Symbolic Execution

Zhaoguo Wang

Adapted From:

<<A Survey of Symbolic Execution Techniques>>

Predicate Logic（谓词逻辑）

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants

A Quick Recap

```
void swap(bool& a, bool& b)
{
      a = a ^ b;
      b = b ^ a;
      a = a ^ b;
}
```

We want to prove that the program really swaps the value of a and b.

A Quick Recap

```
void swap(bool& a, bool& b)
{
      a = a ^ b;
      b = b ^ a;
      a = a ^ b;
}
```

$$(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0)) \land$$
$$(B1 \leftrightarrow B0) \land$$
$$(A2 \leftrightarrow A1) \land$$
$$(B2 \leftrightarrow (A1 \land \neg B1) \lor (\neg A1 \land B1)) \land$$
$$(A3 \leftrightarrow (A2 \land \neg B2) \lor (\neg A2 \land B2)) \land$$
$$(B3 \leftrightarrow B2) \rightarrow$$
$$(A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3)$$

Prove it is a tautology

A Quick Recap

```
void swap(bool& a, bool& b)
{
      a = a ^ b;
      b = b ^ a;
      a = a ^ b;
}
```

$\Rightarrow$

$$(A1 \leftrightarrow (A0 \land \neg B0) \lor (\neg A0 \land B0))\land$$
$$(B1 \leftrightarrow B0)\land$$
$$(A2 \leftrightarrow A1)\land$$
$$(B2 \leftrightarrow (A1 \land \neg B1) \lor (\neg A1 \land B1))\land$$
$$(A3 \leftrightarrow (A2 \land \neg B2) \lor (\neg A2 \land B2))\land$$
$$(B3 \leftrightarrow B2)\land$$
$$\neg((A3 \leftrightarrow B0) \land (A0 \leftrightarrow B3))$$

Prove it is unsatisfiable with SAT solver

Predicate Logic

```
void swap(int& a, int& b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

$$
\begin{aligned}
&(\forall A0)(\forall A1)(\forall A2)(\forall A3)(\forall B0)(\forall B1)(\forall B2)(\forall B3)(\\
&\quad (A1 = xor(A0, B0) \wedge \\
&\quad B1 = B0 \wedge \\
&\quad A2 = A1 \wedge \\
&\quad B2 = xor(A1, B1) \\
&\quad B3 = B2 \\
&\quad A3 = xor(A2, B2) \\
&\quad) \rightarrow \\
&\quad ((A3 = B0) \wedge (B3 = A0)) \\
&)
\end{aligned}
$$

We can generate predicate logic WFF similarly

Predicate Logic

