

Temas Avanzados

Juan Pablo Galeotti - 2017

En Algo1 ...

- A lo largo de la materia hicimos (entre otras cosas)
 - Demostración de corrección de programas
 - Definición de test cases para lograr mayor cobertura
- ¿Es posible **automatizar** (aunque sea en parte) alguna de estas tareas?

Verificación de Programas

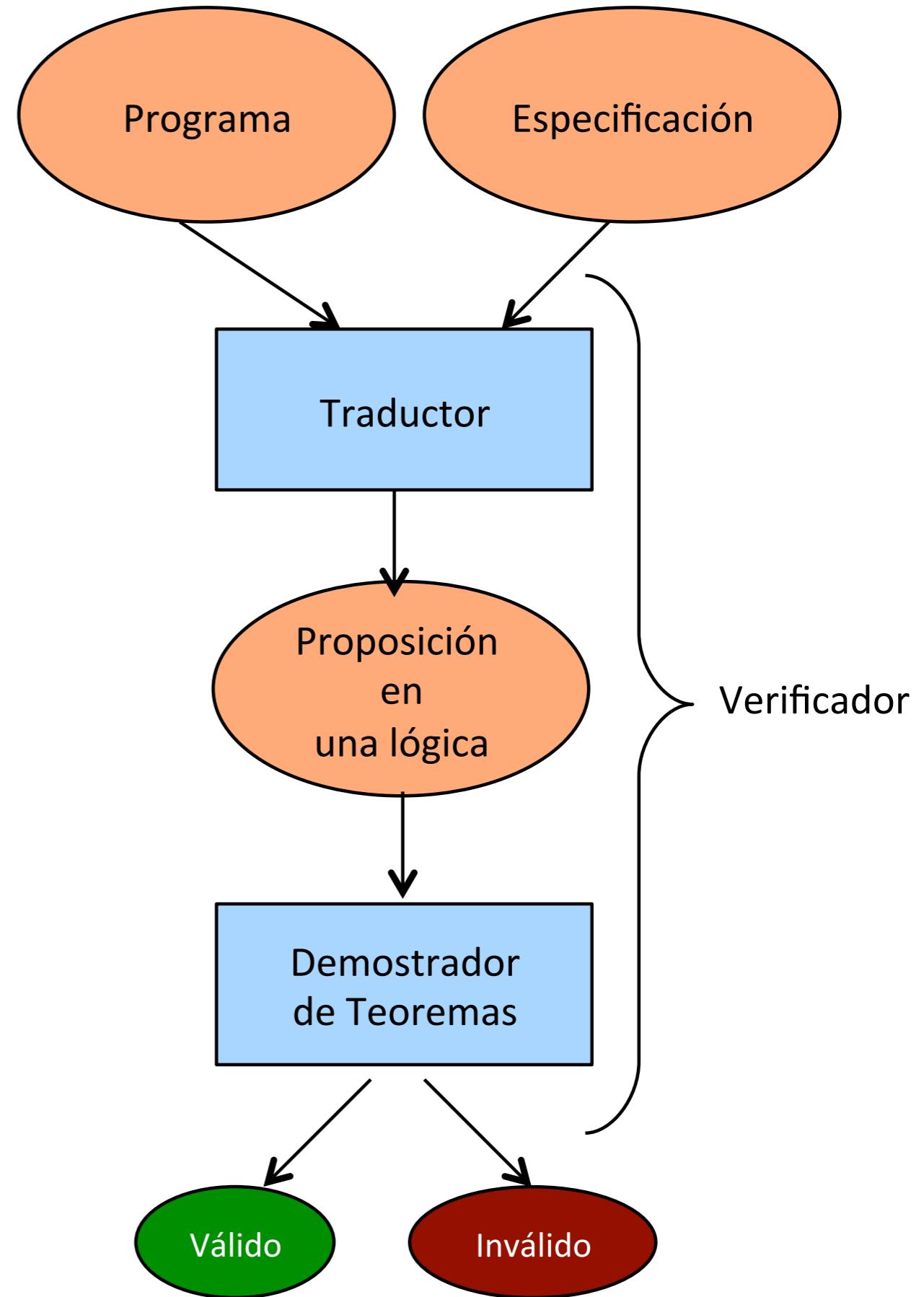
- Tenemos un **sistema formal** para demostrar la corrección de un programa respecto de su especificación
 - Dados el programa y su especificación: $\{P\} S \{Q\}$
 - ... calculamos $wp(S, Q)...$
 - ... y tratamos de demostrar que $P \Rightarrow wp(S, Q)$
- Esto abre la posibilidad de tener **verificadores automáticos** de programas

Demostrador de Teoremas

- La demostración de implicaciones (ie Teoremas) de lógica de primer order (existe, para todo) es **indecidable**.
- Sin embargo, en la práctica se pueden demostrar algunos teoremas (y si tenemos la demostración, el teorema vale)



Verificación Automática de Programas



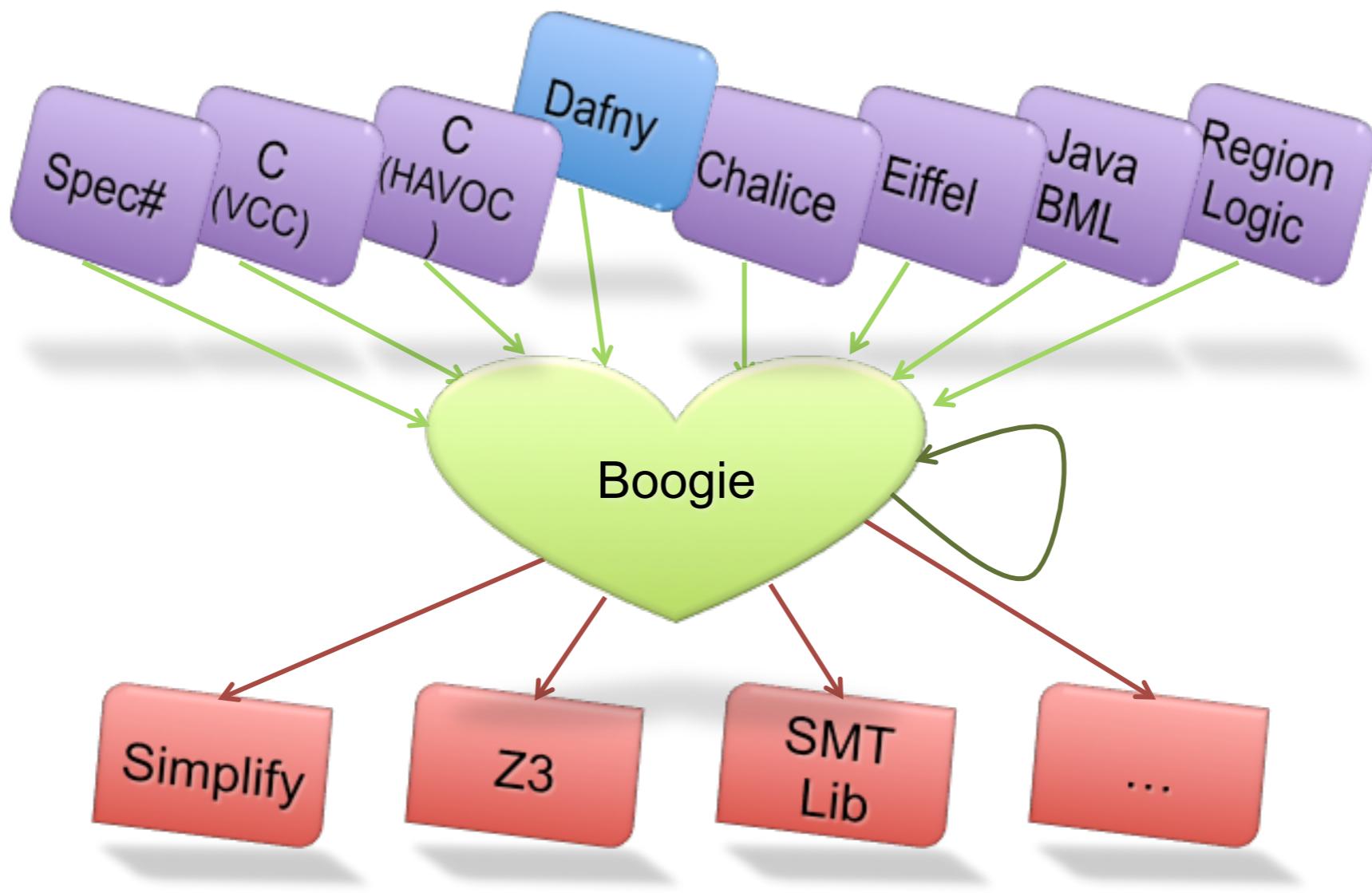
Dafny

research.microsoft.com/dafny

- Existen diversos verificadores automáticos de programas. Uno de ellos es Dafny, desarrollado por Microsoft Research.
- Utiliza un lenguaje intermedio llamado **Boogie** y un demostrador de teoremas llamado **Z3**.
- Dafny tiene su propio lenguaje de programación (parecido a SmallLang)



Boogie como IL



<https://rise4fun.com/Dafny/>

dafny

Microsoft Research

Is this program correct?

```
1 method M(n: int)
2 {
3     // count up to 'n'; will this program terminate?
4     var i := 0;
5     while i < n
6     {
7         i := i + 1;
8     }
9 }
```

