第九章 基于静态和符号分析的服务缺陷检测

王旭

http://xuwang.tech

The Institute of Advanced Computing Technology

程序静态分析

- 静态分析：静态的代码分析
  - 不用真正的编译或者执行代码
  - 源代码、中间代码（比如java字节码）、二进制码
  - 有助于理解程序本身的结构和某些属性
- 功能
  - 发现潜在的程序缺陷：缓冲区溢出、资源未释放等
  - 控制流分析
  - 数据流分析

The Institute of Advanced Computing Technology

代码审查

- 工业界
  - 人工审查(Code Review)：一般是专家
  - 费时费力，效果好
- 自动化的静态分析
  - 在开发过程中进行
  - 不断发现可能的缺陷
  - 存在问题
    - 不可能发现所有的缺陷(false negative)
    - 不能保障发现的一定是缺陷(false positive)，还需要进一步的人工核实

The Institute of Advanced Computing Technology

对编译器的补充

- 编译器中的静态分析
  - 词法分析
  - 语法分析
  - 编译器优化
  - 某些语义分析：warnings，标识符未使用、未初始化等
- 程序静态分析
  - 针对特定的语义属性进行分析，一般用于寻找程序中的缺陷

The Institute of Advanced Computing Technology

静态分析

- 类型分析(Type Analysis)
  - 某个变量的类型，比如变量a的类名是什么？继承、接口、多态？
- 控制流分析(Control-flow Analysis)
  - 控制流语句和分支
- 数据流分析(Dataflow Analysis)
  - 变量值的变化过程
- 指针分析(Point Analysis)
  - 判断两个变量是否指向相同的内存地址，即是否是同一对象

The Institute of Advanced Computing Technology

类型分析

- 类型定义
  - Object A=Org.buaa.edu.MyApp()
- 父类继承
  - MyApp extends BasicApp
- 接口实现
  - MyApp implements AppOperator
- 多态: 静态分析时无法确定是哪一个类
  - MyApp.show()、 AppOperator.show()、 BasicApp.show()

The Institute of Advanced Computing Technology

Java Example: BoolExp hierarchy

```
public class AndExp extends BoolExp {
   private BoolExp _operand1;
   private BoolExp _operand2;

   public AndExp(BoolExp op1, BoolExp op2) { _operand=op1;   _operand2=op2; }
   public boolean Evaluate(Context c) {
       return _operand1.Evaluate(c) && _operand2.Evaluate(c);
    }
}
```

_operand1: {Constant} _operand2: {OrExp}

```
public class OrExp extends BoolExp {
   private BoolExp _operand1;
   private BoolExp _operand2;

   public OrExp(BoolExp op1, BoolExp op2) { _operand=op1;   _operand2=op2; }
   public boolean Evaluate(Context c) {
       return _operand1.Evaluate(c) ||  _operand2.Evaluate(c);
   }
}
```

_operand1: {VarExp} _operand2: {VarExp}

控制流分析(1)

- 控制流图(CFG, Control-flow Graph)
  - 节点：基本块(Basic Block)，代码语句
  - 边：控制流顺序

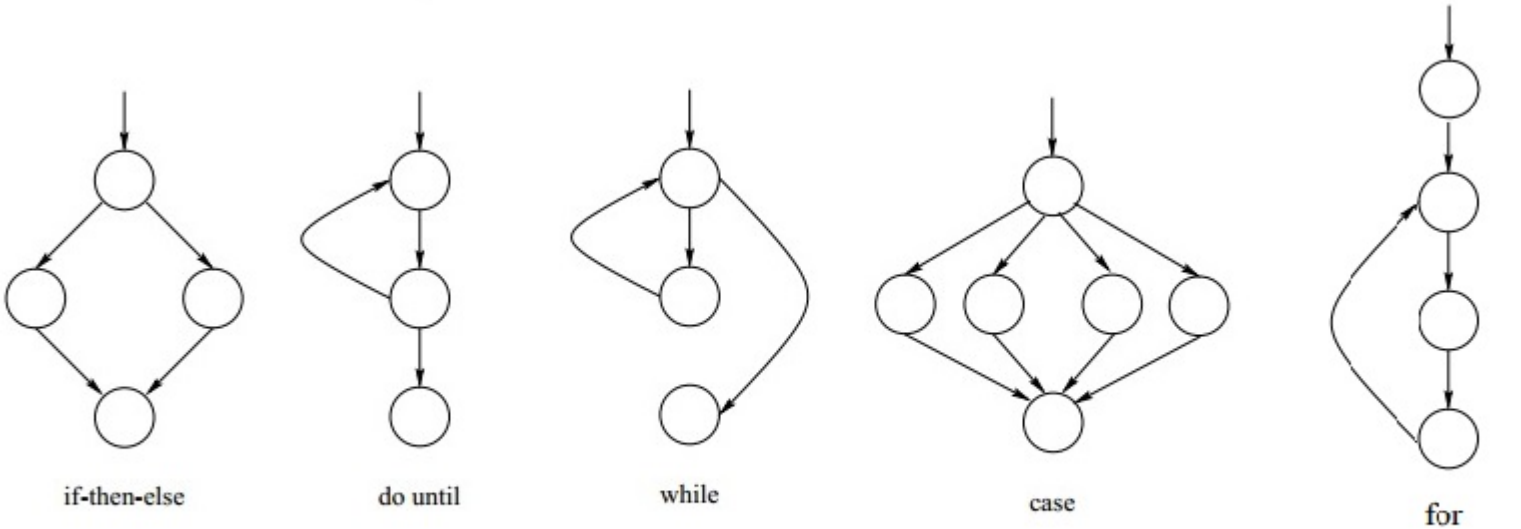


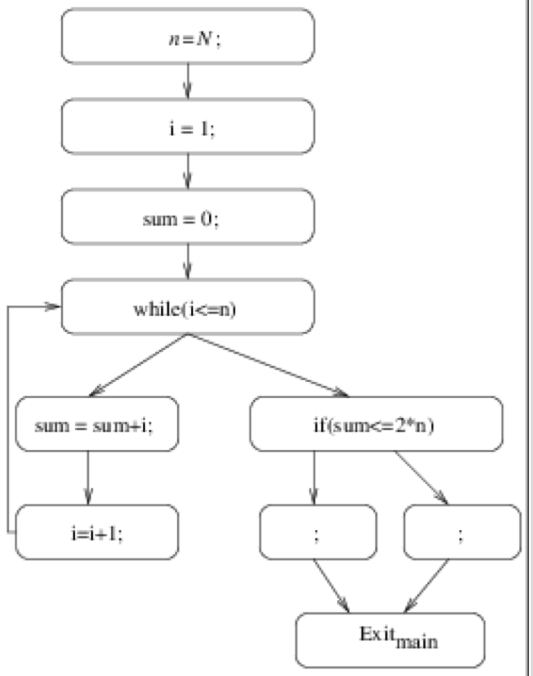
Figure 1: Flow graph representation.

The Institute of Advanced Computing Technology

控制流分析(2)

- 控制流图(CFG, Control-flow Graph)
  - 节点：基本块(Basic Block)，代码语句
  - 边：控制流顺序

```
int n, sum;
main(){
  int i;
  n = N;
  i = 1;
  sum = 0;
  while ( i<=n ){
    sum = sum + i;
    i = i + 1;
  }
  if ( sum < 2*n ){
   ERROR: ;
  } else {
   ;
  }
}
```



The Institute of Advanced Computing Technology

控制流分析(3)

- **控制流图(CFG, Control-flow Graph)**
  - **可嵌套，跨函数分析(Interprocedural analysis)**
  - **每个函数调用形成新的子图： Call Graph**
- **多态调用**
  - Object.showMyApp.show()、 AppOperator.show()、BasicApp.show()
  - Class Hierarchy Analysis (CHA): 包括可能的所有父类和接口
  - Rapid Type Analysis (RTA)：只考虑已经初始化的类

The Institute of Advanced Computing Technology

数据流分析(1)

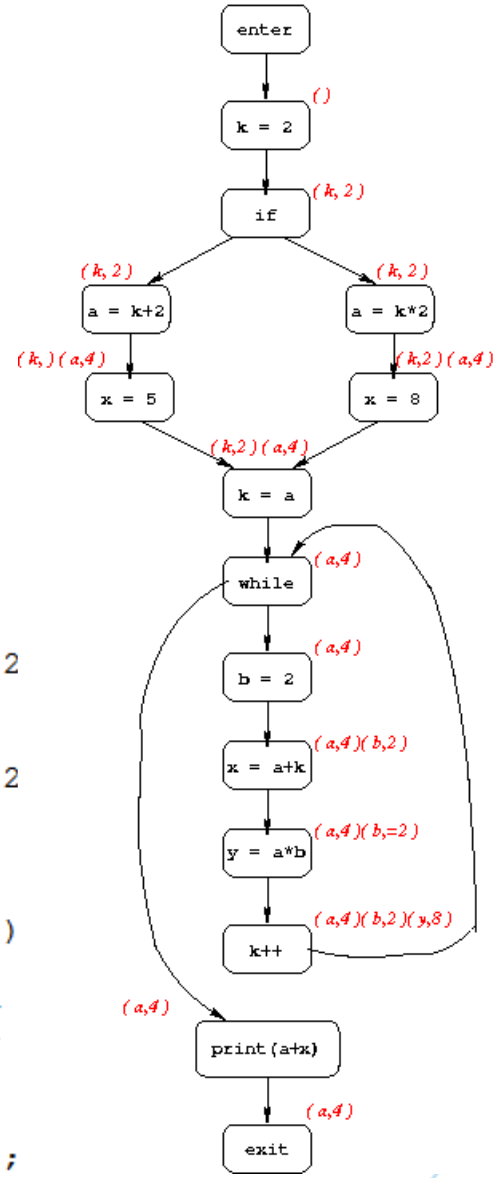
- 基于控制流图(CFG)
- 状态转换函数
  - 程序语句、程序块
- 状态机转换

For each block b:

$$out_b = trans_b(in_b)$$
$$in_b = join_{p \in pred_b}(out_p)$$

- 包括所有分支
  - 含不可能分支

```
1.   k = 2;
2.   if (...) {
3.     a = k + 2
4.     x = 5;
5.   } else {
6.     a = k * 2
7.     x = 8;
8.   }
9.   k = a;
10.  while (...)
11.    b = 2;
12.    x = a +
13.    y = a *
14.    k++;
15.  }
16.  print(a+x);
```



The Institute of Advanced Computing Technology

数据流分析(2)

- 常量传播分析(constant propagation)
  - 分析哪些位置的变量值是常量
- 活性分析(liveness analysis)
  - 分析哪个位置的变量值在被更新前(Overwrite)前可能会被后续的语句使用
- 可用表达式分析(available expression)
  - 分析哪些位置的表达式不用重新计算
- 定义可达性分析(reaching definition analysis)
  - Definition: assignment v=4;
  - Reach: 赋值语句可影响某个位置的语句

The Institute of Advanced Computing Technology

指针分析(1)

- 对象别名，也称别名分析(Alias analysis)

```
class X {
   Z f; Z g;
   void bar(Z z) {
      this.f = z;
   }
}
```

```
class Y extends X {
    void bar (Z z) {
        this.g = z;
    }
}
```

