# 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?

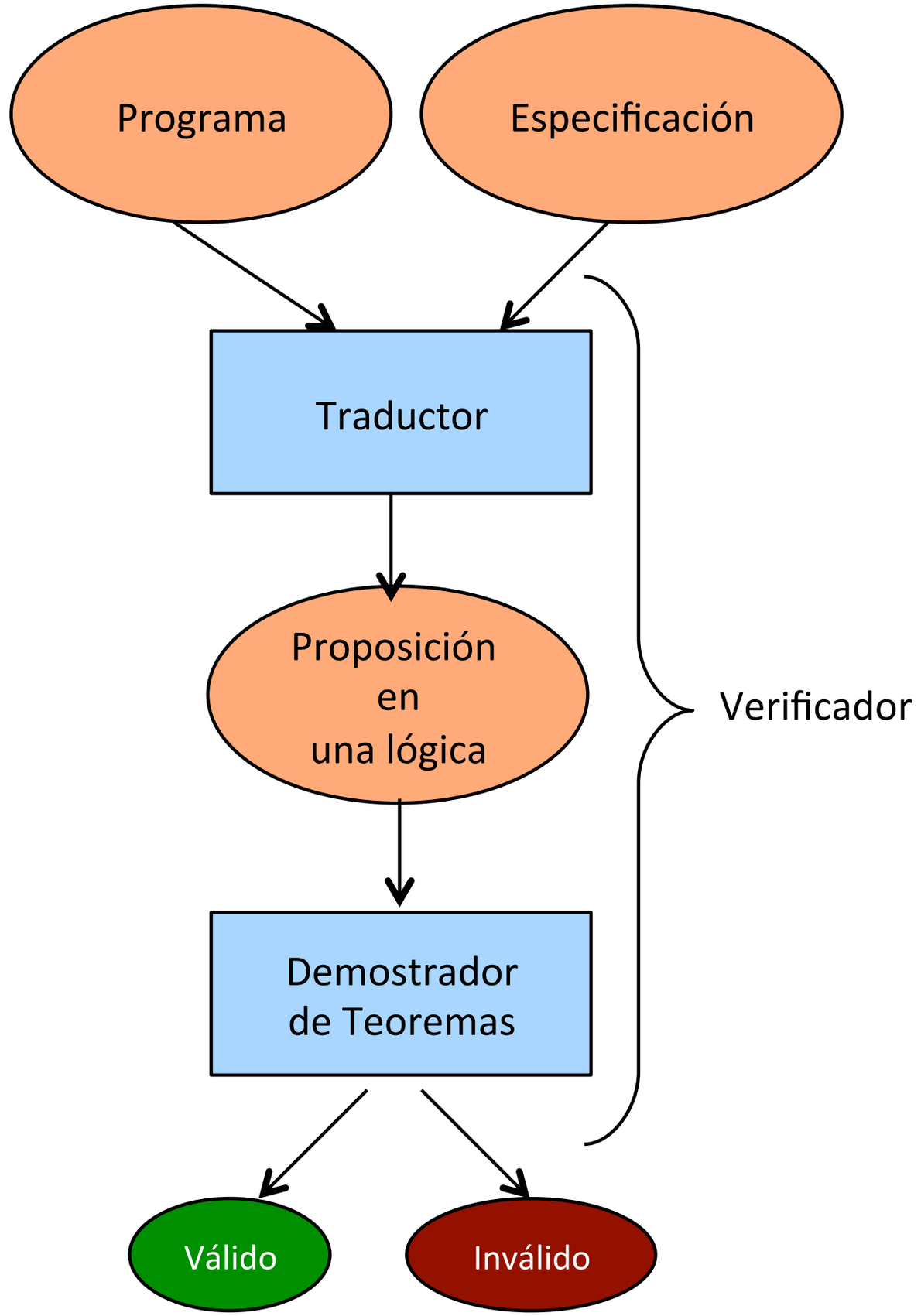
# Verificación de Programas

- 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 => 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 **indecidible**.
- Sin embargo, en la práctica se pueden demostrar algunos teoremas (y si tenemos la demostración, el teorema vale)