10



tutorial

home video permalink

'▶' shortcut: Alt+B

Dafny program verifier finished with 1 verified, 0 errors

Demo: Valor Absoluto

```
method Abs(x: int) returns (r: int)
ensures (x < 0 && r == -x) || (x >= 0 && r == x)
{
    if( x > 0 )
    {
        r := x;
    }
    else
    {
        r := -x;
    }
}
```

Demo: Búsqueda Lineal

```
method Buscar(a: array<int>, x: int) returns (ret: bool)

    requires a != null
    ensures ret == exists i :: (0 <= i < a.Length && a[i] == x)

{
    var i := 0;

    while( i < a.Length && a[i] != x )
        invariant 0 <= i <= a.Length && forall j :: (0 <= j < i ==> a[j] != x)
    {
        i := i+1;
    }

    return i < a.Length;
}
```

```
method BusquedaBinaria(a: array<int>, x: int) returns (ret: bool)

    requires a != null && a.Length > 1
    requires ordenado(a) && a[0] <= x < a[a.Length-1]
    ensures ret == exists i :: (0 <= i < a.Length && a[i] == x)

{
    var i := 0;
    var j := a.Length - 1;

    while( i+1 < j )
        invariant 0 <= i < j < a.Length && a[i] <= x < a[j]
        decreases j-i
    {
        var k := (i+j) / 2;
        if (a[k] <= x)
        {
            i := k;
        }
        else
        {
            j := k;
        }
    }
    return a[i] == x;
}
```

Demo: Búsqueda Binaria

```
method Anular(a: array<int>)

requires a != null
modifies a

ensures forall j :: (0 <= j < a.Length && old(a)[j] <= 0 ==> a[j] == 0)
ensures forall j :: (0 <= j < a.Length && old(a)[j] >= 0 ==> a[j] == old(a)[j])

{
    var i := 0;

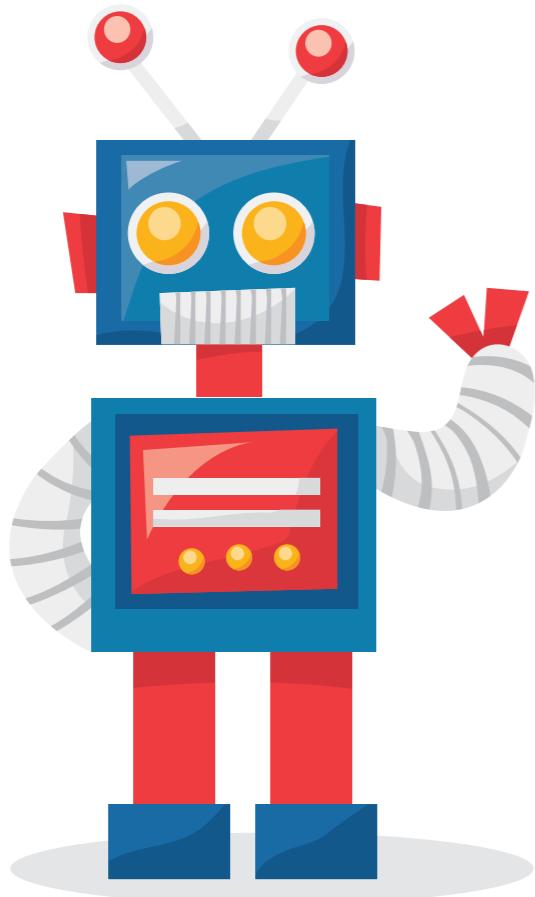
    while( i < a.Length )

        invariant 0 <= i <= a.Length
        invariant forall j :: (0 <= j < i && old(a)[j] <= 0 ==> a[j] == 0)
        invariant forall j :: (0 <= j < i && old(a)[j] >= 0 ==> a[j] == old(a)[j])
        decreases a.Length - i;

    {
        if( a[i] < 0 )
        {
            a[i] := 0;
        }
        i := i+1;
    }
}
```

Demo: Modificar una secuencia

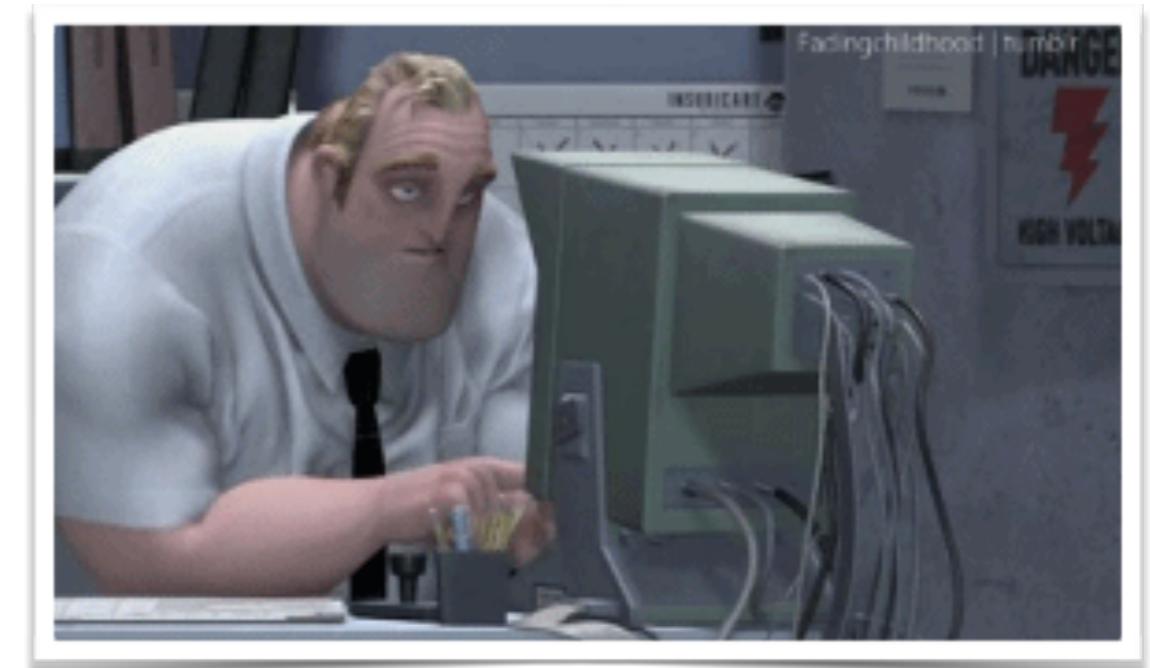
Testing



Creación
Manual

VS

Ejecución
Automática

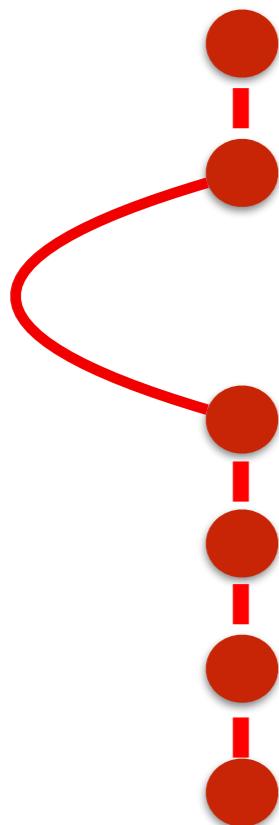


Random Testing



```
1. def testme(x, y):  
2.     if (y<0):  
3.         return -x  
4.     else:  
5.         z = x - y  
6.         if (y+z=10000):  
7.             raise Exception("error")  
8.         else:  
9.             return z
```

```
1. def testme(x, y):  
2.     if (y<0):  
3.         return -x  
4.     else:  
5.         z = x - y  
6.         if (y+z=10000):  
7.             raise Exception("error")  
8.         else:  
9.             return z
```



```
1. def testme(x, y):  
2.     if (y<0):  
3.         return -x  
4.     else:  
5.         z = x - y  
6.         if (y+z=10000):  
7.             raise Exception("error")  
8.         else:  
9.             return z
```

y>=0
x=10000



$2^{31} / 2^{32} = 50\%$
 $1 / 2^{32} \sim 0\%$

Ejecución Simbólica

AI Rescate!

