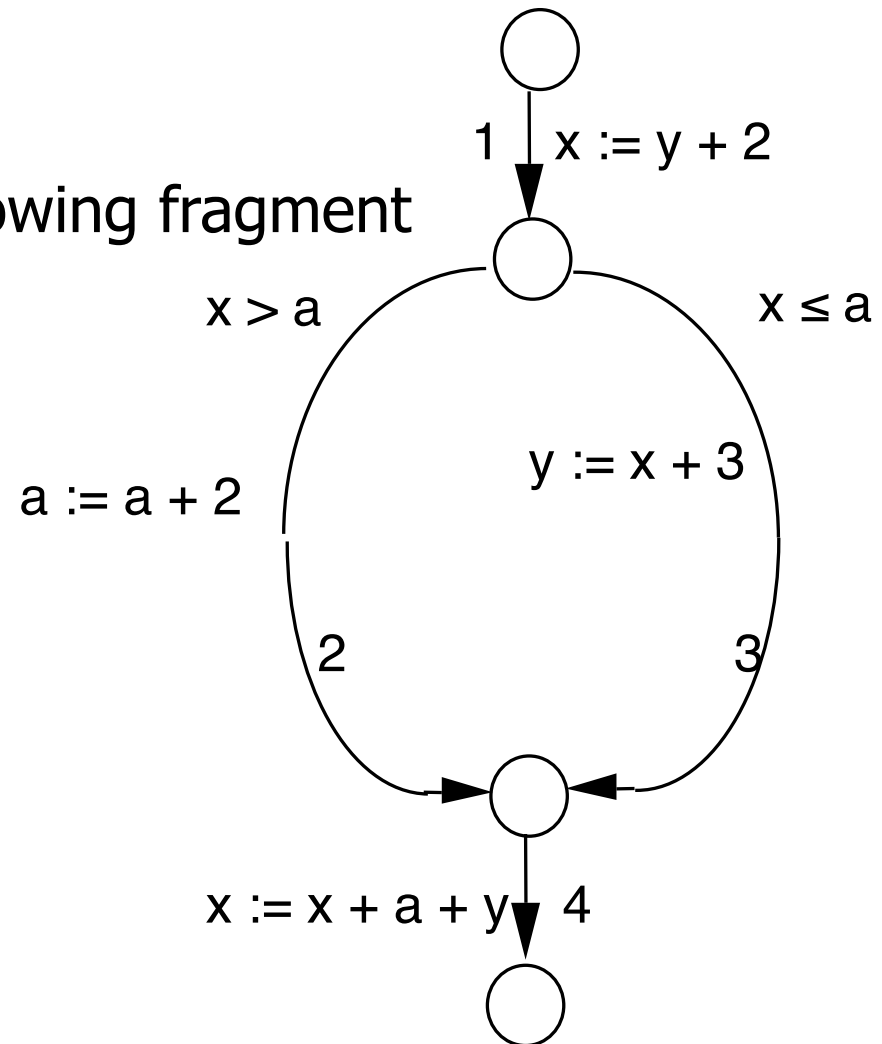
Symbolic execution rules (3)

- **if cond then S1; else S2; endif**
- **while cond loop...endloop**
 - condition is symbolically evaluated
 - if $\text{eval}(\text{cond}) \Rightarrow \text{true}$ or false then execution proceeds by following the appropriate branch
 - otherwise, make nondeterministic choice of true or false, and conjoin $\text{eval}(\text{cond})$ (resp., $\text{not eval}(\text{cond})$) to the path condition