```
void swap(int& a, int& b)
{
        a = a ^ b;
        b = b ^ a;
        a = a ^ b;
}
```

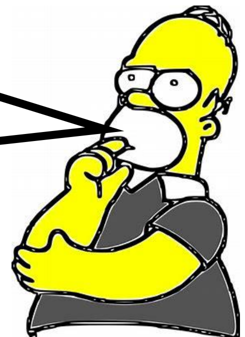
$$
\begin{aligned}
&(\exists A0)(\exists A1)(\exists A2)(\exists A3)(\exists B0)(\exists B1)(\exists B2)(\exists B3)(\\
&\qquad (A1 = xor(A0, B0) \land \\
&\qquad B1 = B0 \land \\
&\qquad A2 = A1 \land \\
&\qquad B2 = xor(A1, B1) \\
&\qquad B3 = B2 \\
&\qquad A3 = xor(A2, B2) \\
&\qquad) \land \\
&\qquad \neg((A3 = B0) \land (B3 = A0)) \\
&\quad)
\end{aligned}
$$

Prove it is unsat with SMT solver

Symbolic Execution (符号执行)

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

If the program has many "if", the WFF will be very long

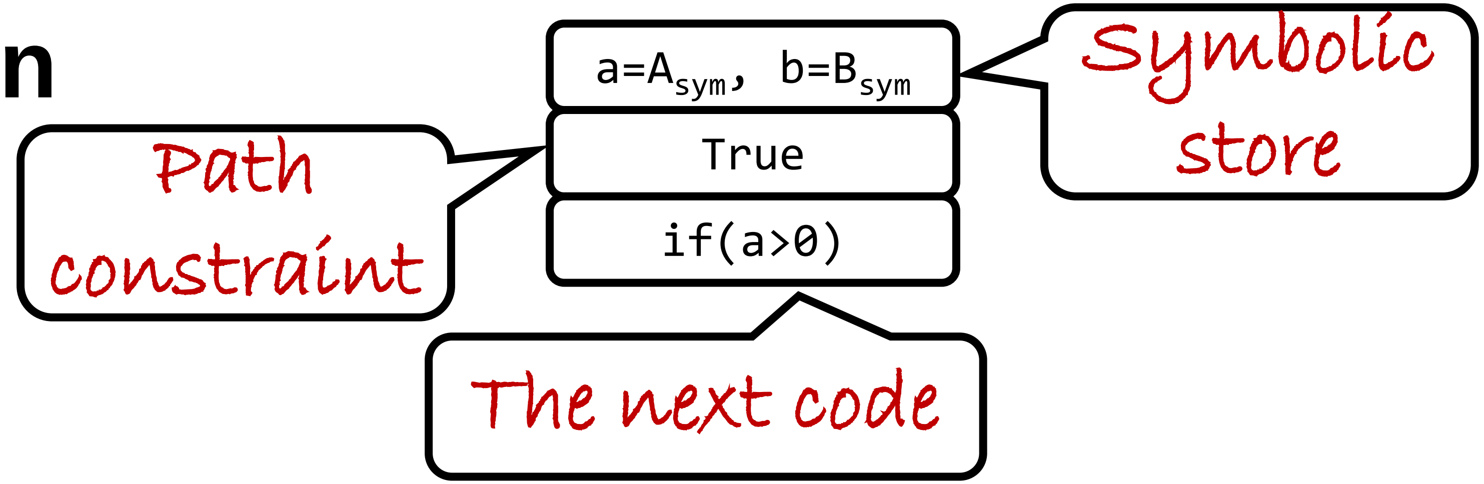


Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

We can generate multiple WFFs for different execution paths.

Symbolic Execution



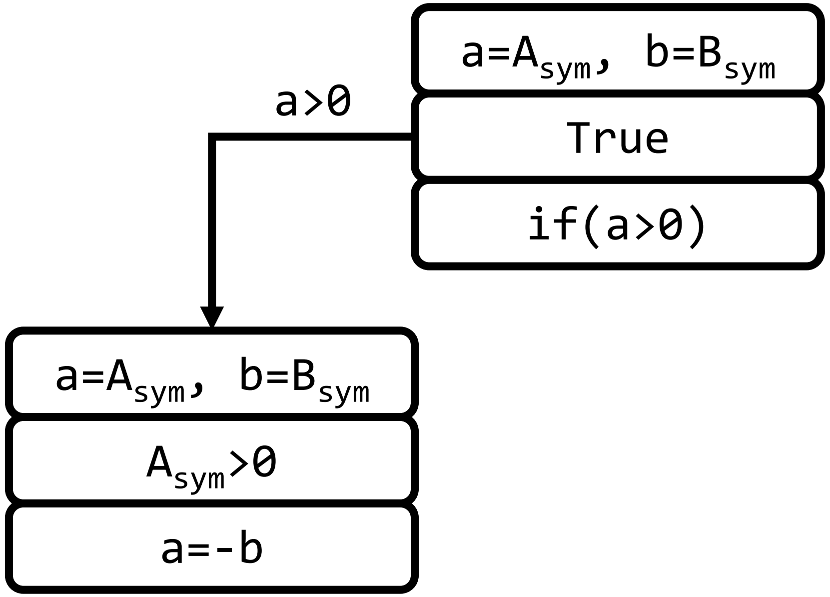

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

Symbolic execution represents value of variables with symbols.
It maintains 3 things to generate WFF.

- **Path constraints:** **the path condition**
- **Next code:** **the next instruction to be executed**