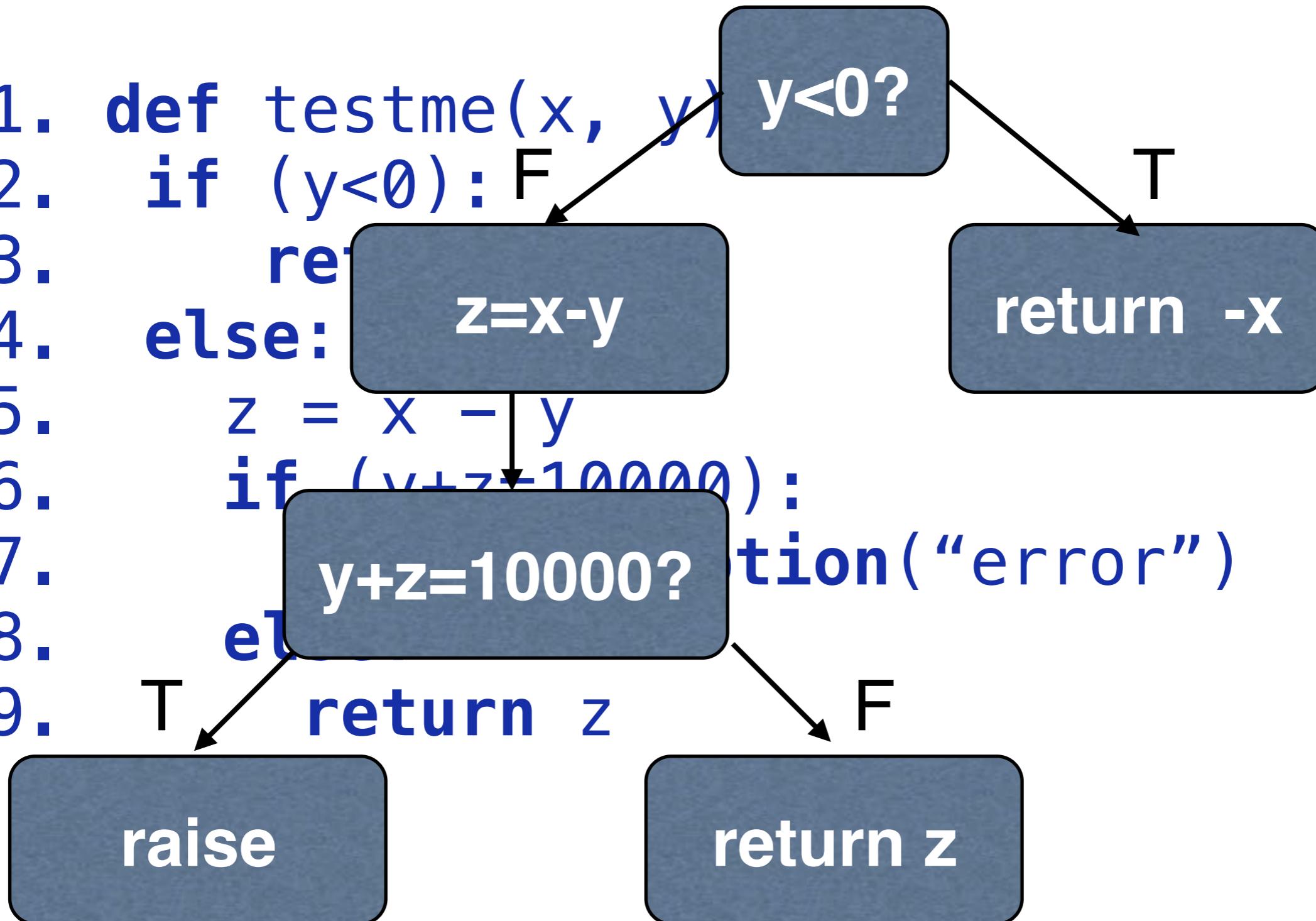
Necesitamos...

- Recolectar restricciones del código del programa
- Resolver restricciones para crear nuevos inputs



Ejecución Simbólica

```
1. def testme(x, y)
2.     if (y<0): F
3.         return -x
4.     else:
5.         z = x - y
6.         if (y+z=10000):
7.             raise "error"
8.         else:
9.             T
```

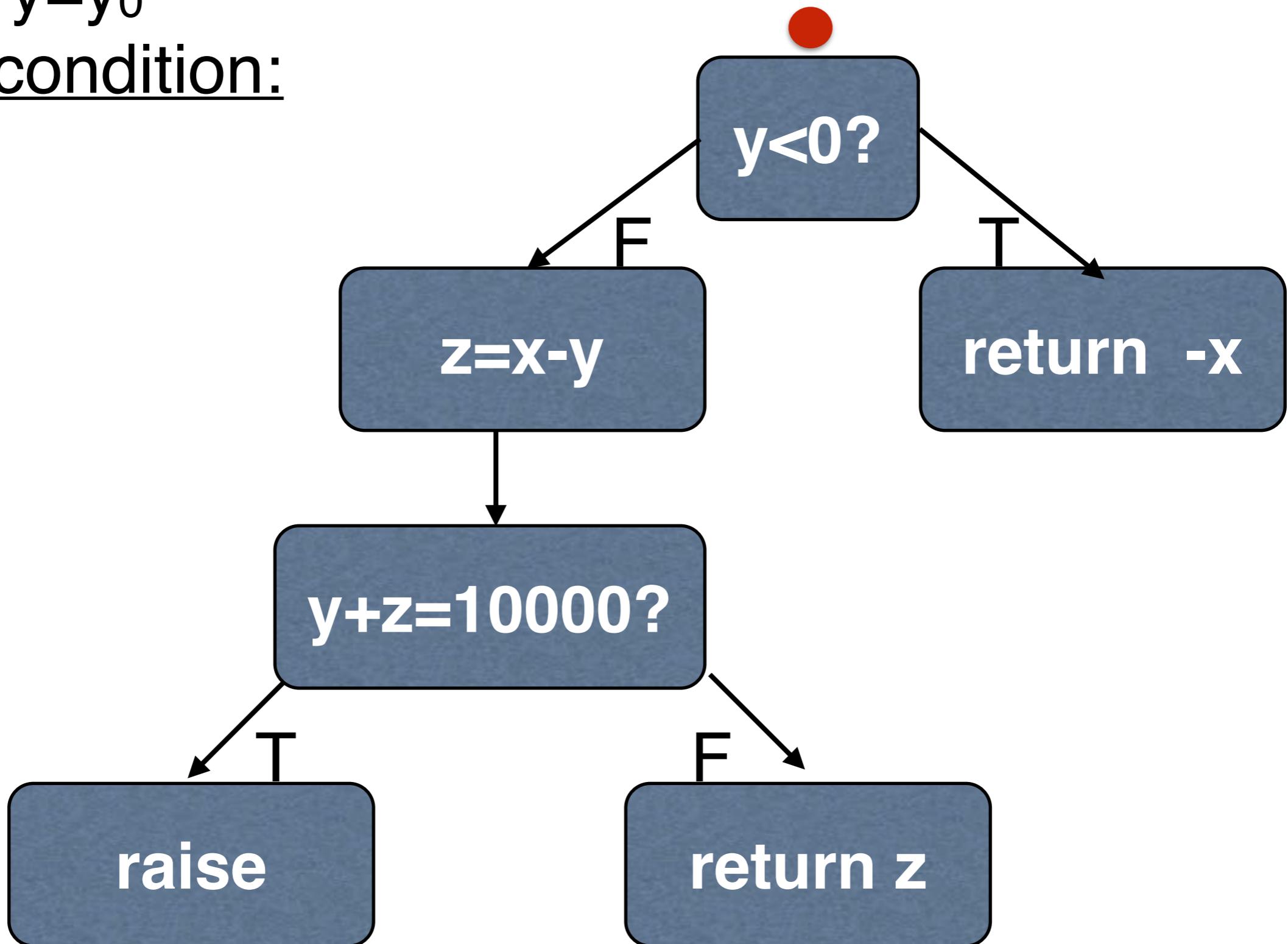


symbolic state:

$x=x_0, y=y_0$

path condition:

true

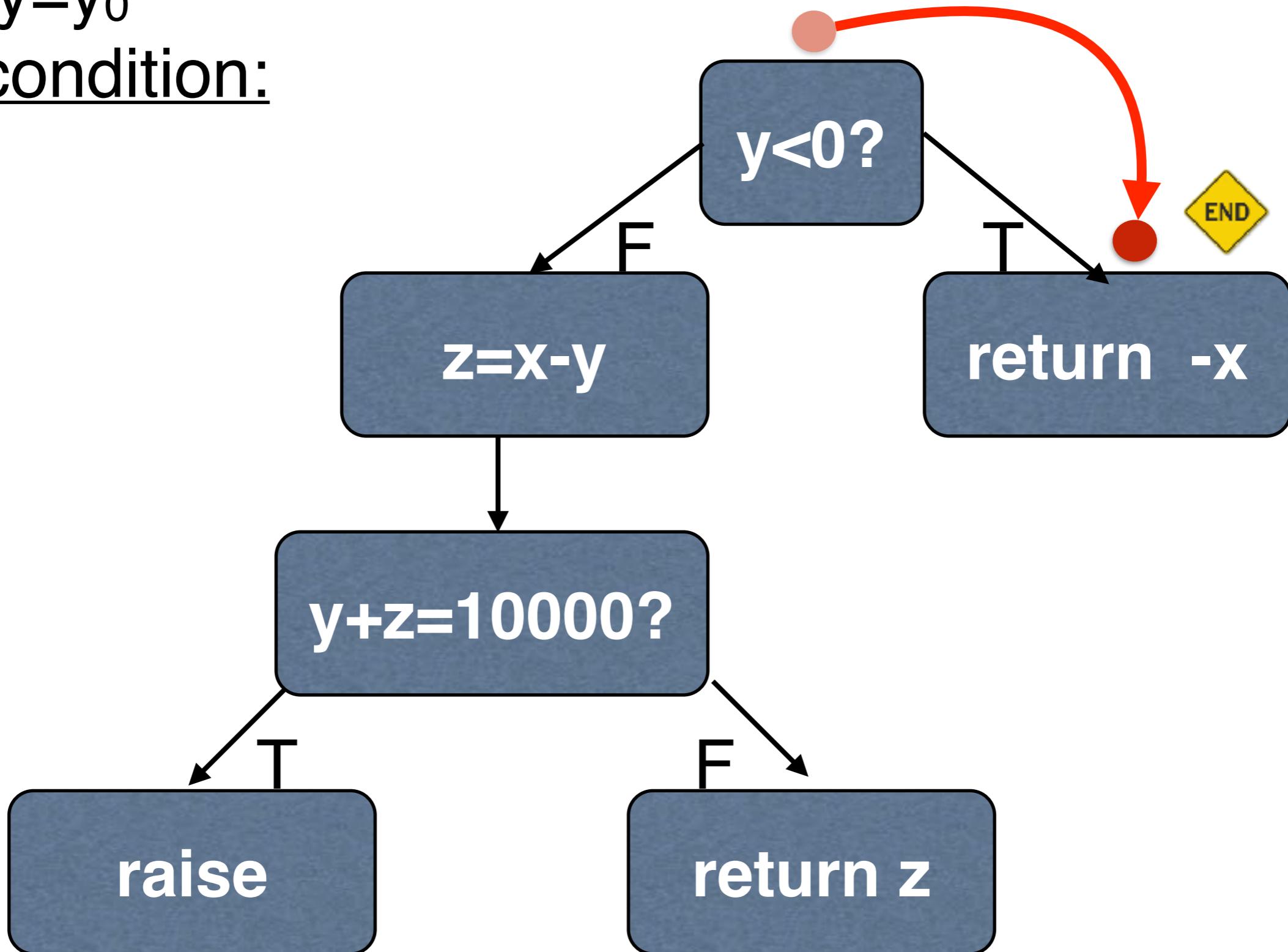


symbolic state:

$x=x_0, y=y_0$

path condition:

$y_0 < 0$

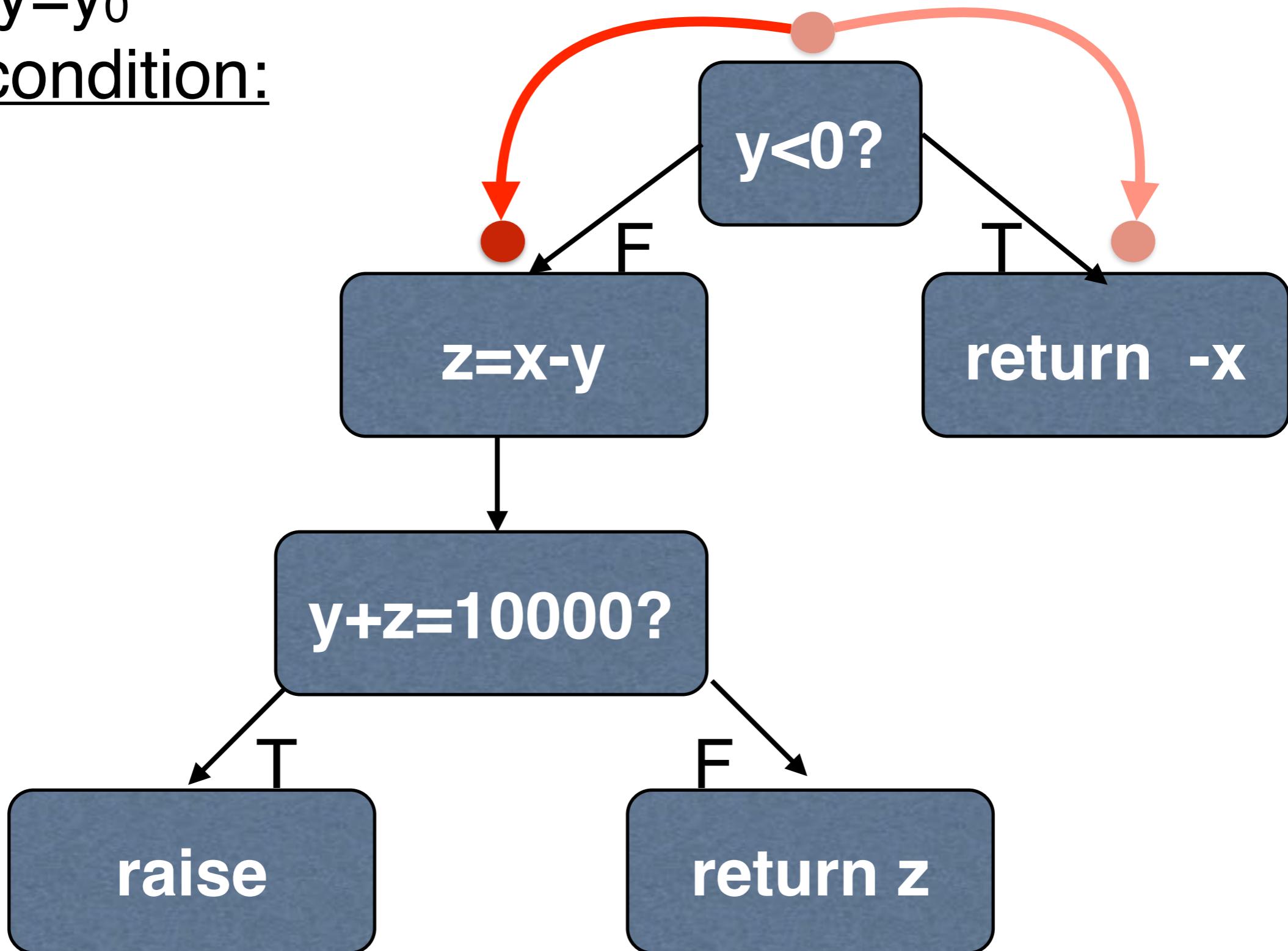


symbolic state:

$x=x_0, y=y_0$

path condition:

$y_0 \geq 0$

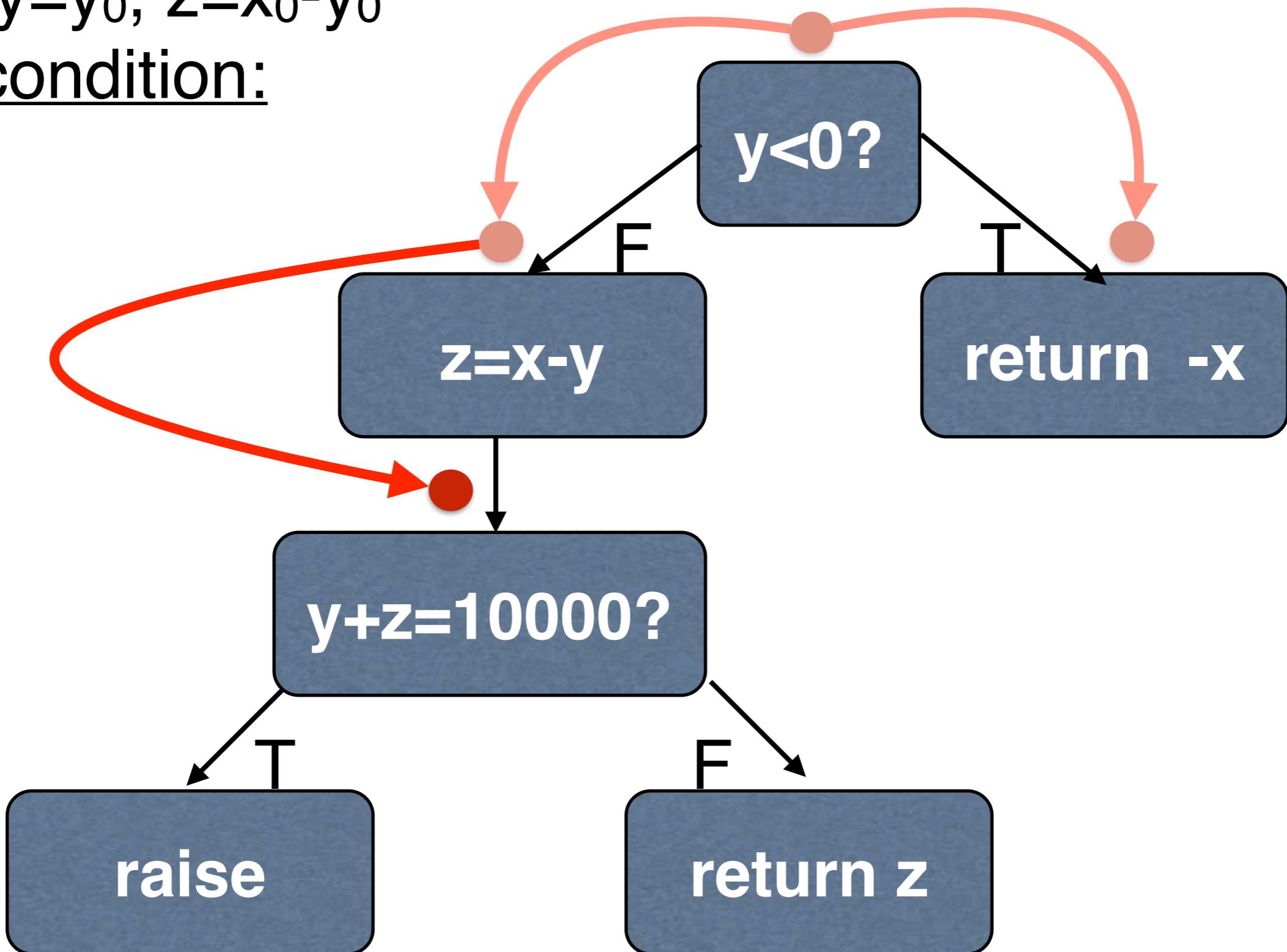


symbolic state:

$x=x_0, y=y_0, z=x_0-y_0$

path condition:

$y_0 \geq 0$



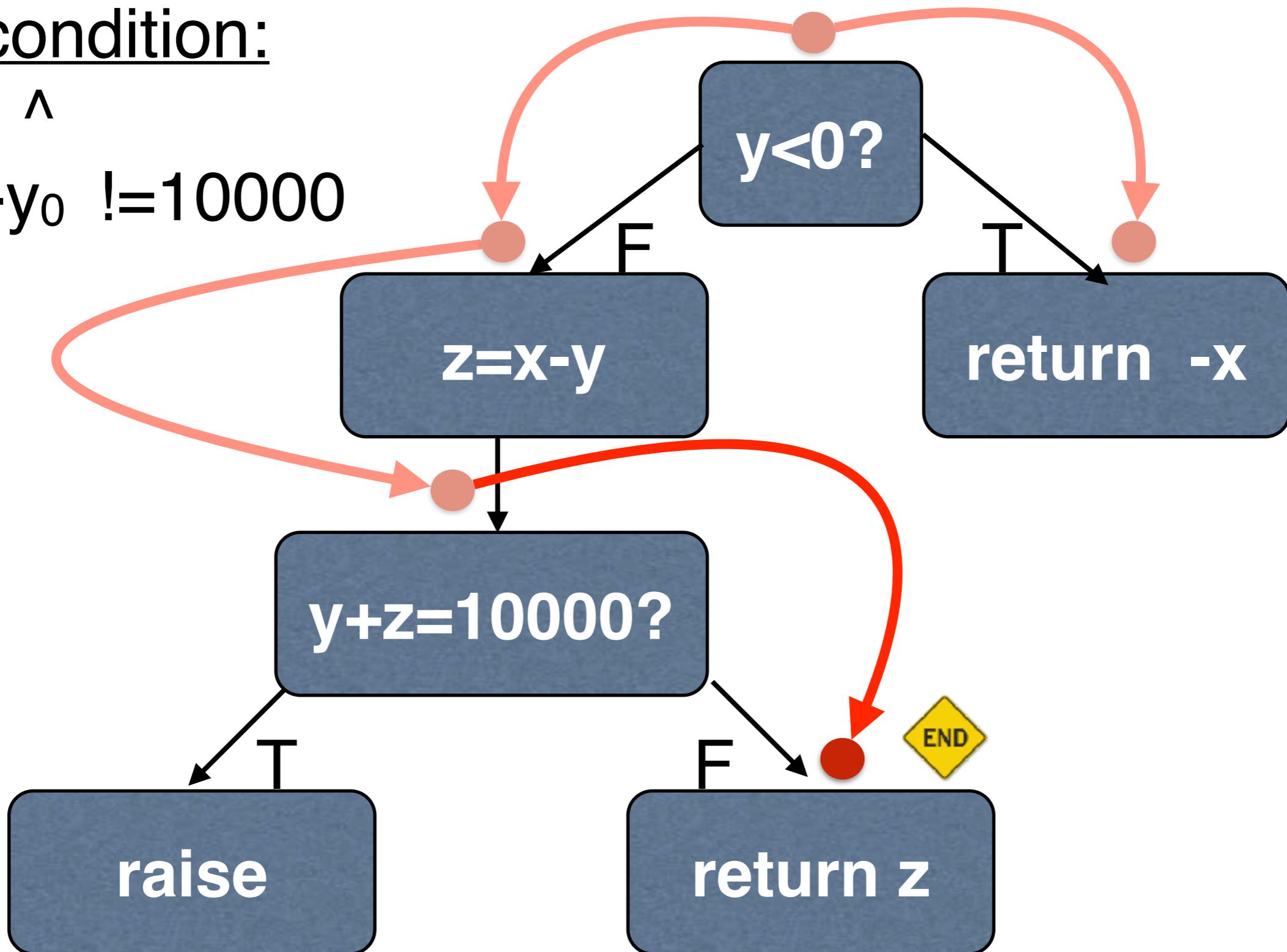
symbolic state:

$x=x_0$, $y=y_0$, $z=x_0-y_0$

path condition:

$y_0 \geq 0 \wedge$

$y_0+x_0-y_0 \neq 10000$



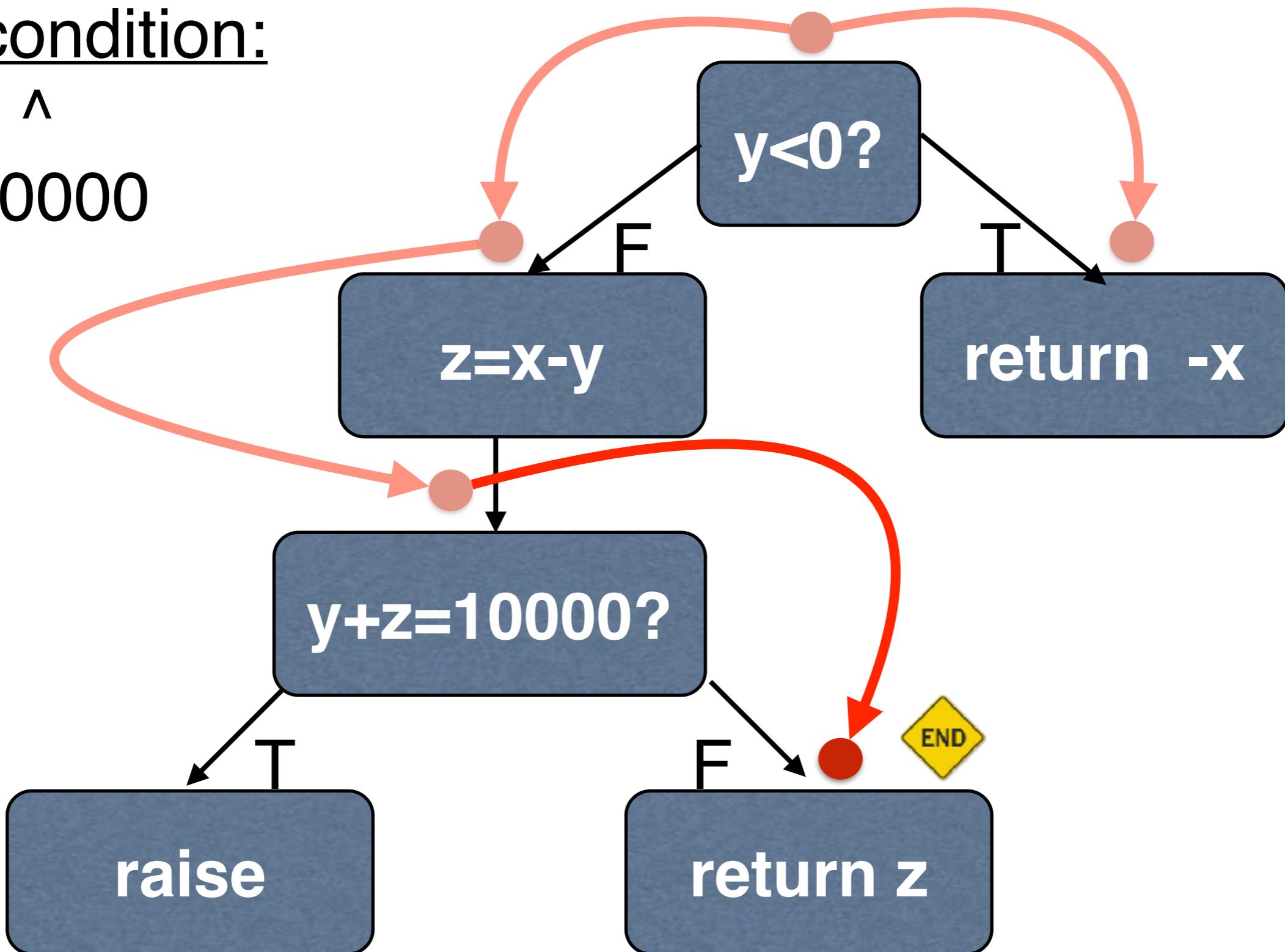
symbolic state:

$x=x_0$, $y=y_0$, $z=x_0-y_0$

path condition:

$y_0 \geq 0 \wedge$

$x_0 \neq 10000$



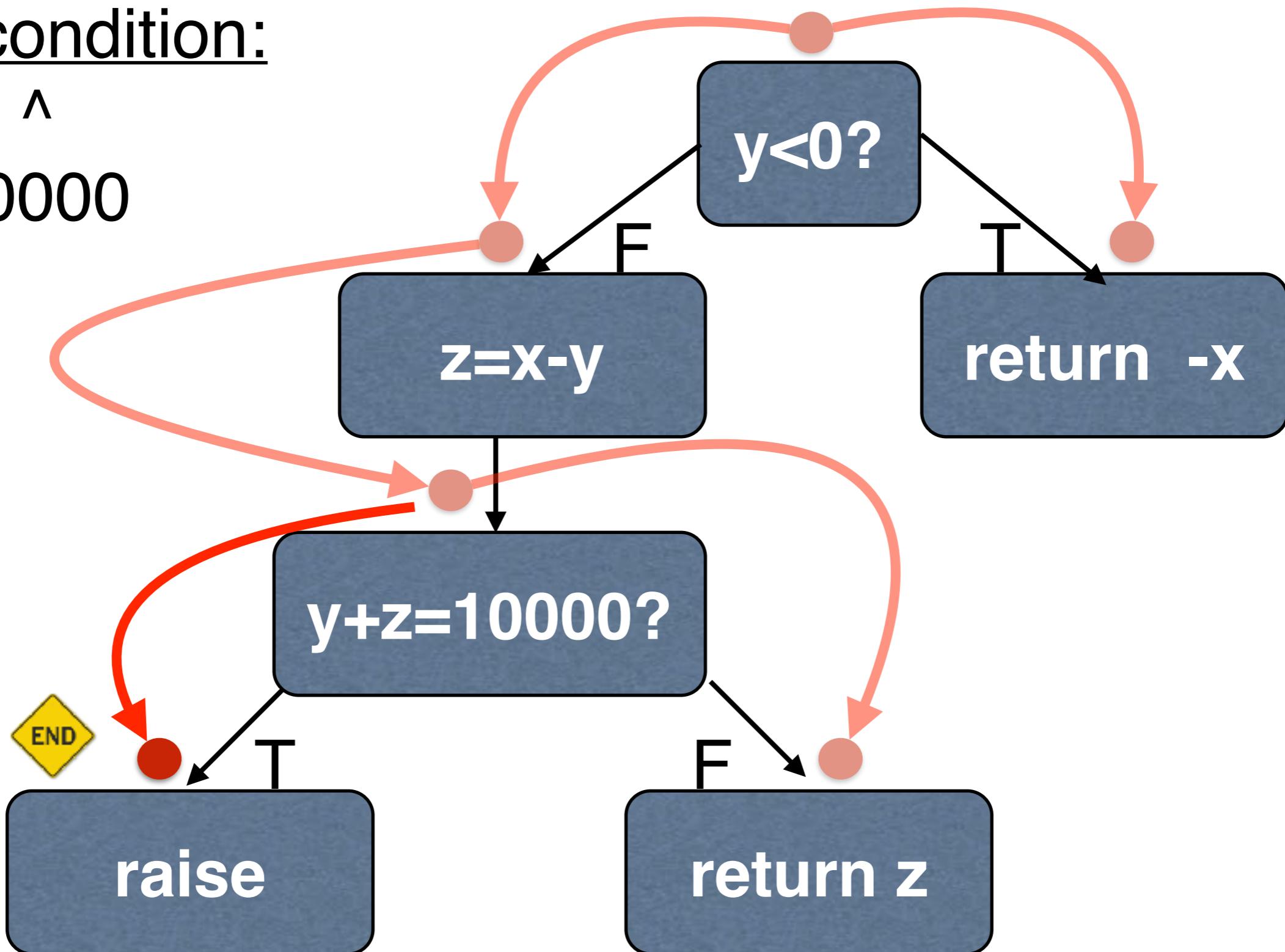
symbolic state:

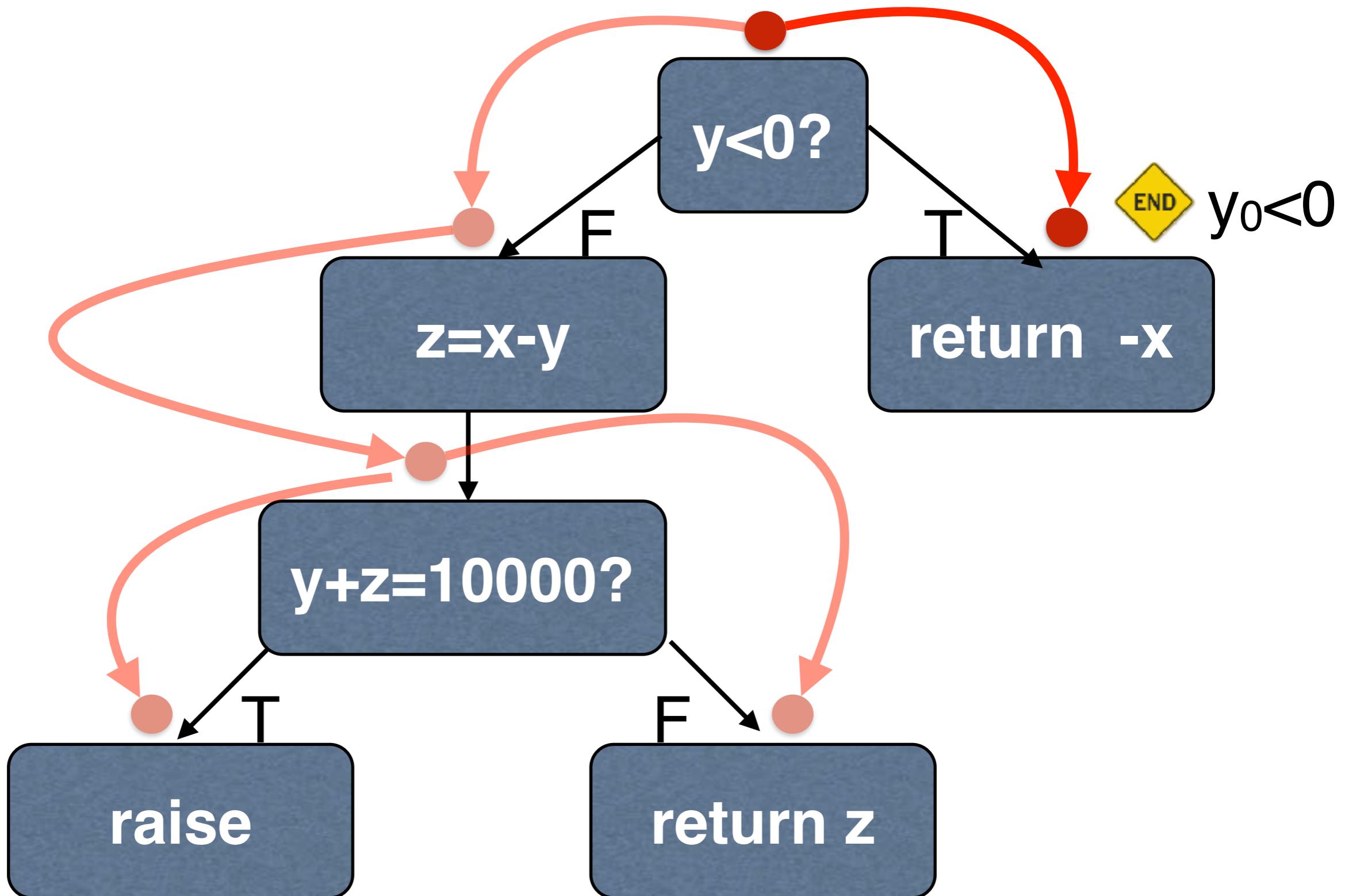
$x=x_0$, $y=y_0$, $z=x_0-y_0$

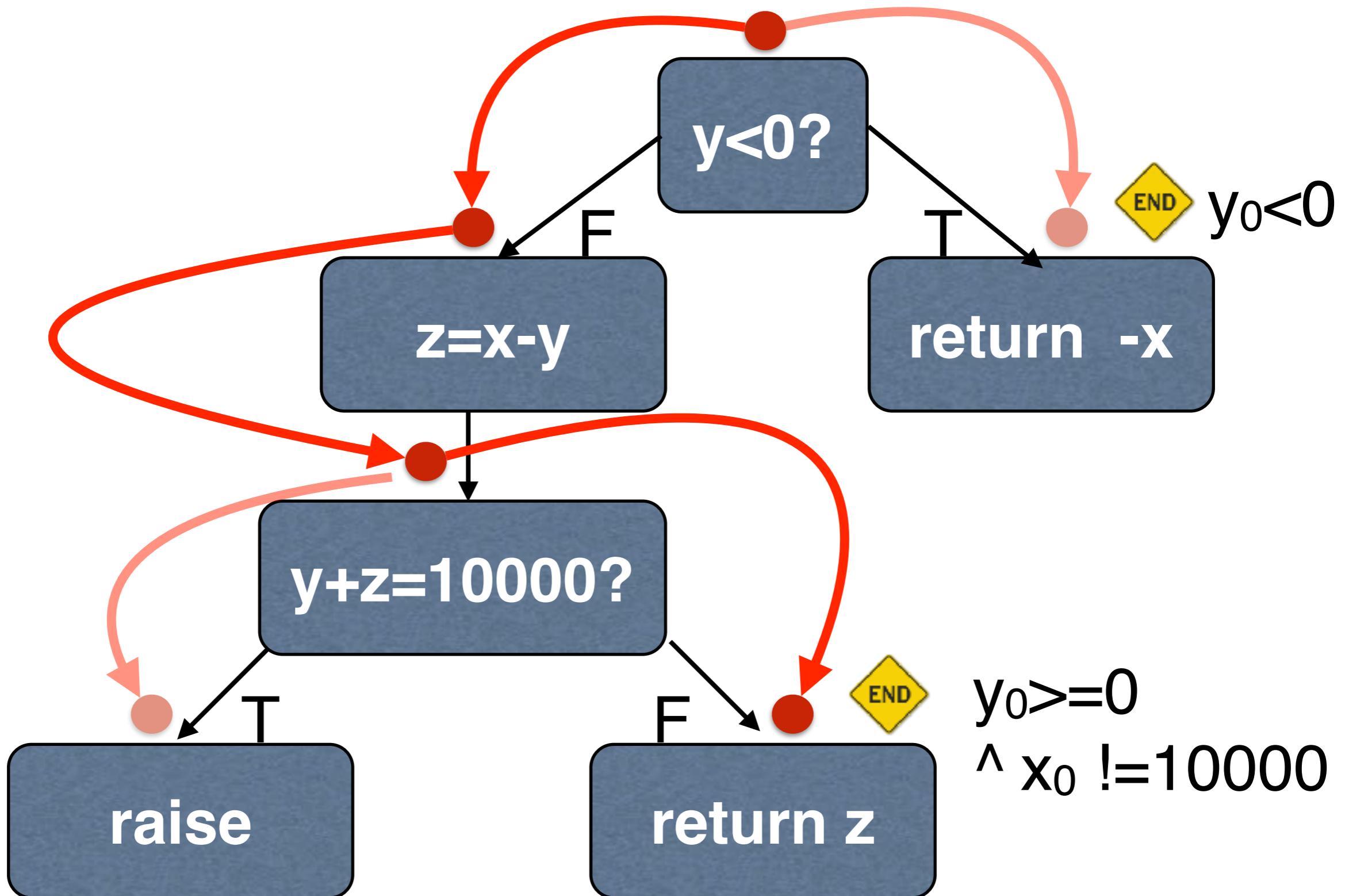
path condition:

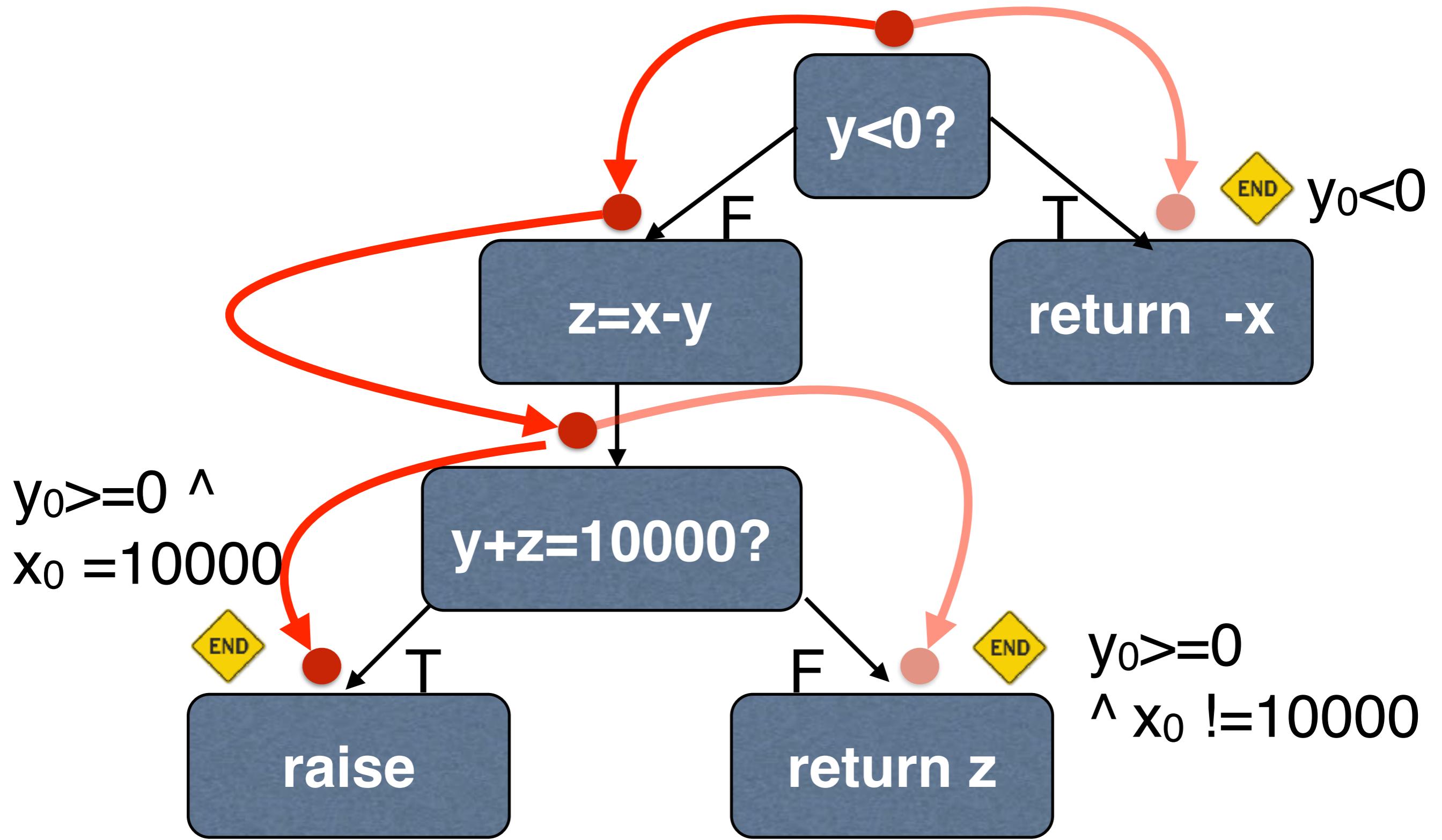
$y_0 \geq 0 \wedge$

$x_0 = 10000$







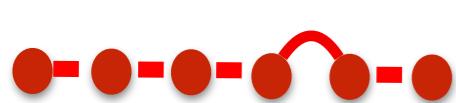
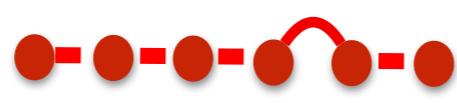
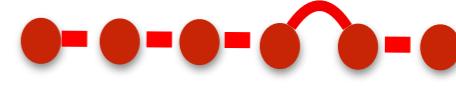


Constraint Solving



- Para poder hacer “constraint solving”, usamos un demostrador de teorema:
 - SAT
 - UNSAT
 - UNKNOWN/TIMEOUT
- Si es SAT, da un valor para cada variable

Constraint Solving

 $y_0 < 0$  $y_0 \geq 0 \wedge x_0 \neq 10000$  $y_0 \geq 0 \wedge x_0 = 10000$  $y_0 \geq 0 \wedge x_0 \neq 10000$

Article development led by ACM Queue
queue.acm.org

SAGE has had a remarkable impact at Microsoft.

BY PATRICE GODEFROID, MICHAEL Y. LEVIN, AND DAVID MOLNAR

SAGE: Whitebox Fuzzing for Security Testing



and its users millions of dollars. If monthly security update costs you \$0.001 (one tenth of one cent) in just electricity or loss of productivity, then this number multiplied by one billion people is \$1 million. Of course, if malware were spreading on your machine, possibly leaking some of your private data, then that might cost you millions of dollars.

your operating system) has read the image data, decoded it, created new data structures with the decoded data, and passed those to the graphics card in your computer. If the code implementing that jpg parser contains a bug such as a buffer overflow that can be triggered by a corrupted jpg image, then the execution of this jpg parser on your computer could potentially

Scalable Automated Guided Execution

- Primer herramienta en realizar **concolic execution** en binario x86
 - “*what you fuzz is what you ship,*” ya que los compiladores pueden realizar transformaciones que afectan la seguridad.
- Security bugs: los programadores pueden fallar en alojar memoria o manipular buffers apropiadamente, resultando en **vulnerabilidades**.



SAGE



Microsoft

- "*Since 2008, SAGE has been running 24/7 on approximately 100-plus machines/cores automatically fuzzing hundreds of applications in Microsoft security testing labs*"
- "*...has saved Microsoft millions of dollars as well as saved world time and energy, by avoiding expensive security patches to more than one billion PCs.*"
- "*The software running on your PC has been affected by SAGE.*"

Project Springfield

Fuzz your code before hackers do

[Sign up](#)

What is Project Springfield?

Project Springfield is Microsoft's unique fuzz testing service for finding security critical bugs in software. Project Springfield helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.



"Million Dollar" Bugs

Project Springfield uses "Whitebox Fuzzing" technology which discovered 1/3rd of the



Battle tested tech

The same state-of-the-art tools and practices used inside Microsoft to fuzz Windows and



Fast, Consistent Roll-out

Project Springfield provides the



Available now

Enterprise customers are currently using Project Springfield to find and fix





master branch

[Getting Started](#) [Documentation](#) [Tutorials](#) [Publications](#) [Projects](#) [Getting Involved](#) [Releases](#)

KLEE LLVM Execution Engine

KLEE is a symbolic virtual machine built on top of the [LLVM](#) compiler infrastructure, and available under the UIUC open source license. For more information on what KLEE is and what it can do, see the [OSDI 2008](#) paper.

[Documentation](#)

Learn how to use KLEE



[Use KLEE Docker image](#)

[Building KLEE \(LLVM 3.4\)](#)

[Tutorials](#)

Try KLEE for Yourself

http://klee.doc.ic.ac.uk/

Tutorials ▾

KLEE

JUANPABLOGALEOTTI  LOGOUT

CURRENT FILE

get_sign.c

get_sign.c

regexp.c

maze.c

SYM. ARGS □ ▾

SYM. FILES □ ▾

SYM. INPUT □ ▾

OPTIONS □ ▾

COVERAGE □

▶ RUN KLEE

```
1  /*
2   * First KLEE tutorial: testing a small
3   * function
4   * http://klee.github.io/tutorials/testing-
5   * function/
6   */
7
8  int get_sign(int x) {
9      if (x == 0)
10         return 0;
11
12     if (x < 0)
13         return -1;
14     else
15         return 1;
16
17 int main() {
18     int a;
19     klee_make_symbolic(&a, sizeof(a), "a");
20     return get_sign(a);
21 }
```

KLEE RESULTS

OUTPUT STATS

Job queued!

Executing KLEE

Executing KLEE

Executing KLEE

Executing KLEE

Done!

Ran command "klee /tmp/code/code.o".