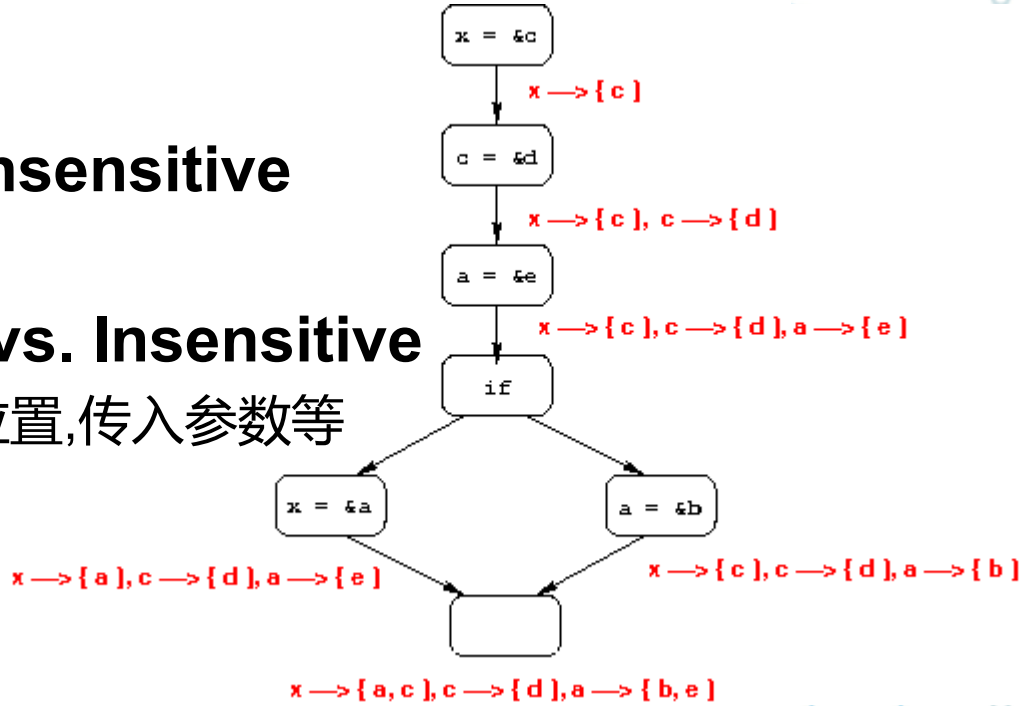
```
class A {
   X x; X y;
   void a1() {
      y = new Y();
      x = y;
      Z z = new Z();
      foo(z);
   }
   void a2() {
      x = new X();
      y = x;
      Z z = new Z();
      foo(z);
   }
   void foo(Z a) {
      x.bar(a);
      y.bar(a);
   }
}
```

- x.f 和y.g是同一个对象吗？

The Institute of Advanced Computing Technology

指针分析(2)

- 类似于数据流分析
  - 记录每个对象的指针地址
- 不同维度
  - Flow Sensitive vs. Insensitive
    - 是否考虑控制流
  - Context Sensitivity vs. Insensitive
    - 是否考虑函数调用的位置,传入参数等



其余分析方法

- 跨函数分析
  - 考虑函数调用
    - A.a() 实际调用的是哪一个方法？
  - 参数传递
    - Obj b = New B(1); A.a(b) 局部变量
  - Heap变量
    - ClassA{ obj c; void a(obj b){c=b;}
    - Obj b = New B(1); A.a(b)； A.c = d;
- Slicing分析
  - 分析可能会影响某个位置变量值的代码语句集合
  - 静态：所有path的可能语句; 动态：特定path

The Institute of Advanced Computing Technology

常用开源工具

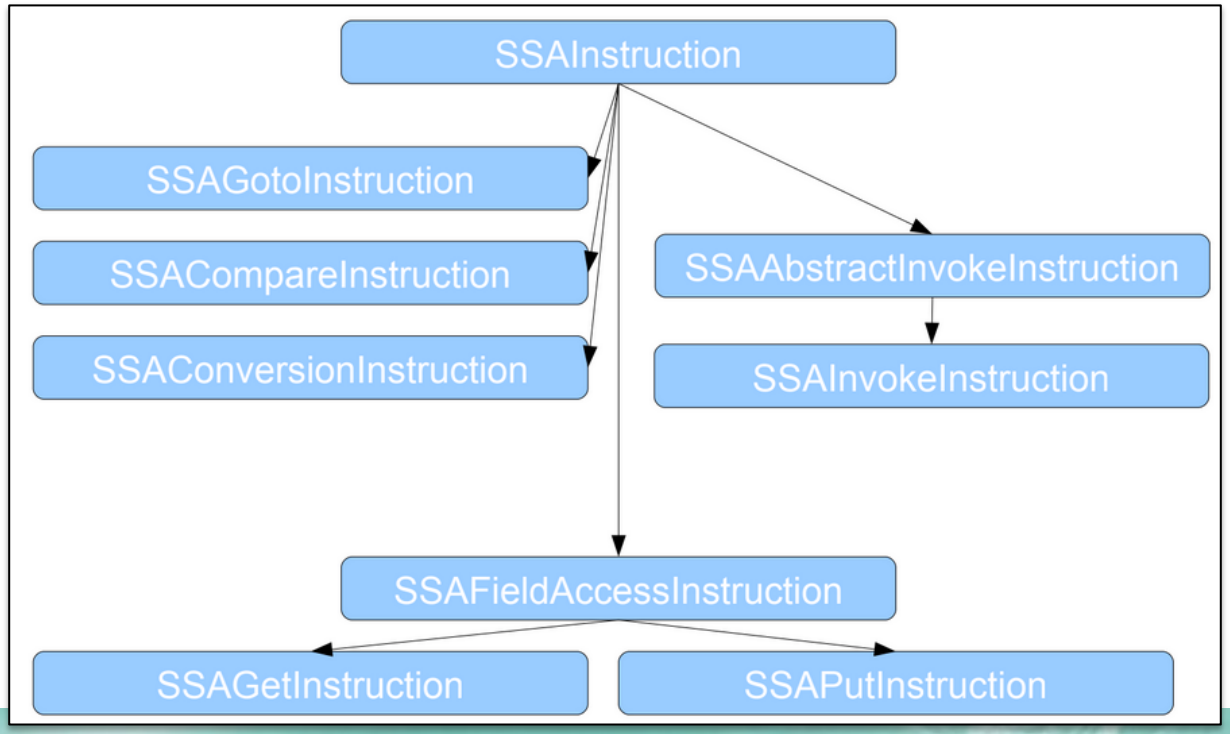
- Java语言
  - WALA

- C/C++语言
  - LLVM

The Institute of Advanced Computing Technology

静态分析例子-WALA(1)

- Intermediate representation (IR)
  - Static single assignment form (SSA)
  - 每个变量赋值一次，使用前一定被定义



The Institute of Advanced Computing Technology

静态分析例子-WALA(2)

- **控制流图：CallGraph**

```
buildCG(String jarFile) {
  // represents code to be analyzed
  AnalyisScope scope = AnalysisScopeReader                    (1)
      .makeJavaBinaryAnalysisScope(jarFile, null);
  // a class hierarchy for name resolution, etc.
  IClassHierarchy cha = ClassHierarchy.make(scope);           (2)
  // what are the call graph entrypoints?
  Iterable<Entrypoint> e =                                    (3)
      Util.makeMainEntrypoints(scope, cha);
  // encapsulates various analysis options
  AnalysisOptions o = new AnalysisOptions(scope, e);          (4)
  // builds call graph via pointer analysis
  CallGraphBuilder builder =                                  (5)
    Util.makeZeroCFABuilder(o, new AnalysisCache(),
                            cha, scope);
  CallGraph cg = builder.makeCallGraph(o, null);              (6)
}
```

The Institute of Advanced Computing Technology

静态分析例子-WALA(3)

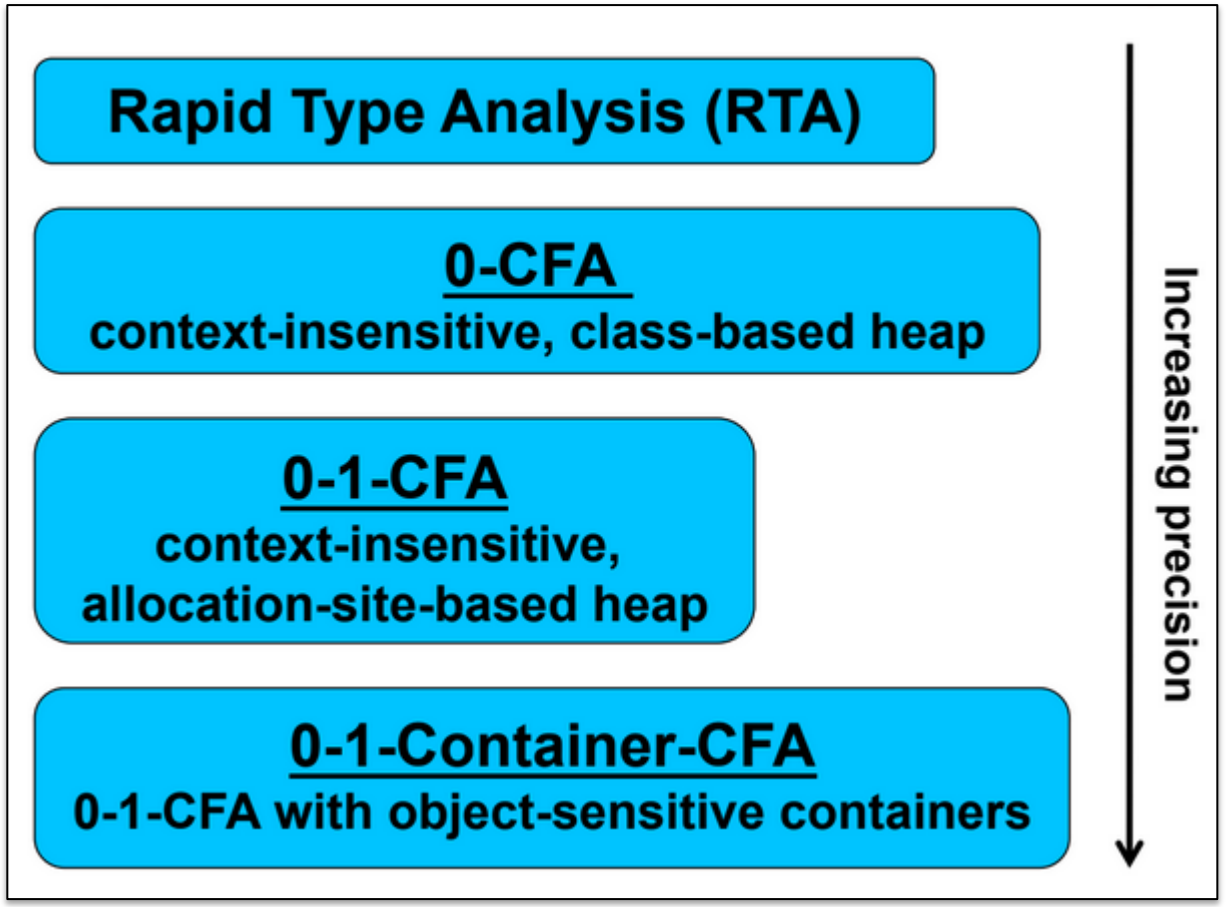
- 获取类、方法和属性

Entity	Reference Type	Resolved Type	Resolver Method
class	TypeReference	IClass	`lookupClass()`
method	MethodReference	IMethod	`resolveMethod()`
Field	FieldReference	IField	`resolveField()`

The Institute of Advanced Computing Technology

静态分析例子-WALA(4)