- **Symbolic store:** **the value (symbolic expression) for each variable**

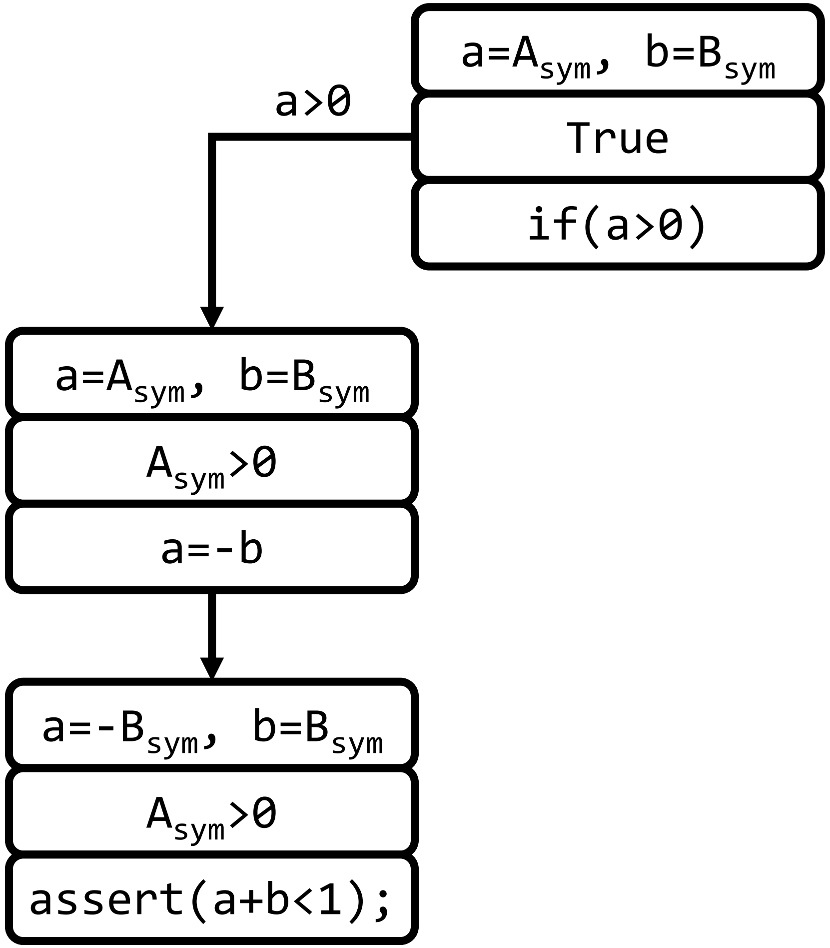
Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```



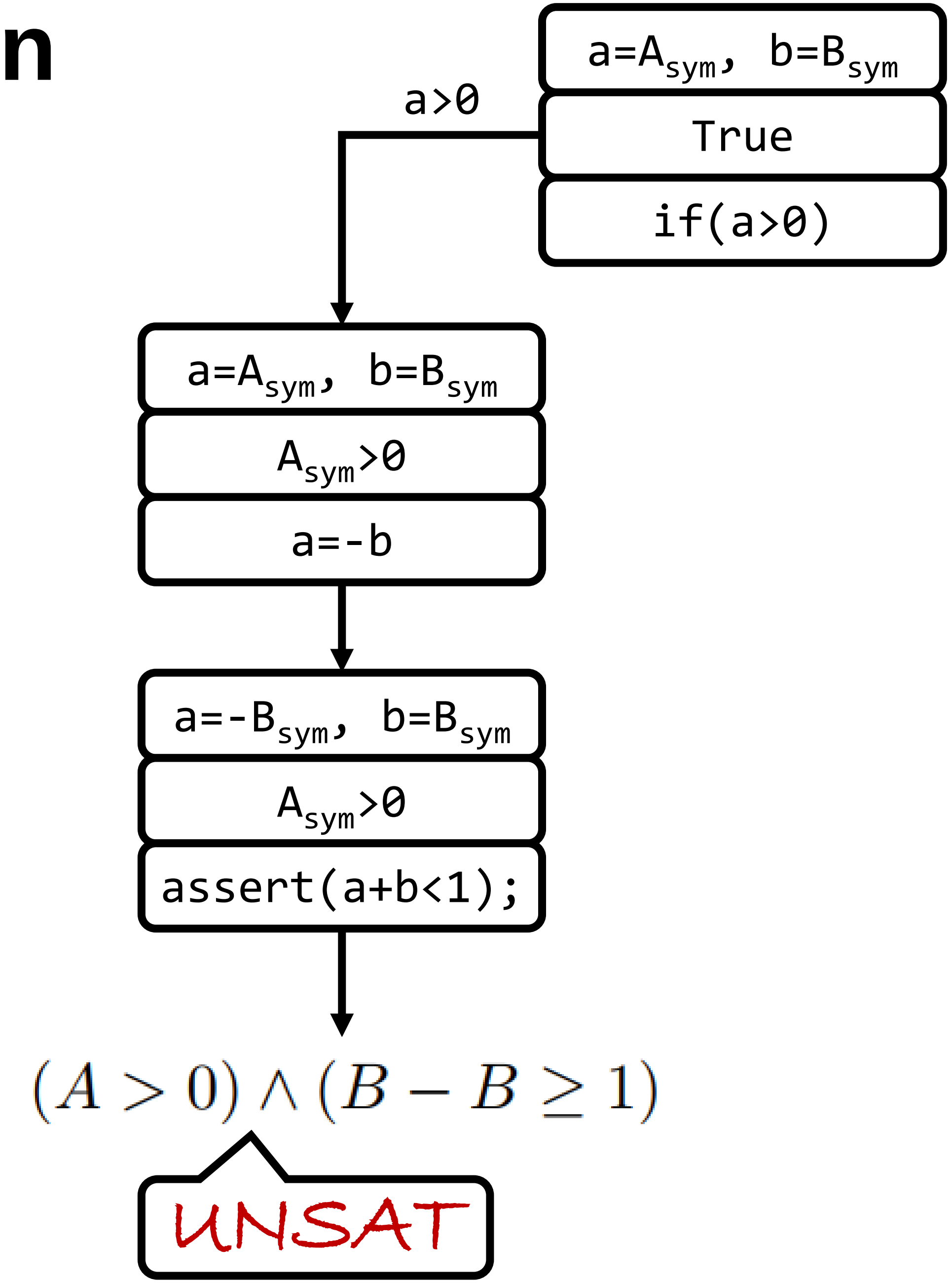
Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```



Symbolic Execution

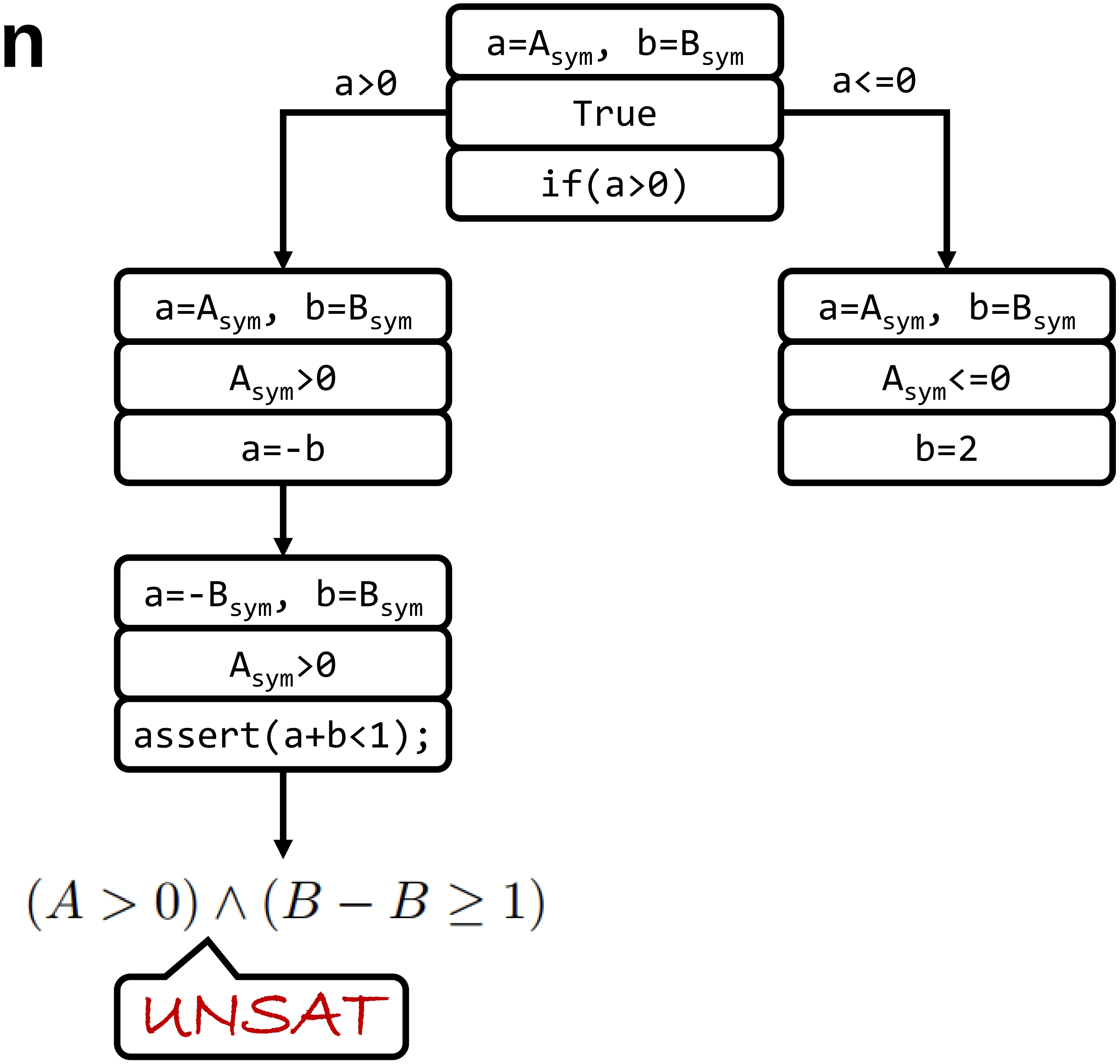
```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```




```
context ctx;
expr A = ctx.int_const("A");
expr B = ctx.int_const("B");
solver s(ctx);
s.add(A > 0);
s.add(B - B >= 1);
std::cout << s.check();
```