KLEE: output directory is "/tmp/code/klee-out-0"

KLEE: Using STP solver backend

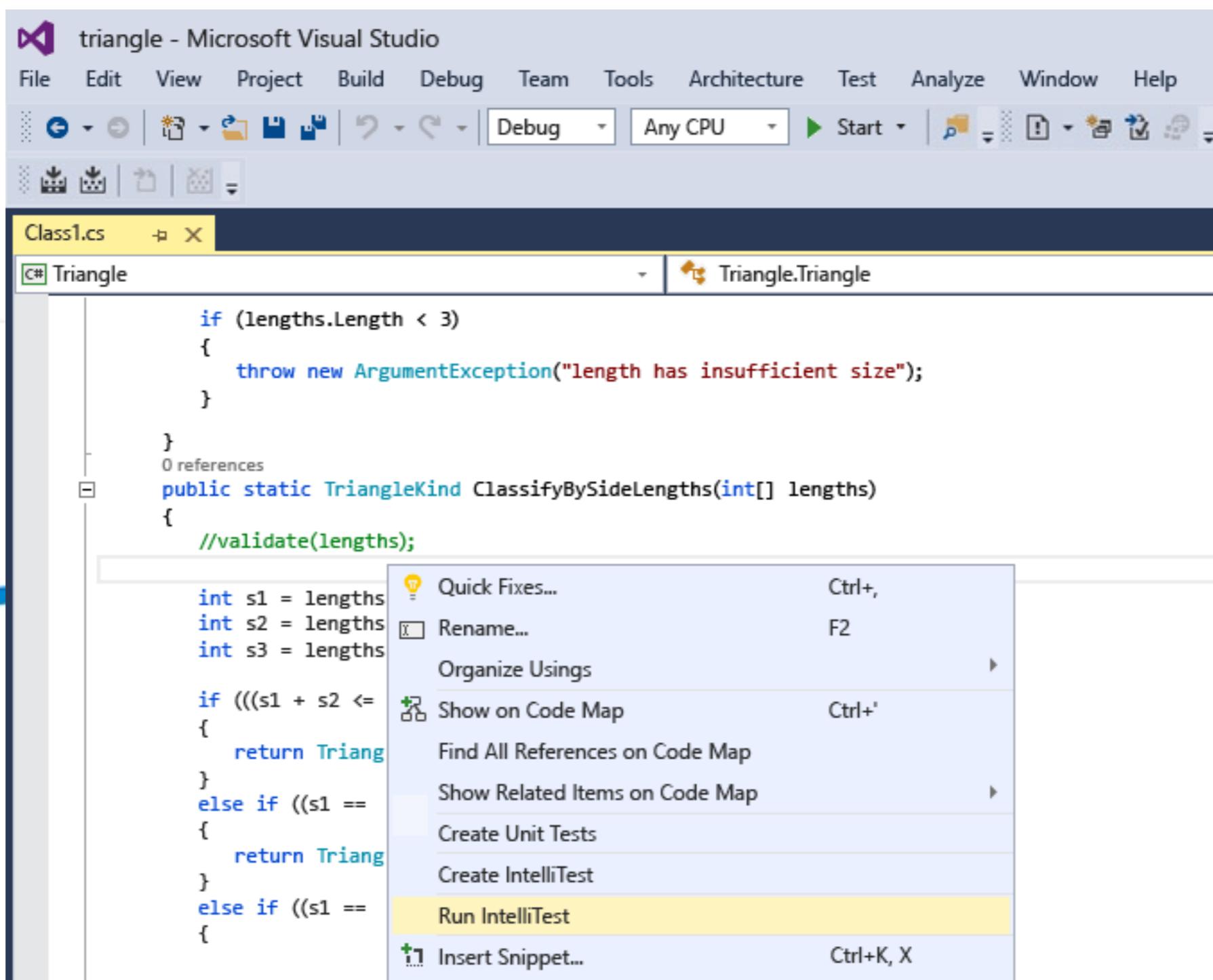
KLEE: done: total instructions = 30

KLEE: done: completed paths = 3

KLEE: done: generated tests = 3

Microsoft Visual Studio 2015

IntelliTest



Microsoft Visual Studio 2015

IntelliTest

IntelliTest Exploration Results - stopped

Triangle.ClassifyBySideLengths(int[] lengths) | Run | Save | Open | 0 Warnings

8 ✓ 4 ✗ 16/16 blocks, 0/0 asserts, 12 runs

	lengths	result	Summary/Exception	Error Message
✗ 1	null		NullReferenceException	Object refer...
✗ 2	{}		IndexOutOfRangeException	Index was out...
✗ 3	{0}		IndexOutOfRangeException	Index was out...
✗ 4	{0, 0}		IndexOutOfRangeException	Index was out...
✓ 5	{0, 0, 0}	Invalid		
✓ 6	{5, 538, 0}	Invalid		
✓ 7	{67, 0, 0}	Invalid		
✓ 8	{422, 536, 6...}	Scalene		
✓ 9	{528, 413, 5...}	Isosceles		
✓ 10	{2, 2, 3}	Isosceles		
✓ 11	{1, 512, 512}	Isosceles		
✓ 12	{512, 512, 5...}	Equilateral		

► Details:

► Stack trace:

System.NullReferenceException...
at Triangle.ClassifyBySideLengths...
at TriangleTest.ClassifyBySideLengths...

<http://www.codehunt.com>

CODE HUNT

INSTRUCTIONS



PLAY



LEADERBOARD



SELECT SECTOR

CHANGE ZONE



HUNTER
STATS

101/15 SECTORS
UNLOCKED

100/15 SECTORS
COMPLETED

Discover the arithmetic operation applied to 'x'.

CAPTURE CODE

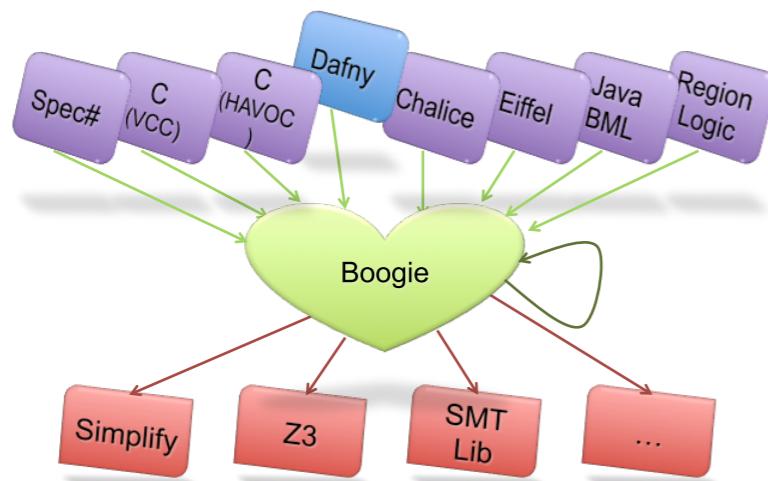
RESET LEVEL

Java

```
1 public class Program {  
2     public static int Puzzle(int x) {  
3         return 0;  
4     }  
5 }  
6 }
```

	X	EXPECTED RESULT	YOUR RESULT	DESCRIPTION
		0	1	Mismatch
		-1	0	
@	Well done so far. Look at numbers on line 4 to capture the code.			

Boogie como IL



<https://rise4fun.com/Dafny/>

The screenshot shows the Microsoft Research Dafny program verifier interface. At the top, it asks "Is this program correct?". Below is a code snippet:

```

1 method M(n: int)
2 {
3     // count up to 'n'; will this program terminate?
4     var i := 0;
5     while i < n
6     {
7         i := i + 1;
8     }
9 }

```

At the bottom, it displays the message: "Dafny program verifier finished with 1 verified, 0 errors".

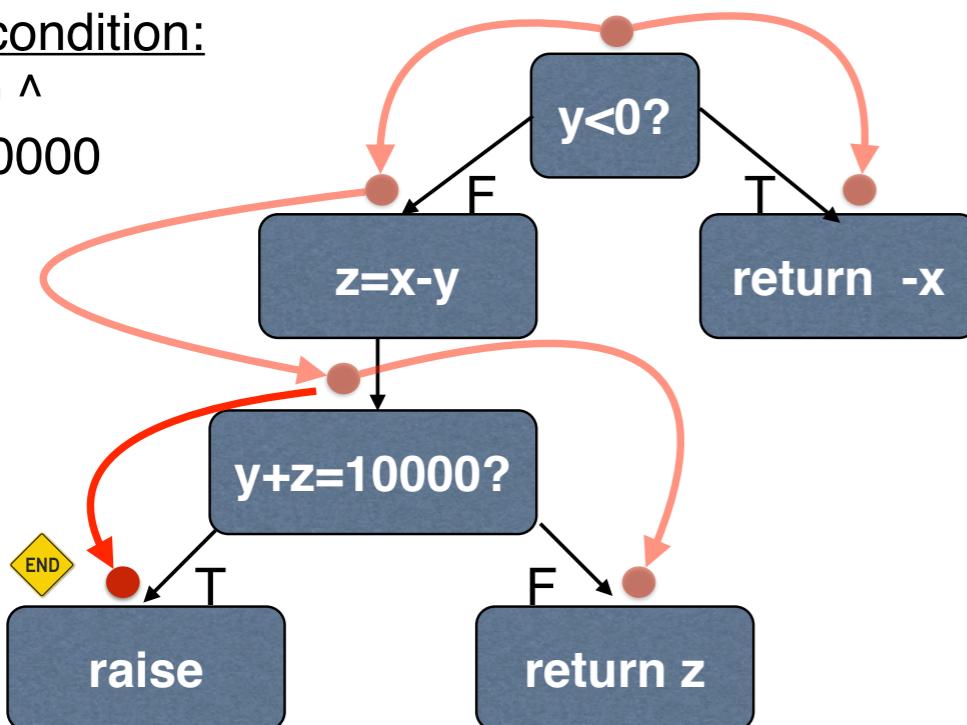
symbolic state:

$$x=x_0, y=y_0, z=x_0-y_0$$

path condition:

$$y_0 >= 0 \wedge$$

$$x_0 = 10000$$



Microsoft Visual Studio 2015
IntelliTest