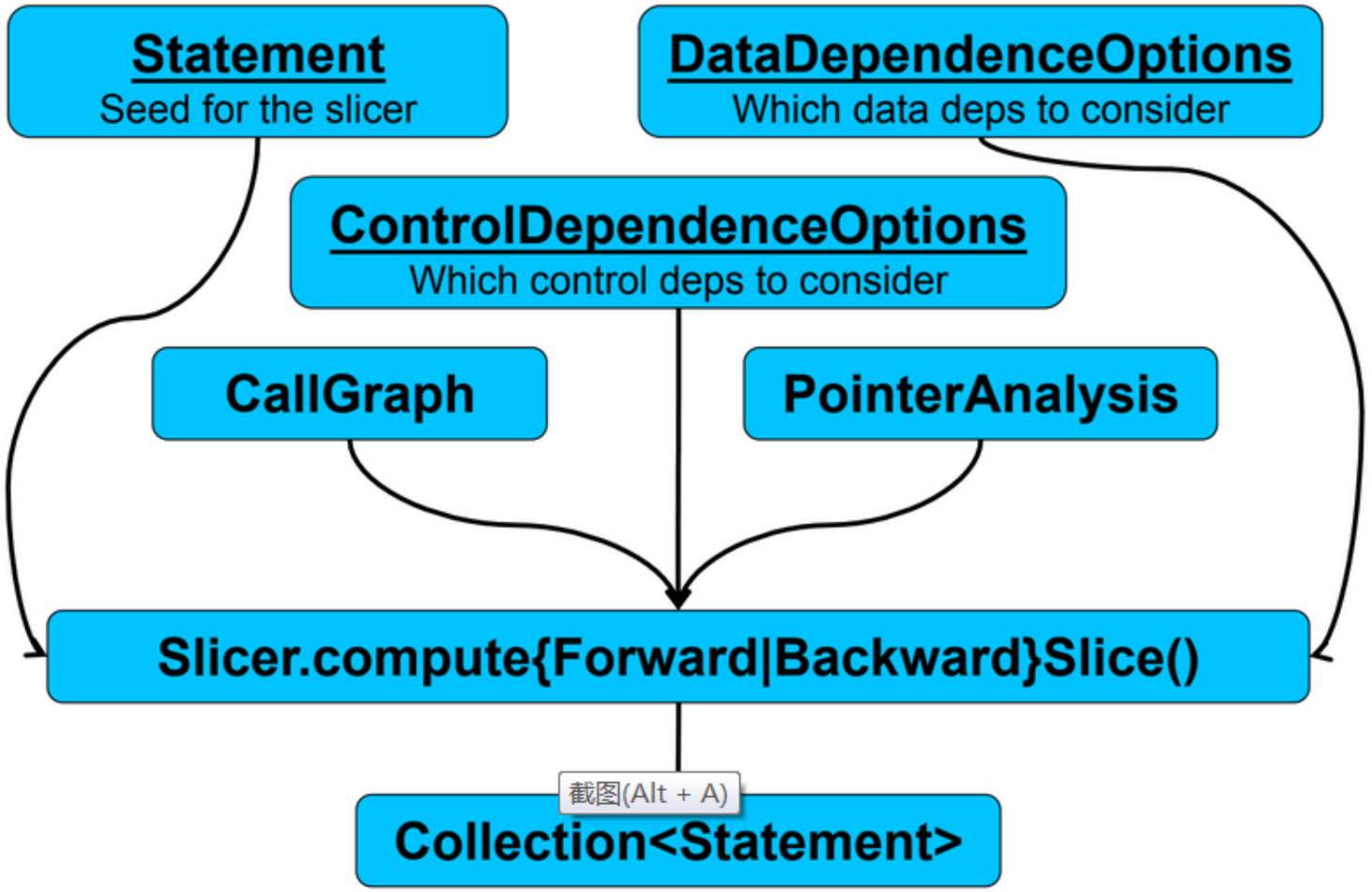
- 指针分析



The Institute of Advanced Computing Technology

静态分析例子-WALA(5)

- **Slicing分析**



The Institute of Advanced Computing Technology

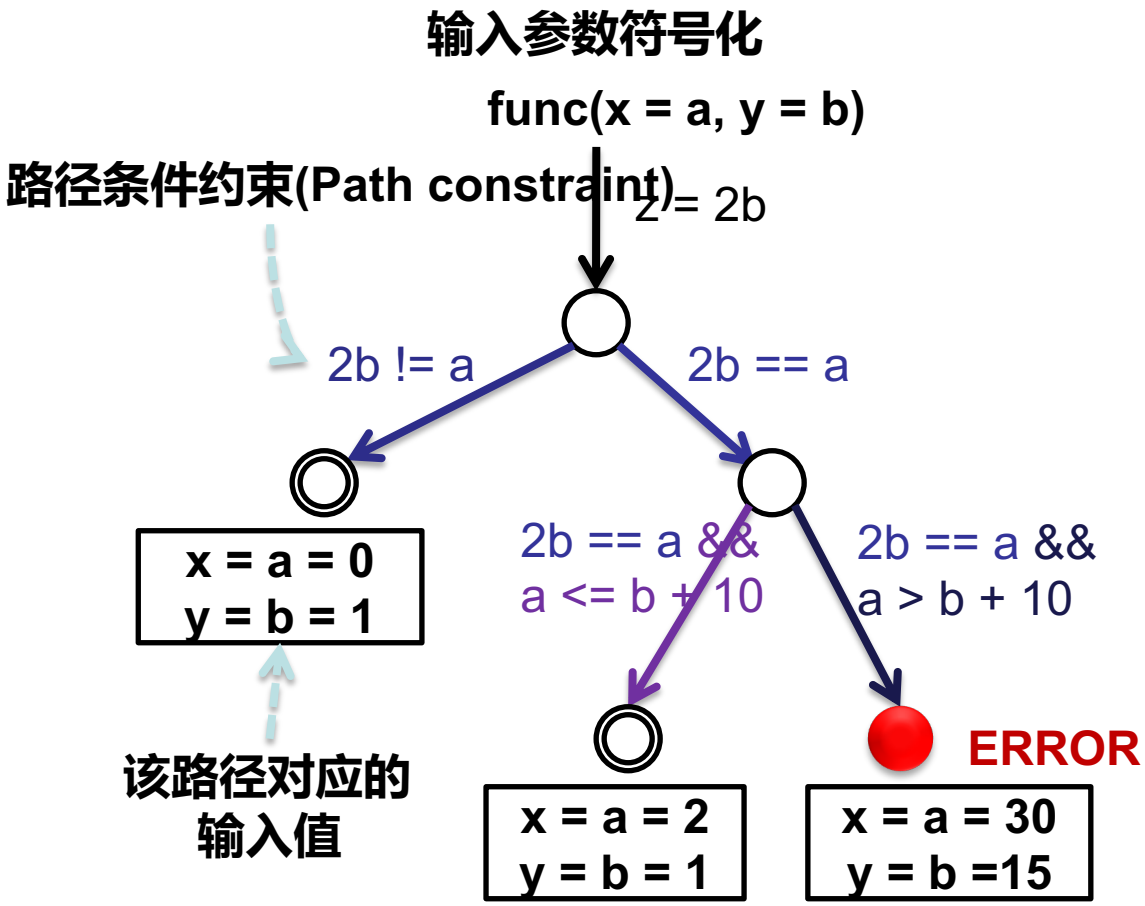
符号执行分析

- 将程序的输入（含环境输入）设为符号
  - Fun(int x, int y) x→a y→b
  - Scanf(“%d”,&z) z→c
- 分析符号取值对程序执行路径的影响
  - 分支条件约束取值
    - If(a<0&&b>10)
  - While, for, if, switch, if-else
  - 嵌套条件分支
    - if(a<0&&b>10) { while(c<5) ...}
  - 可以判断不同输入条件下程序的执行覆盖度

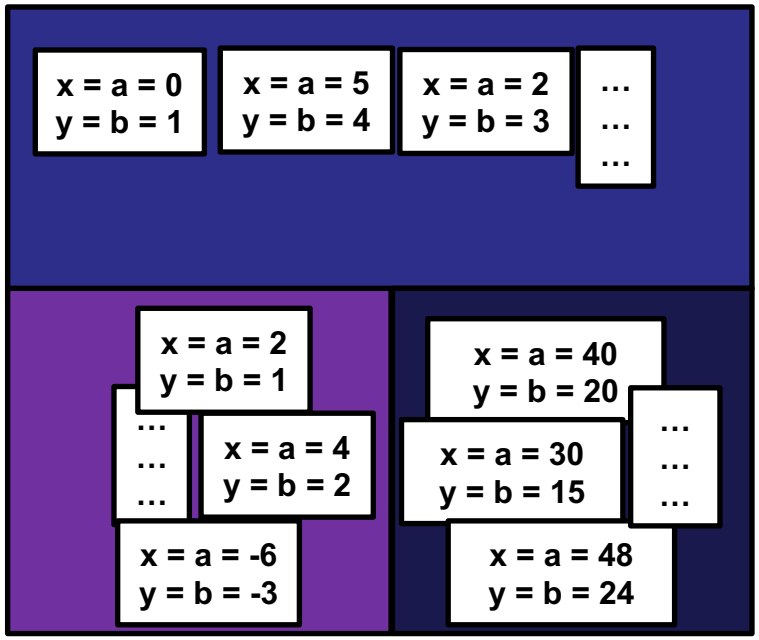
The Institute of Advanced Computing Technology

一个例子

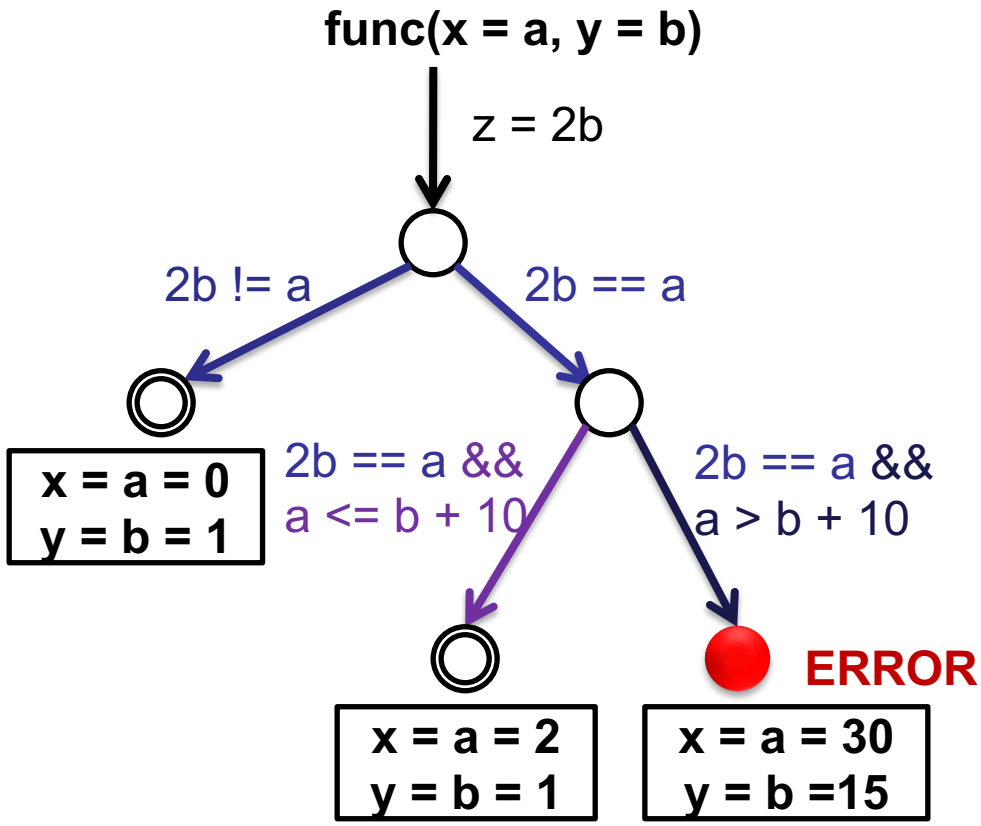
```
Void func(int x, int y){
    int z = 2 * y;
    if(z == x){
        if (x > y + 10)
            ERROR
    }
}
int main(){
    int x = sym_input();
    int y = sym_input();
    func(x, y);
    return 0;
}
```