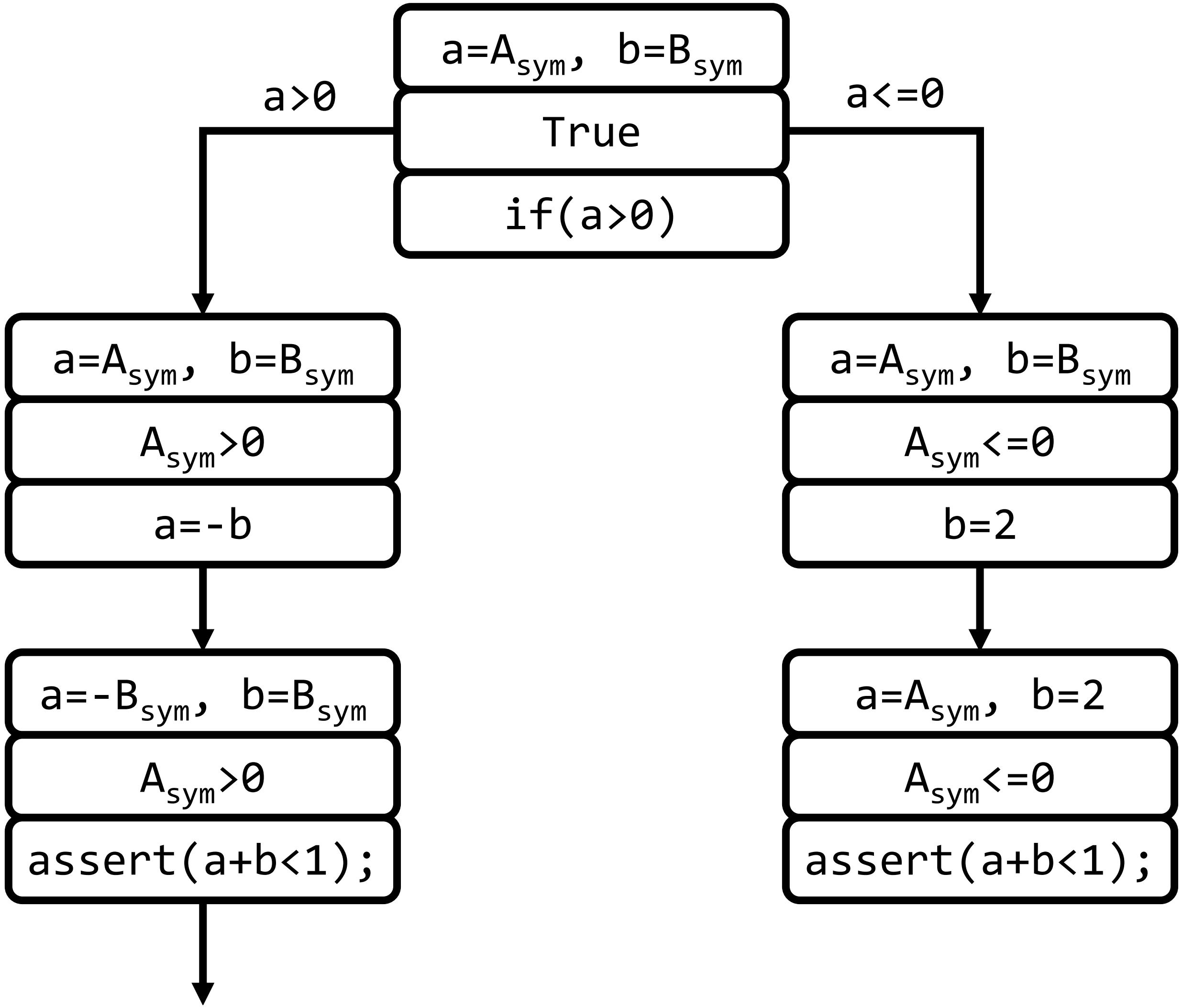
Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```



Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```



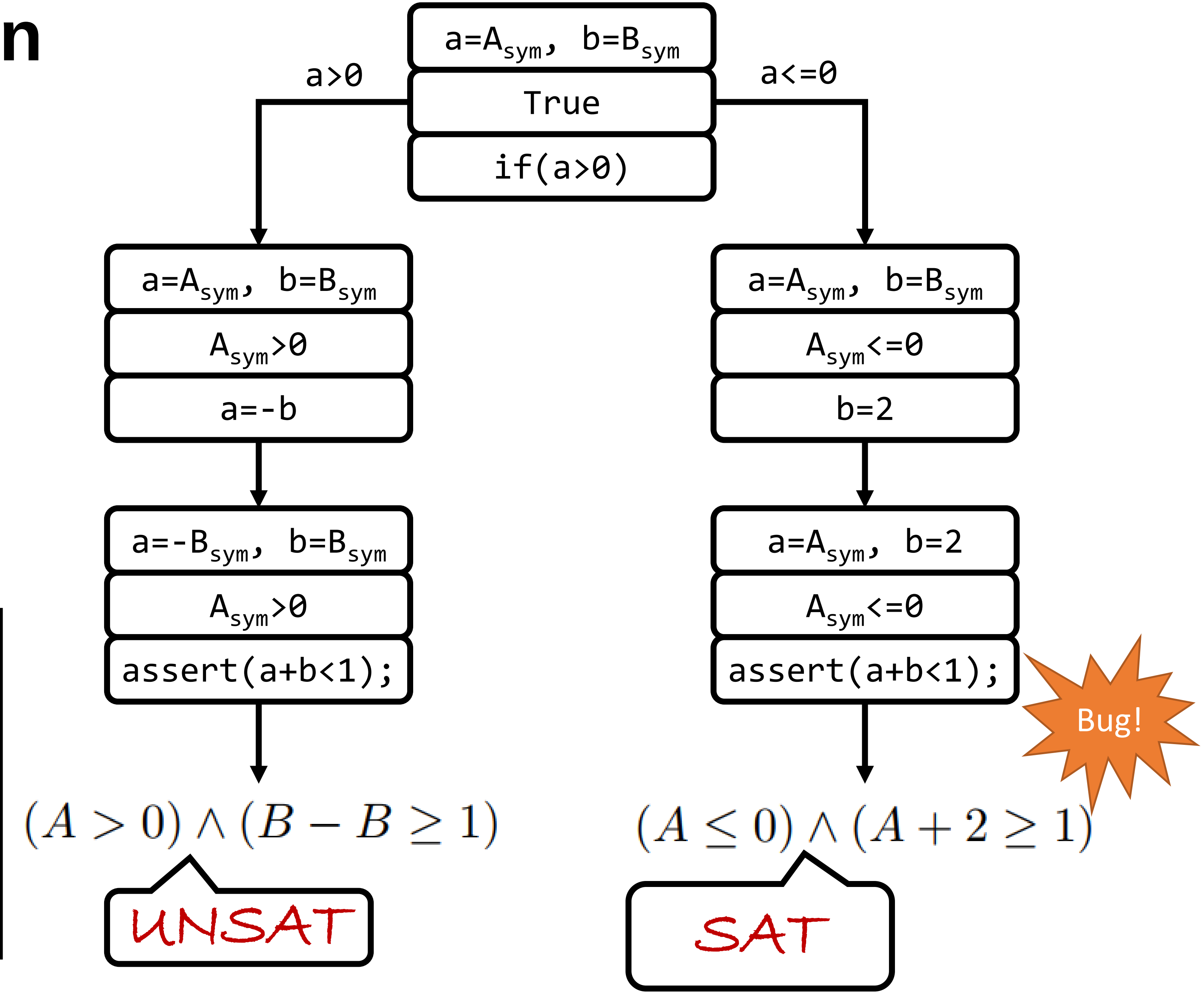
$$(A > 0) \wedge (B - B \geq 1)$$

UNSAT

Symbolic Execution

```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

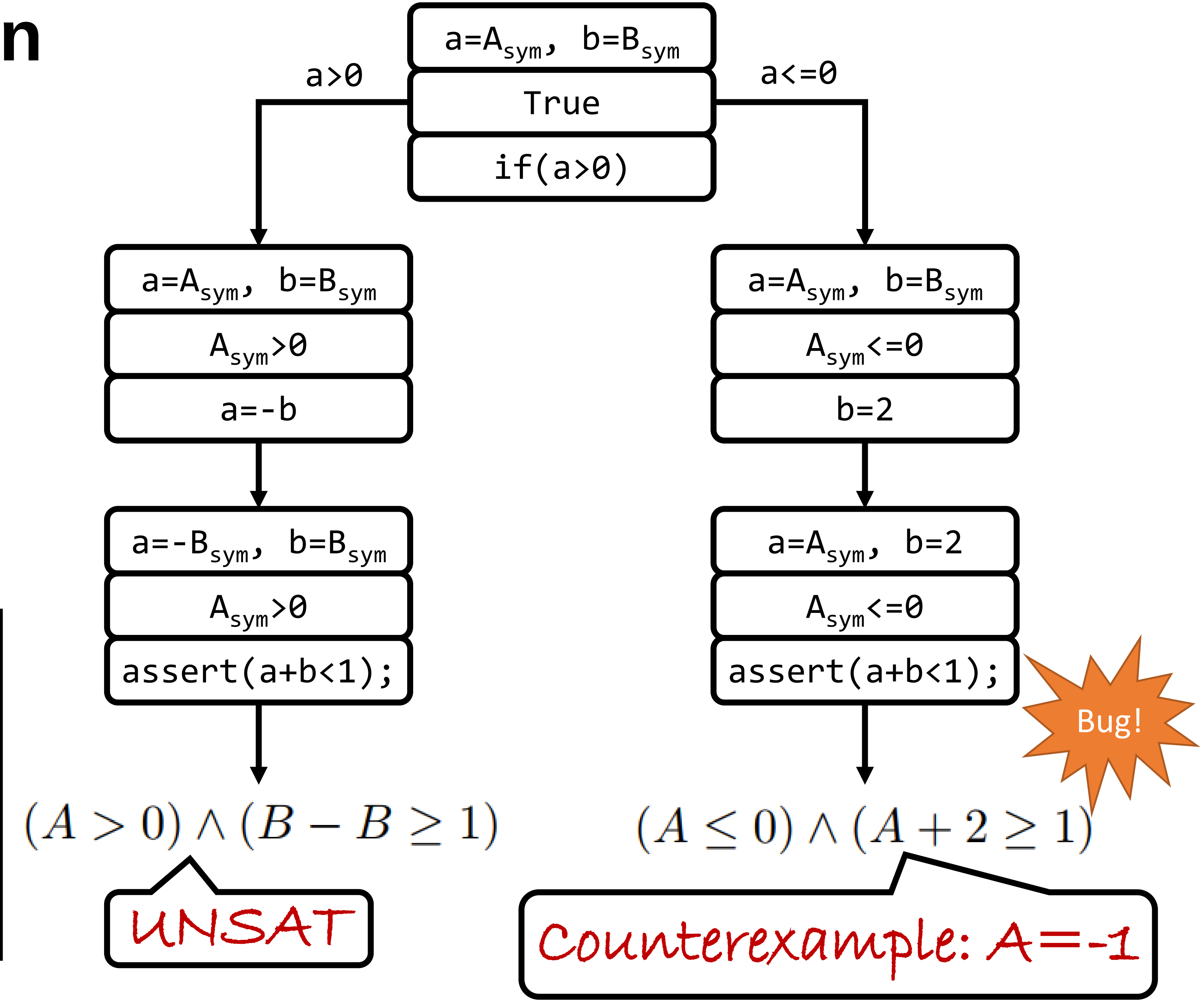
```
expr A = ctx.int_const("A");
expr B = ctx.int_const("B");