Example 1

Consider executing the following fragment
with $x=X$, $y=Y$, $a=A$

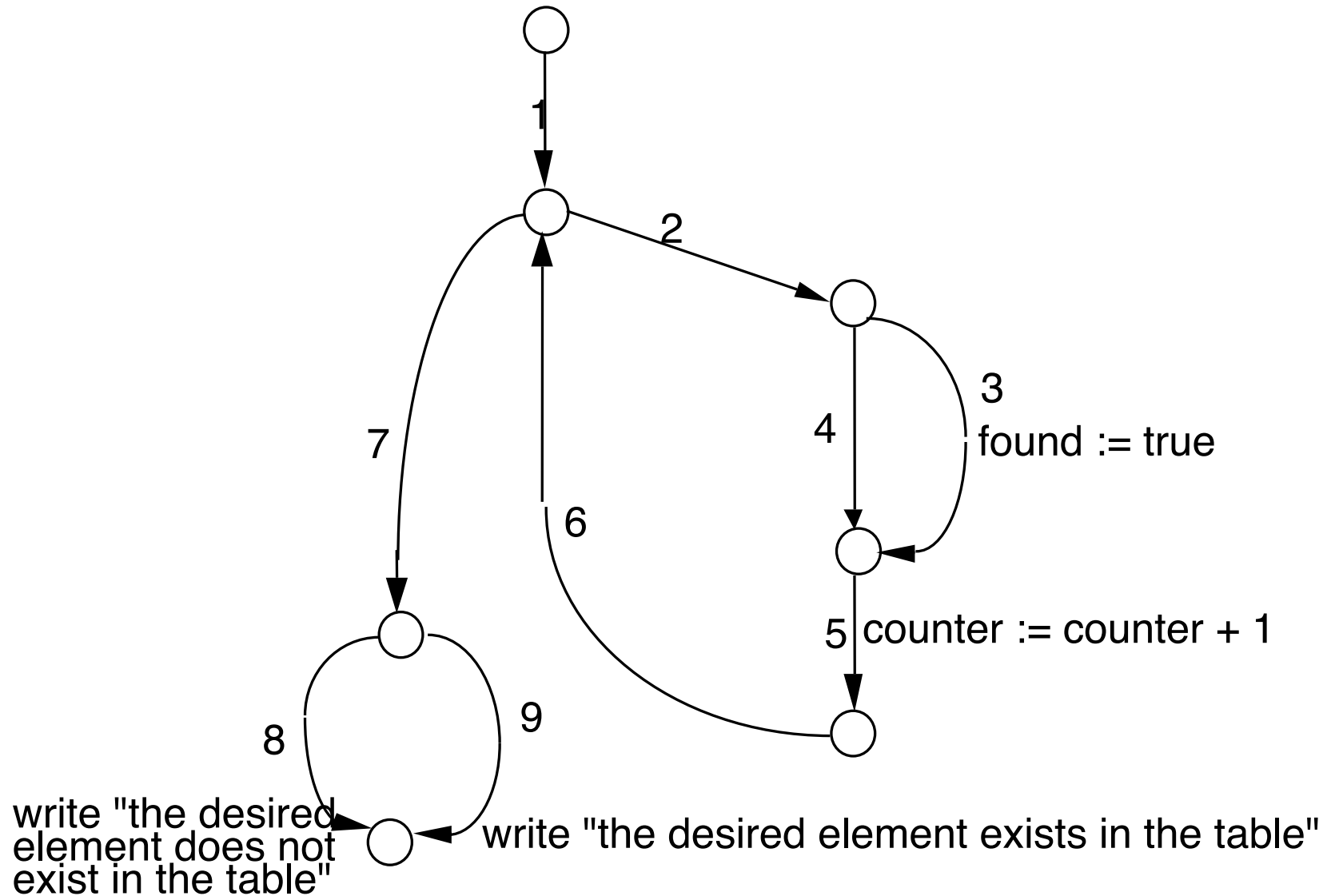
```
x := y + 2;  
if x > a then  
    a := a + 2;  
else  
    y := x + 3;  
end if;  
x := x + a + y;
```



Example 2

```
found := false; counter := 1;
while (not found) and counter ≤ N loop
    if table (counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element exists in the table");
else
    write ("the desired element does not exist
          in the table");
end if;
```

Example 2



```

1 found := false; counter := 1;
  while (not found) and counter ≤ N loop 2
    if table (counter) = desired_element then
      found := true; 3
    // else 4
    end if;
    counter := counter + 1; 5
  end loop; 6
7
  if found then
    9 write ("the desired element exists in the table");
  else
    8 write ("the desired element does not exist
           in the table");

```

what are the conditions for these paths ?

<1,2,3,5,6,7,9>

<1,2,3,5,6,2,3,5,6>

Symbolic execution and testing

- The path condition describes the data that traverse a certain path
- Use in testing:
 - select path (e.g., for complete coverage)
 - symbolically execute it
 - synthesize data that satisfy the path condition and use as test inputs
 - they will execute that path

Exercises

```
read(x);
if x < 0
then
  if x >= 0
  then
    write("oops");
  end if;
end if;
```

```
read (x);read(z);
if x ≠ 0 then
  y := 5;
else
  z := z - x;
end if;
if z > 1 then
  z := z / x;
else
  z := 0;
end if;
```

```
read (x); read (y);
if x > 0 then
  z := x + 1;
else
  z := x + 2;
end if;
if y > 0 then
  z := z + 3;
else
  z := z + 4;
end if;
```

symbolic state:

$\langle \text{symbolic_variable_values, execution_path, path_condition} \rangle$

Why so many approaches to testing and analysis?

- Testing vs. (correctness) analysis
- Formal vs. informal techniques
- White-box vs. black-box techniques
- Techniques in the small/large

view all these as complementary!

Reading for next class

Section 9.3