路径约束对应的输入集合



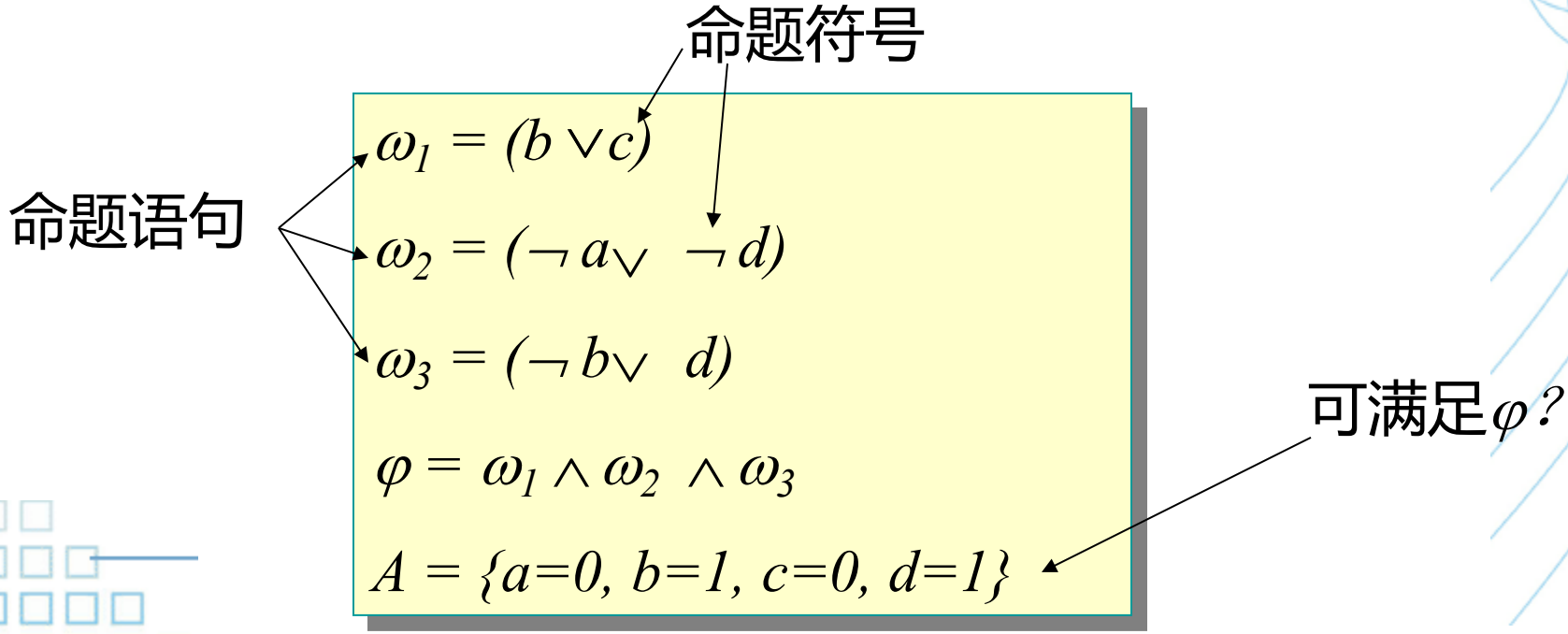
Path constraints represent equivalence classes of inputs



通过符号化，可以将输入参数分类，每一类只需要选取一个值，从而减少可能的输入参数

原理：命题逻辑可满足性

- **命题逻辑的可满足性问题(Boolean satisfiability problem)**
  - 命题逻辑的合取范式是否可满足（为真）？



The Institute of Advanced Computing Technology

原理：可满足性模理论SMT

- 可满足性模理论(Satisfiability Modulo Theory)
  - 在命题逻辑基础上推广至一阶逻辑(First-order logic)
  - 每个命题符号可以用实数、整数、列表、数组、比特向量等构成的公式代替
  - 已有SMT Solver: Z3, CVC4

输入: a first-order formula φ over background theory (Arithmetic, Arrays, Bit-vectors, Algebraic Datatypes)

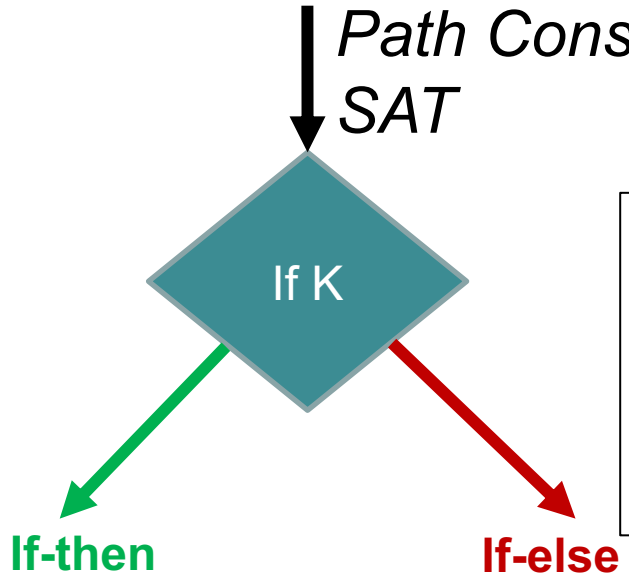
输出: is φ satisfiable?

比如：$x+y-z<5 \wedge x<0$

The Institute of Advanced Computing Technology

基于SMT查询的符号执行

- 静态分析获取所有的前置分支条件C_1, C_2, …, C_n
- 分支条件查询判断某个分支是有效的



Use queries to determine validity of a branch

else path is impossible: $C_1 \wedge C_2 \wedge \ldots \wedge C_n \wedge \neg K$ is UNSAT

then path is impossible: $C_1 \wedge C_2 \wedge \ldots \wedge C_n \wedge K$ is UNSAT

符号执行的不足

- 循环和递归
  - 无限分支树
- 路径爆炸
  - 条件分支指数级增长
- 程序覆盖率
  - 无法探测到深层次的分支，比如碰到循环
- 复杂数据结构的支持
- SMT Solver对复杂条件的求解有局限
- 环境交互产生的输入

The Institute of Advanced Computing Technology

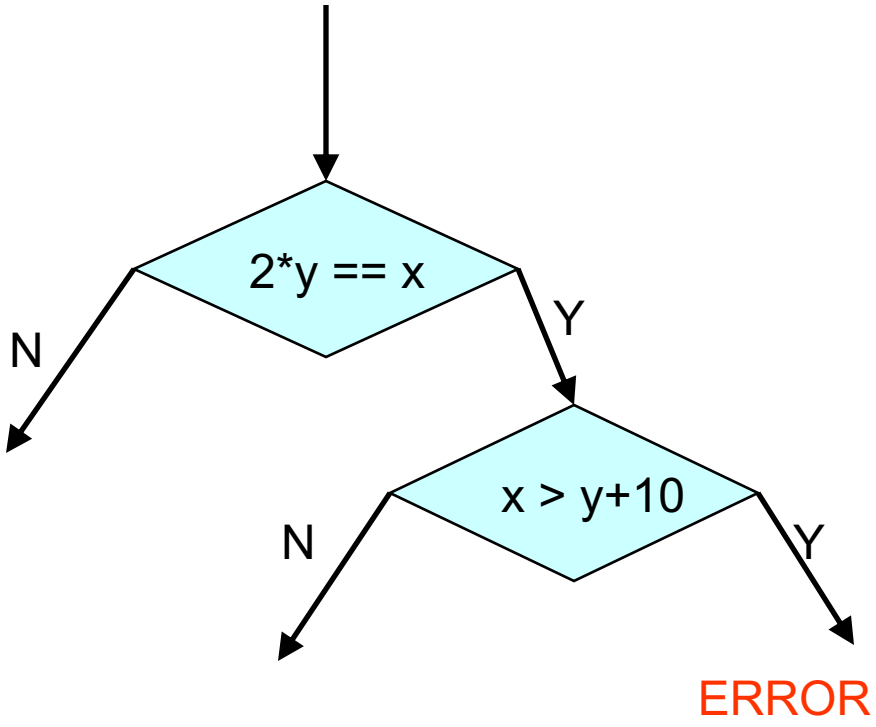
动态符号执行

- 动态符号执行(Dynamic Symbolic Execution)
  - 具体执行+符号执行(Concoloc Execution)
- 提升程序覆盖率
  - 具体执行：随机生成具体的输入参数，探测尽可能深的程序分支
  - 符号执行：基于探测到的程序分支条件，进行SMT求解

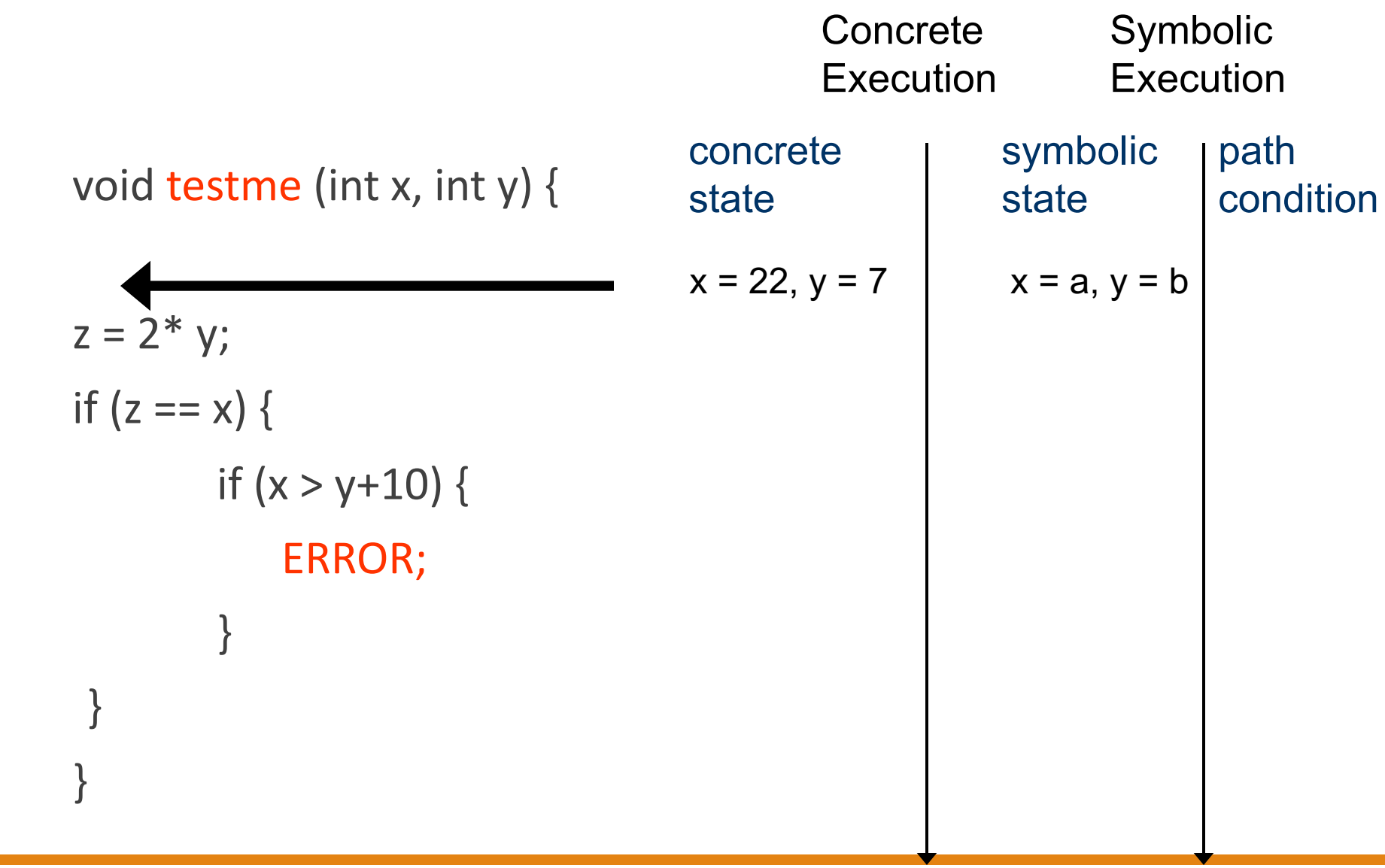
The Institute of Advanced Computing Technology