solver s(ctx);
s.add(A <= 0);
s.add(A+2 >=1);
std::cout << s.check();
std::cout << s.get_model();
```



Symbolic Execution

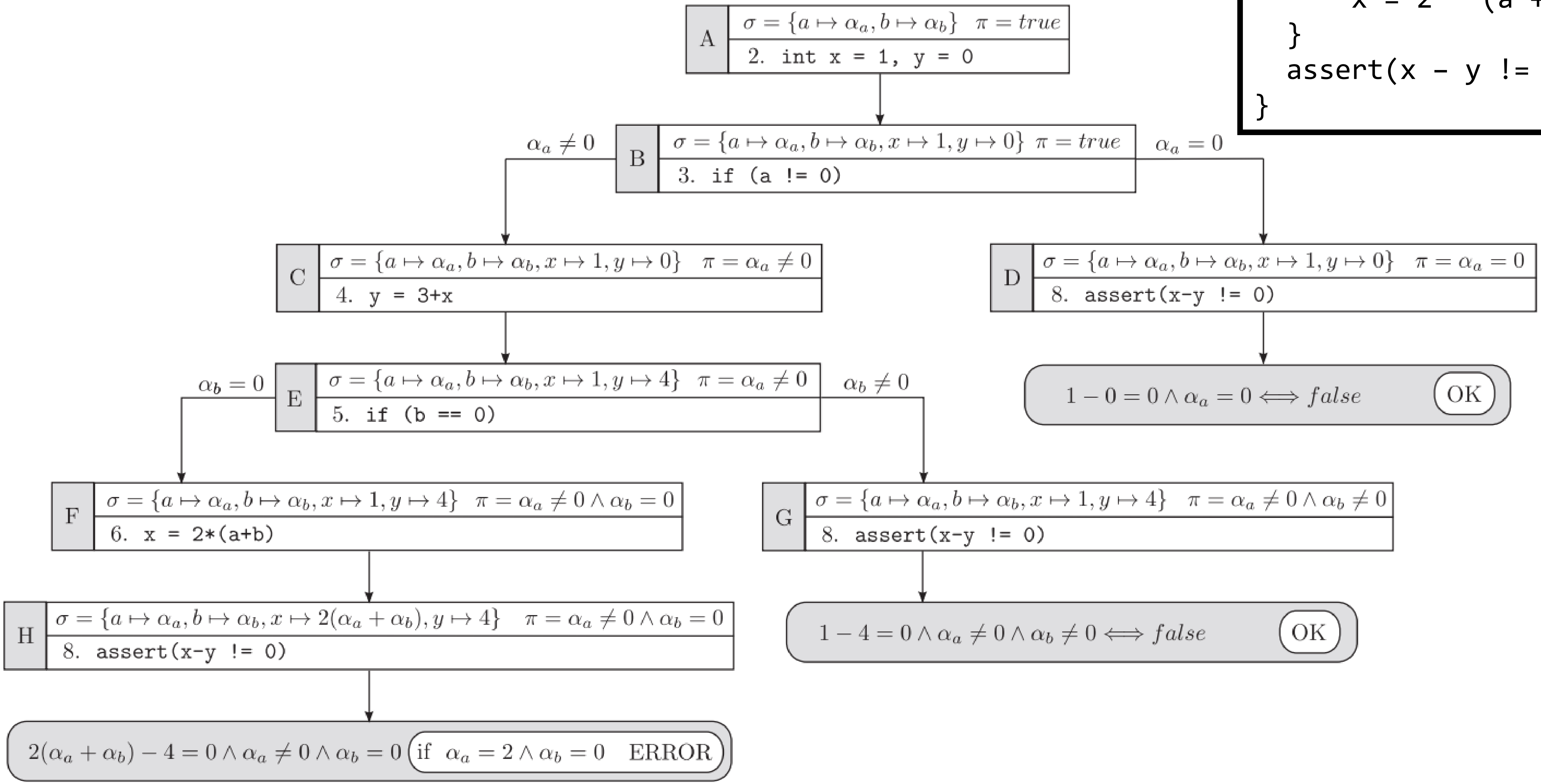
```
void func(int a, int b) {
  if(a > 0)
    a = -b;
  else
    b = 2;
  assert(a + b < 1);
}
```

```
expr A = ctx.int_const("A");
expr B = ctx.int_const("B");
solver s(ctx);
s.add(A <= 0);
s.add(A+2 >=1);
std::cout << s.check();
std::cout << s.get_model();
```



Example

```
void func(int a, int b) {
  int x = 1, y = 0;
  if(a != 0) {
    y = 3 + x;
    if(b == 0)
      x = 2 * (a + b);
  }
  assert(x - y != 0);
}
```



Thanks & Questions