DSE例子

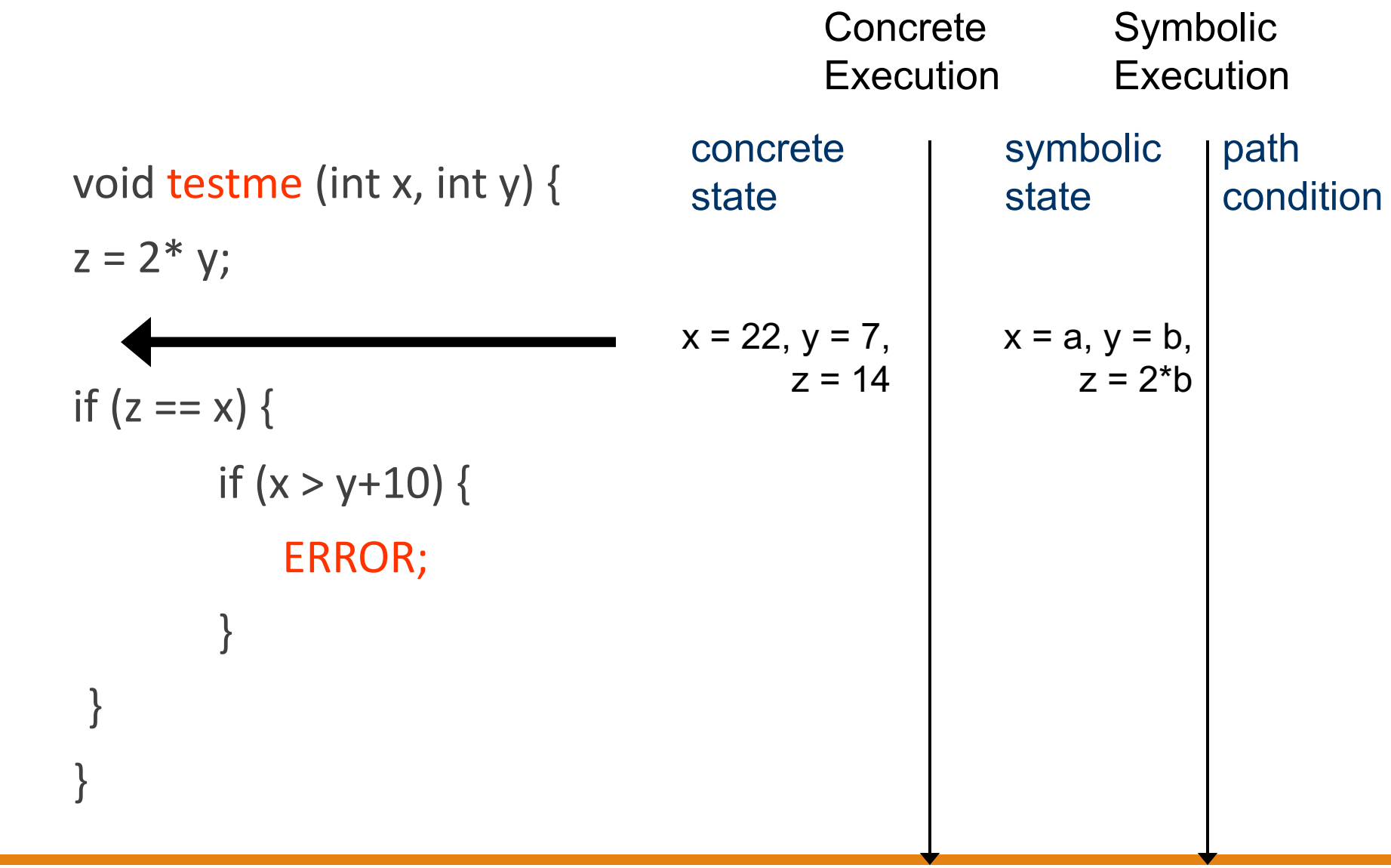
```
void testme (int x, int y)
{
 z = 2*y;
 if (z == x) {
        if (x > y+10) {
            ERROR;
        }
 }
}
```



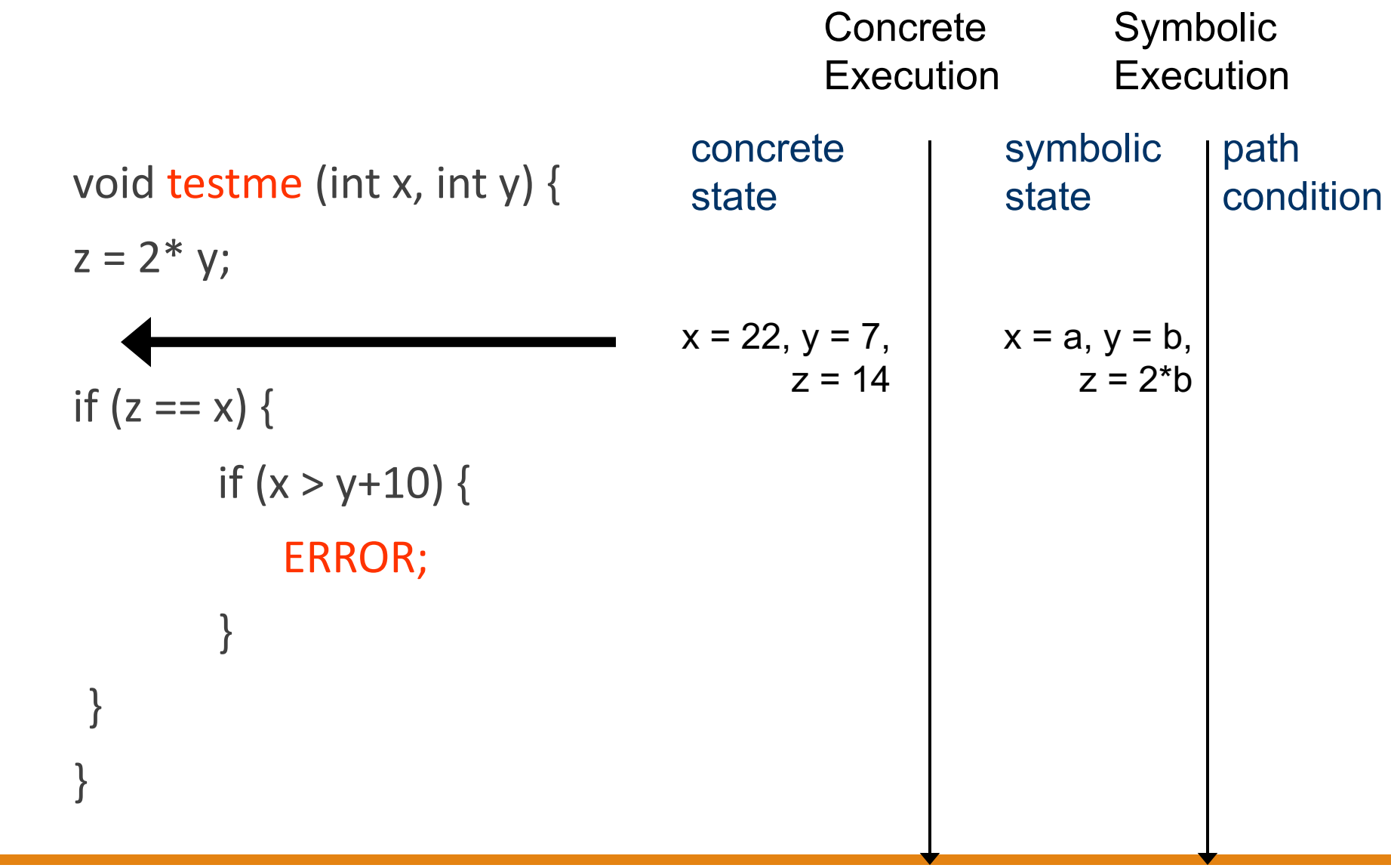
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
x = 22, y = 7
x = a, y = b
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

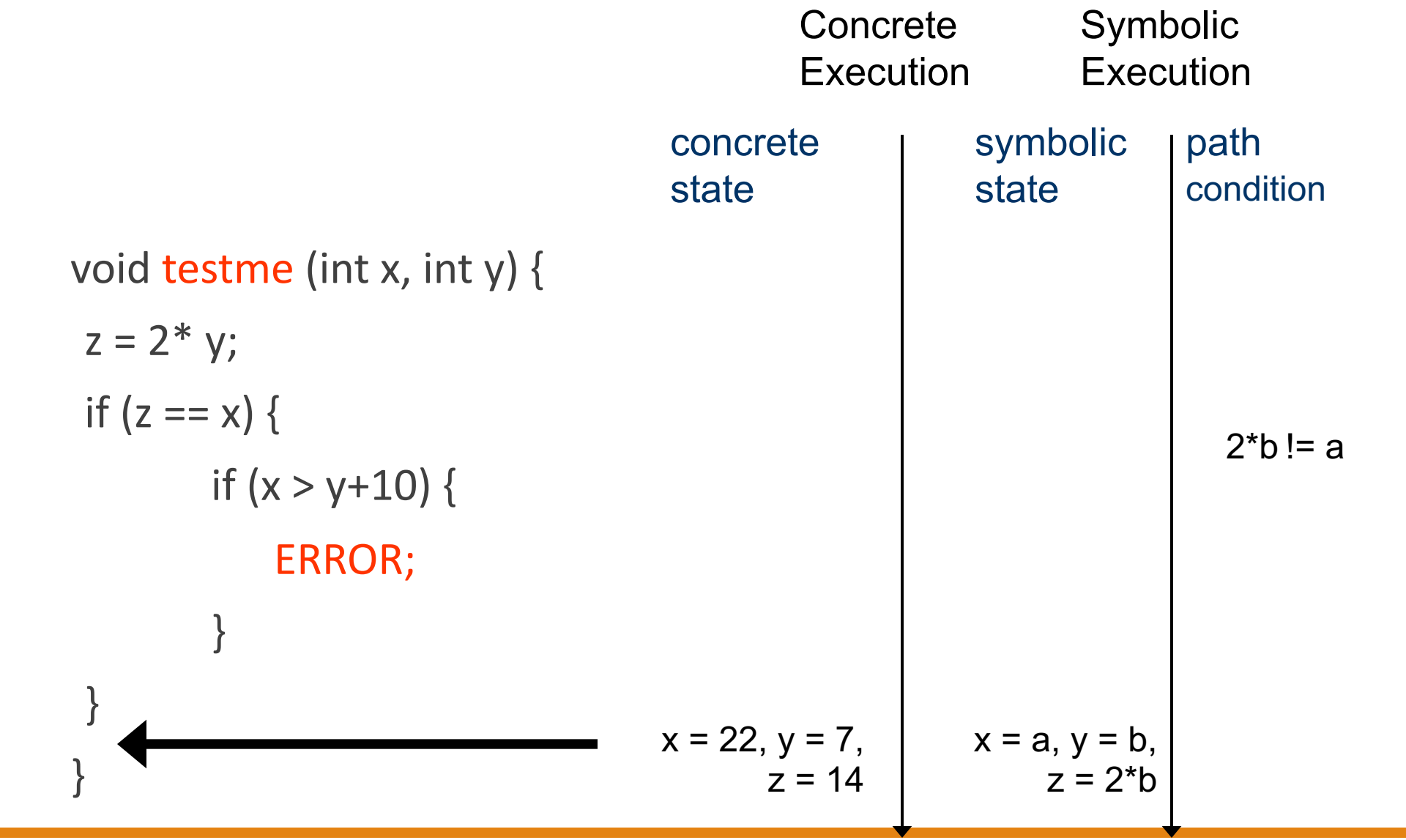
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
x = 22, y = 7, z = 14
x = a, y = b, z = 2*b
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

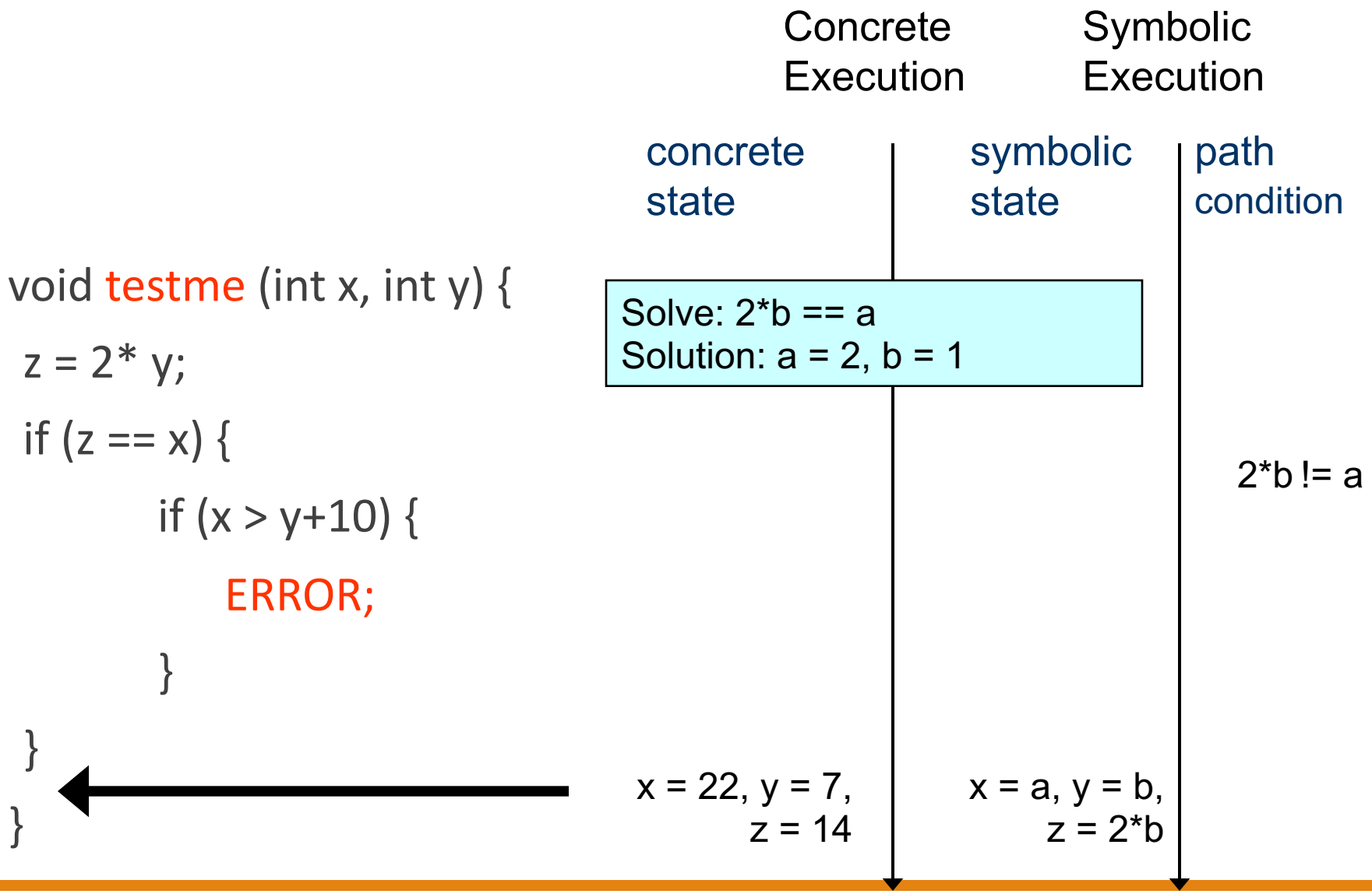
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
x = 22, y = 7, z = 14
x = a, y = b, z = 2*b
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

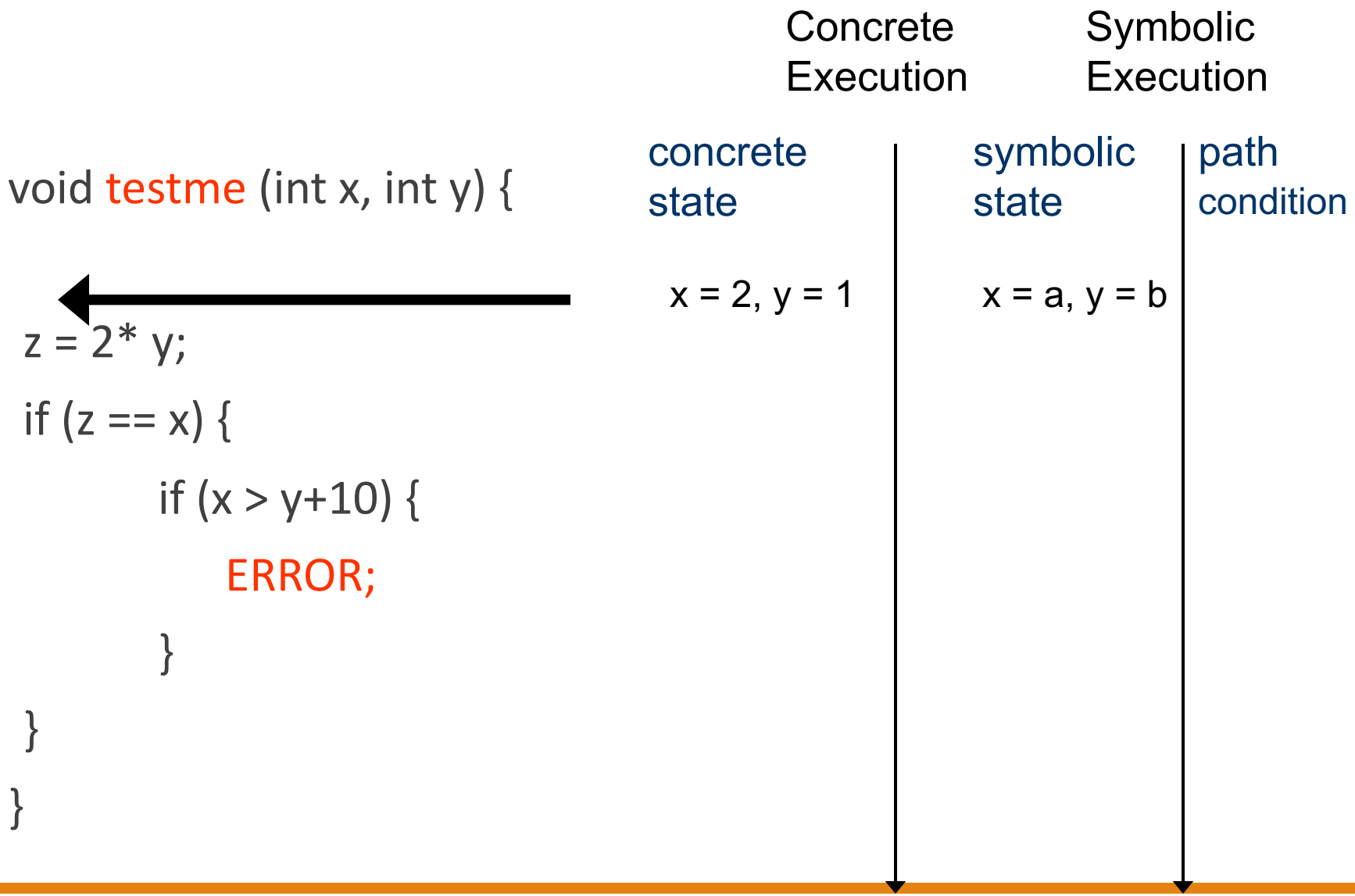
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}
2*b != a
x = 22, y = 7, z = 14
x = a, y = b, z = 2*b

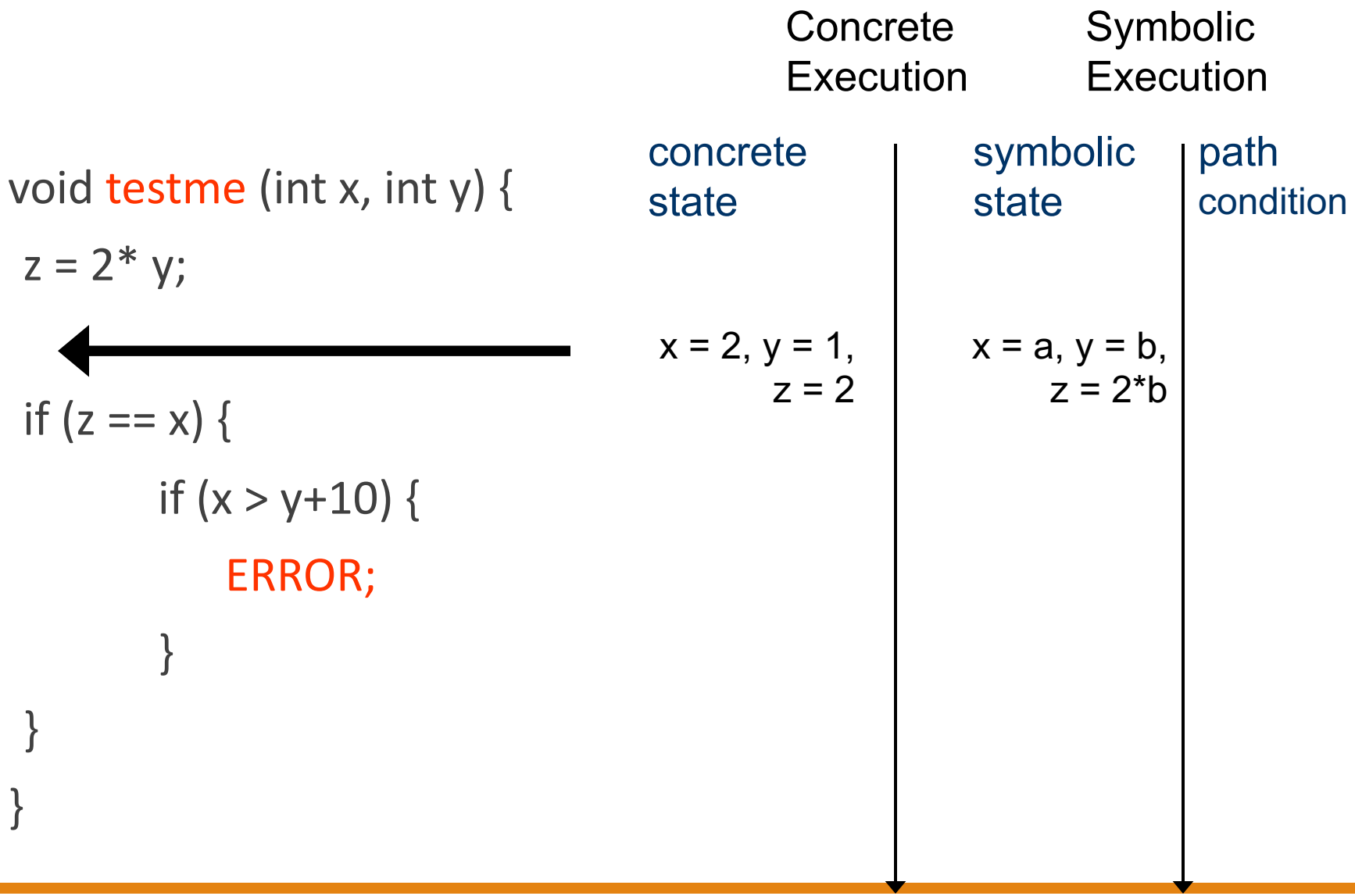
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}
Solve: 2*b == a
Solution: a = 2, b = 1
2*b != a
x = 22, y = 7, z = 14
x = a, y = b, z = 2*b

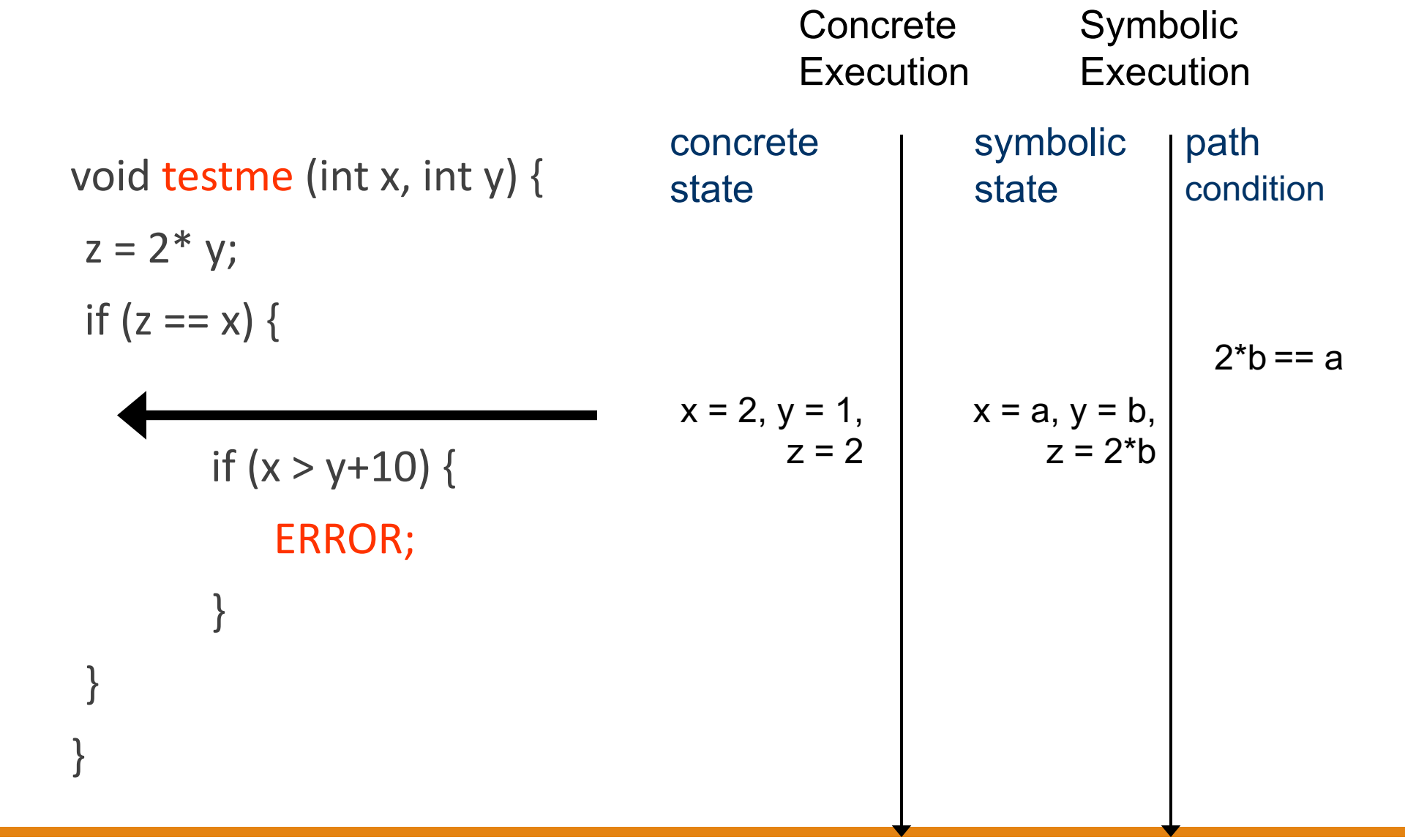
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
x = 2, y = 1
x = a, y = b
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

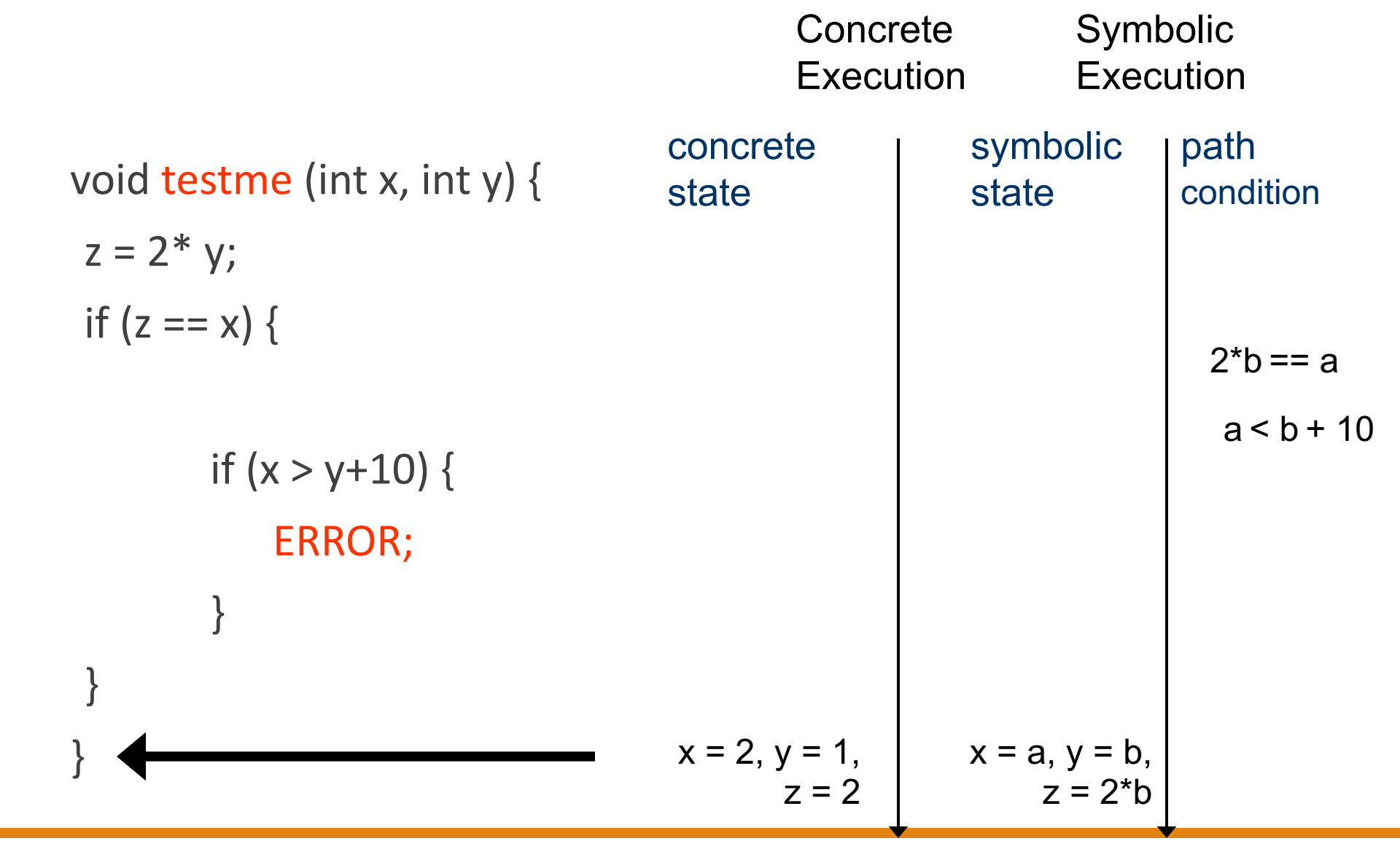
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
x = 2, y = 1, z = 2
x = a, y = b, z = 2*b
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

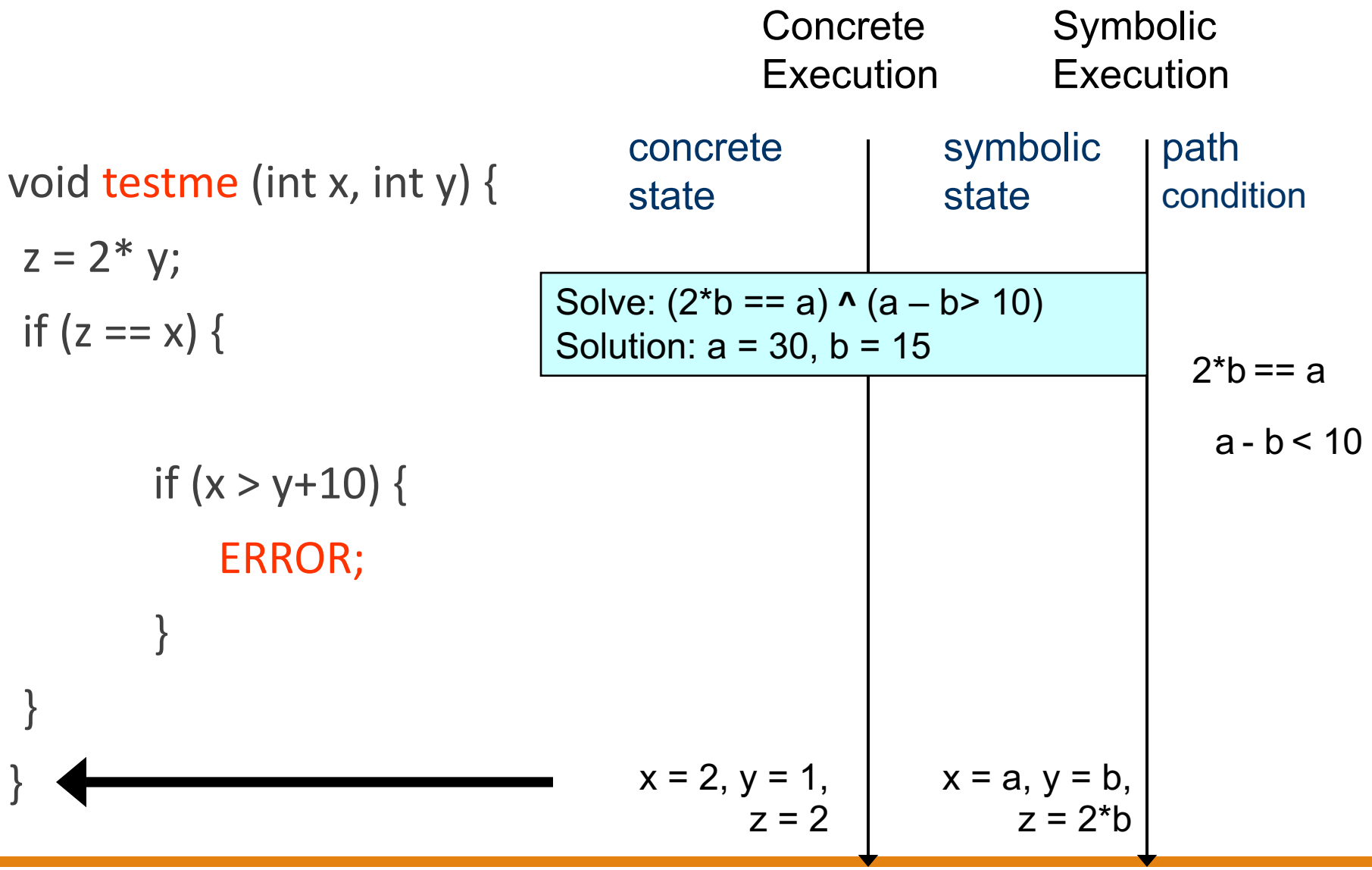
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
x = 2, y = 1, z = 2
x = a, y = b, z = 2*b
2*b == a
if (x > y+10) {
ERROR;
}
}
}

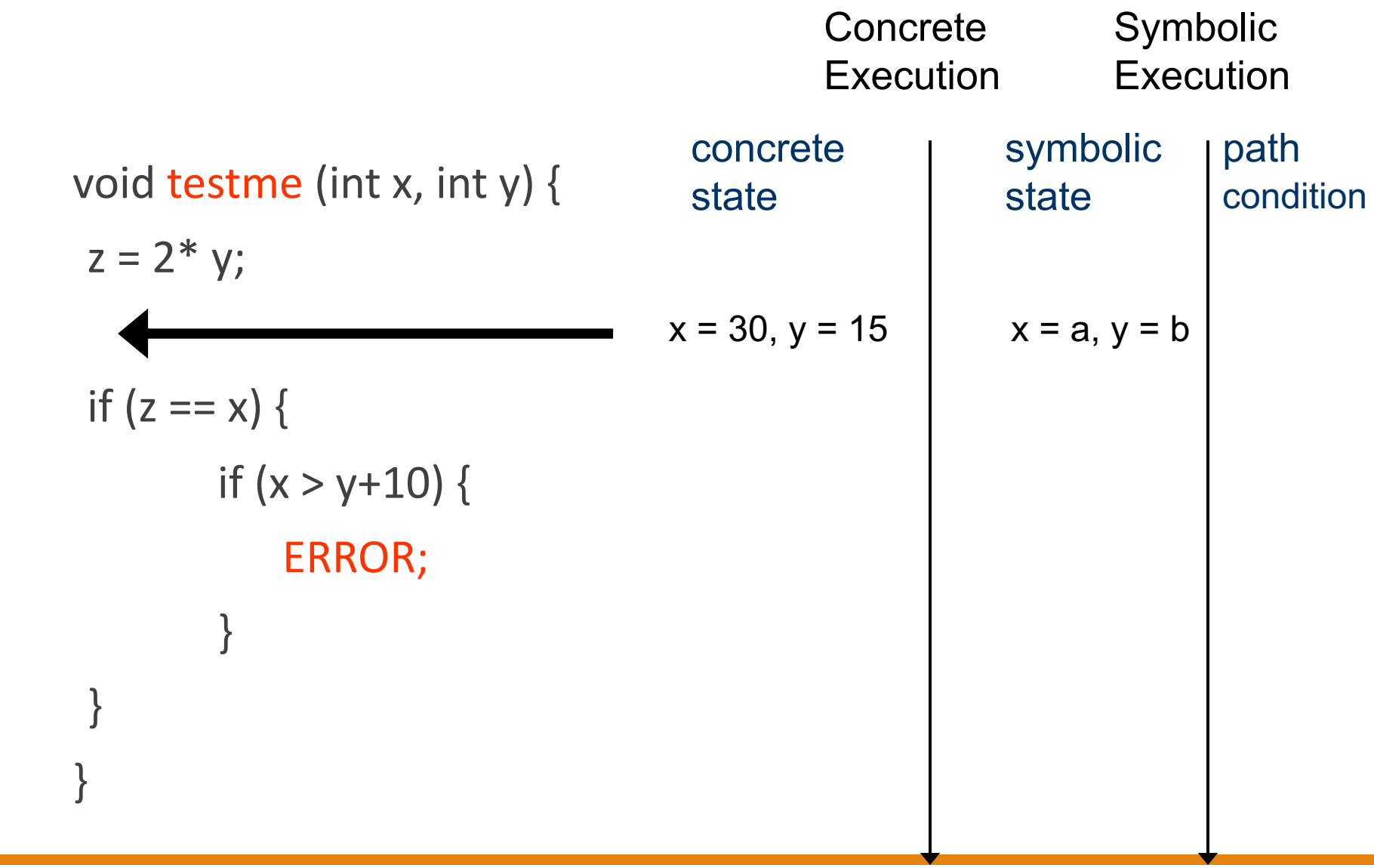
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}
2*b == a
a < b + 10
x = 2, y = 1, z = 2
x = a, y = b, z = 2*b

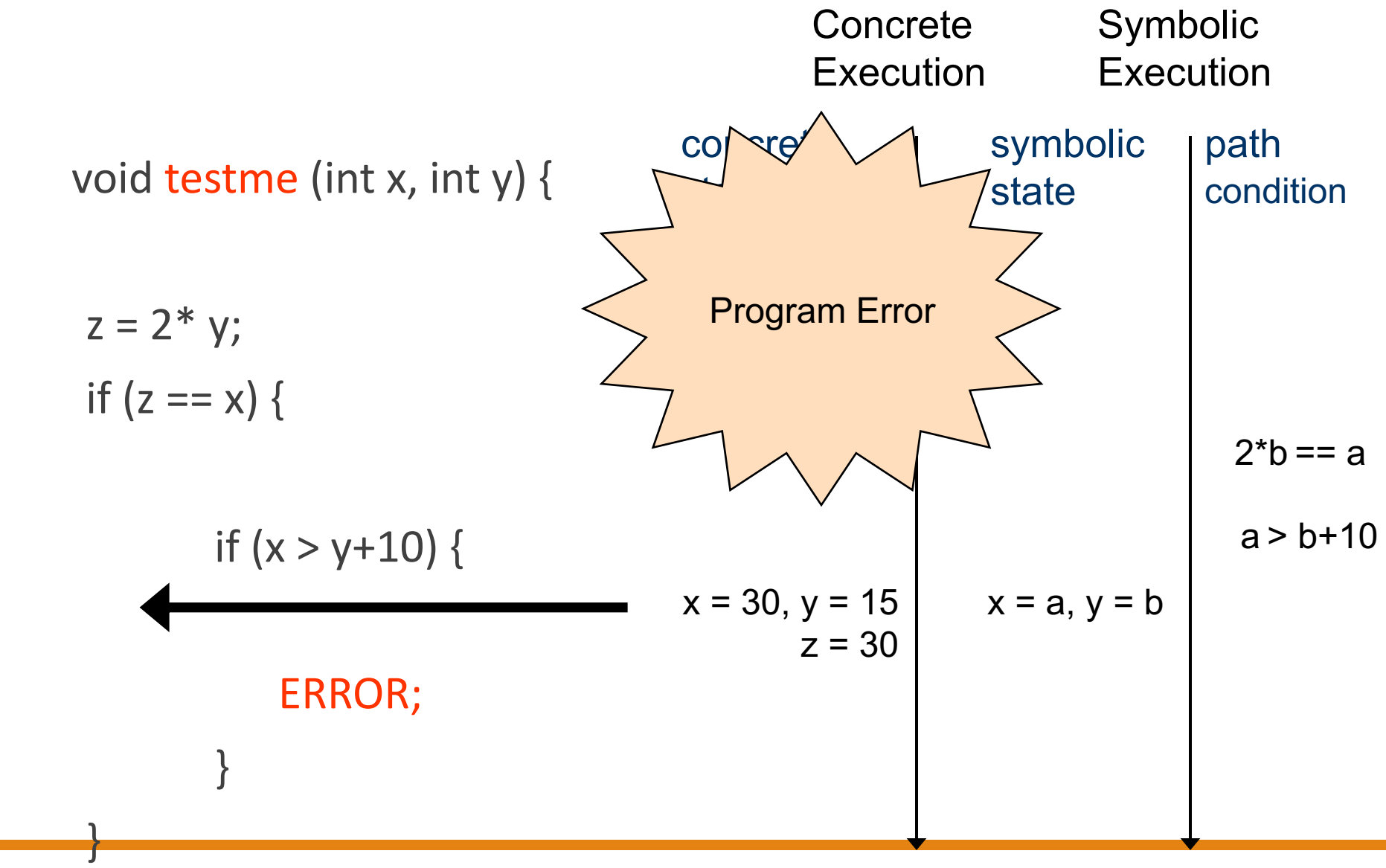
DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}
Solve: (2*b == a) ^ (a – b> 10)
Solution: a = 30, b = 15
2*b == a
a - b < 10
x = 2, y = 1, z = 2
x = a, y = b, z = 2*b

DSE例子


Concrete Execution
Symbolic Execution
concrete state
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
x = 30, y = 15
x = a, y = b
if (z == x) {
if (x > y+10) {
ERROR;
}
}
}

DSE例子


Concrete Execution
Symbolic Execution
symbolic state
path condition
void testme (int x, int y) {
z = 2* y;
if (z == x) {
Program Error
2*b == a
if (x > y+10) {
a > b+10
x = 30, y = 15
z = 30
x = a, y = b
ERROR;
}
}

常用开源工具

- Java语言
  - Java Pathfinder

- C/C++语言
  - LLVM KLEE

- 二进制(binary)
  - angr

The Institute of Advanced Computing Technology

LLVM KLEE例子(1)

- 简单程序get_sign
  - 增加main函数，输入变量a符号化

```
int get_sign(int x) {
  if (x == 0)
    return 0;

  if (x < 0)
    return -1;
  else
    return 1;
}
```

```
int main() {
  int a;
  klee_make_symbolic(&a, sizeof(a), "a");
  return get_sign(a);
}
```

The Institute of Advanced Computing Technology

LLVM KLEE例子(1)

- 编译运行KLEE
  - 自动生成3个测试用例，覆盖不同的路径

```
$ ktest-tool --write-ints klee-last/test000001.ktest
ktest file : 'klee-last/test000001.ktest'
args       : ['get_sign.bc']
num objects: 1
object    0: name: 'a'
object    0: size: 4
object    0: data: 1

$ ktest-tool --write-ints klee-last/test000002.ktest
...
object    0: data: -2147483648

$ ktest-tool --write-ints klee-last/test000003.ktest
...
object    0: data: 0
```

The Institute of Advanced Computing Technology

基于静态和符号分析的服务缺陷检测

- 程序静态分析
  - 缓冲区溢出缺陷(Buffer Overflow)
  - 空指针(NULL Pointer)
  - 资源泄露(Resource Leak)
  - 异常根源追踪(Dependency Analysis)
- 符号执行分析
  - 自动测试用例生成，可用于程序属性的检测
    - 除0缺陷
    - 空指针缺陷
    - 后门漏洞(backdoor bug)

The Institute of Advanced Computing Technology

缓冲区溢出缺陷

- 静态分析
  - 定位缓冲区数组
  - 寻找对数据的操作，比如strcpy
  - 分析是否存在index的判断

```
char buf[80];

hp = gethostbyaddr(...);
strcpy(buf, hp->hp_hname);
```

The Institute of Advanced Computing Technology

资源泄露缺陷

- 静态分析
  - 定位各种资源
  - 寻找资源创建
  - 分析是否存在资源释放

```
1  public void test(File file, String enc) throws IOException {
2    PrintWriter out = null;
3    try {
4      try {
5        out = new PrintWriter(
6                  new OutputStreamWriter(
7                      new FileOutputStream(file), enc));
8      } catch (UnsupportedEncodingException ue) {
9        out = new PrintWriter(new FileWriter(file));
10     }
11     out.append('c');
12   } catch (IOException e) {
13   } finally {
14     if (out != null) {
15       out.close();
16     }
17   }
18 }
```

Figure 1: Code example adapted from ant.

The Institute of Advanced Computing Technology

除0缺陷

- 静态分析和符号执行分析
  - 定位除法操作
  - 分析前置的条件分支变量C
  - 判断y=0 && C是否可满足

```
Void fun(int x, int y)
{   float z;
    if(x>5)
    { if(y<3)
        {   z= x/y; }
    }
}
```

The Institute of Advanced Computing Technology

Q&A

The Institute of Advanced Computing Technology