# Verificación Automática de Programas



# Dafny

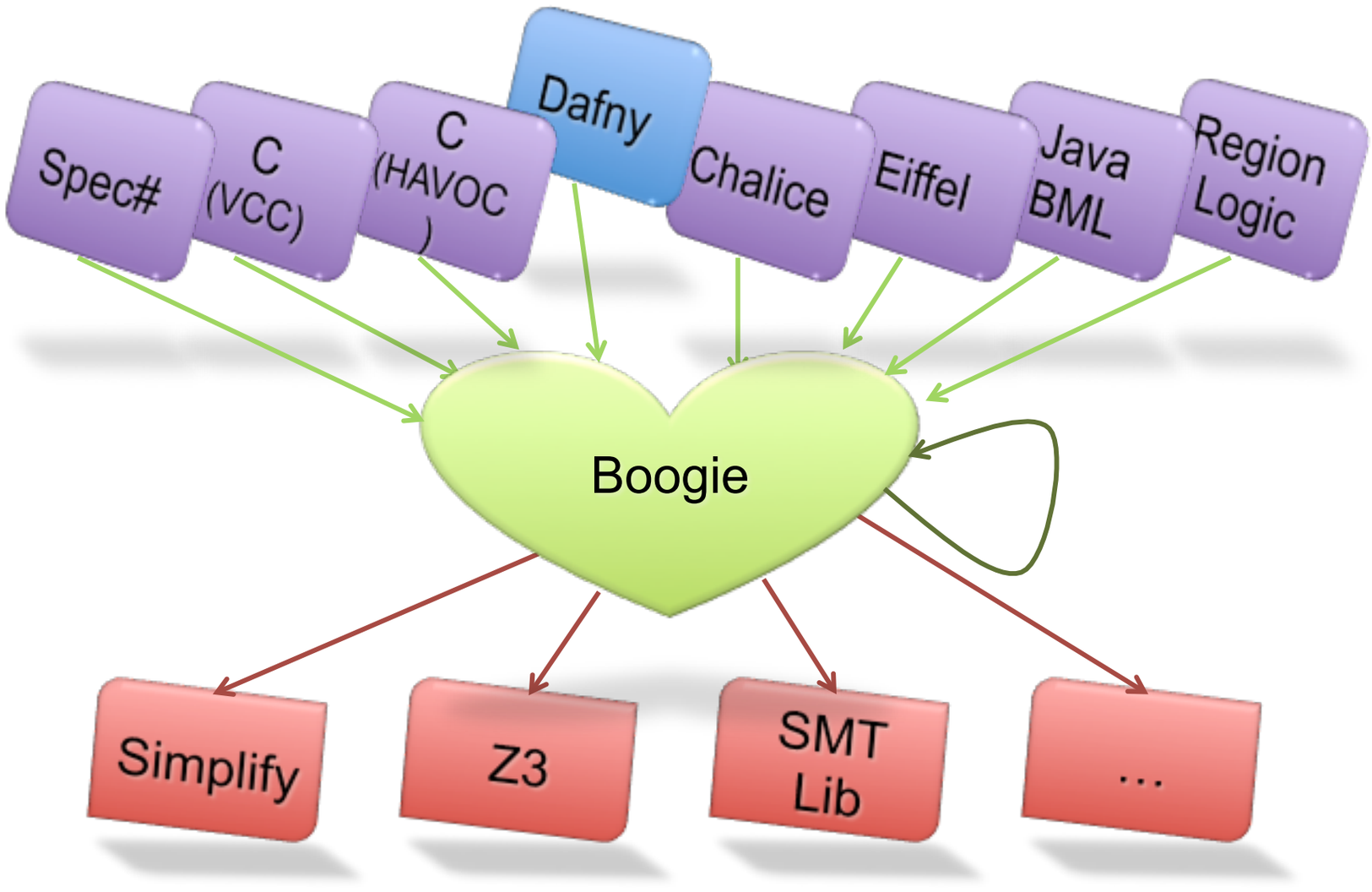
**research.microsoft.com/dafny**

- 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;
}
```

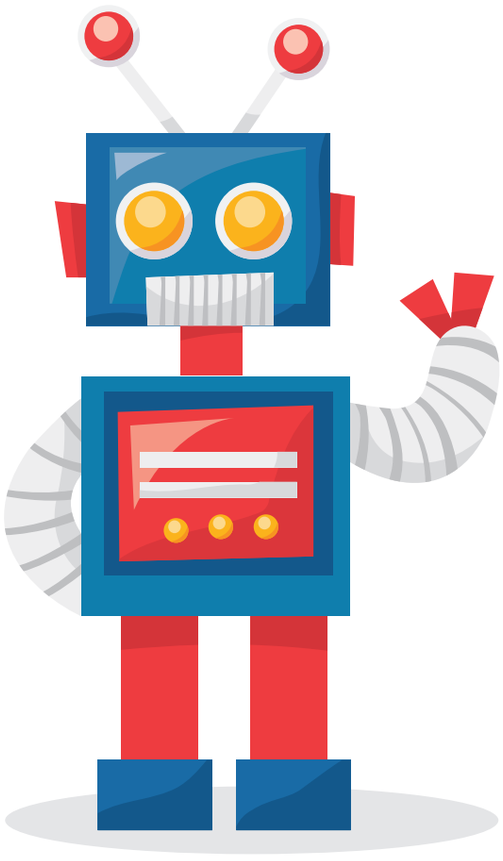
```
method BusquedaBinaria(a: array<int>, x: int) returns (ret: bool)

    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

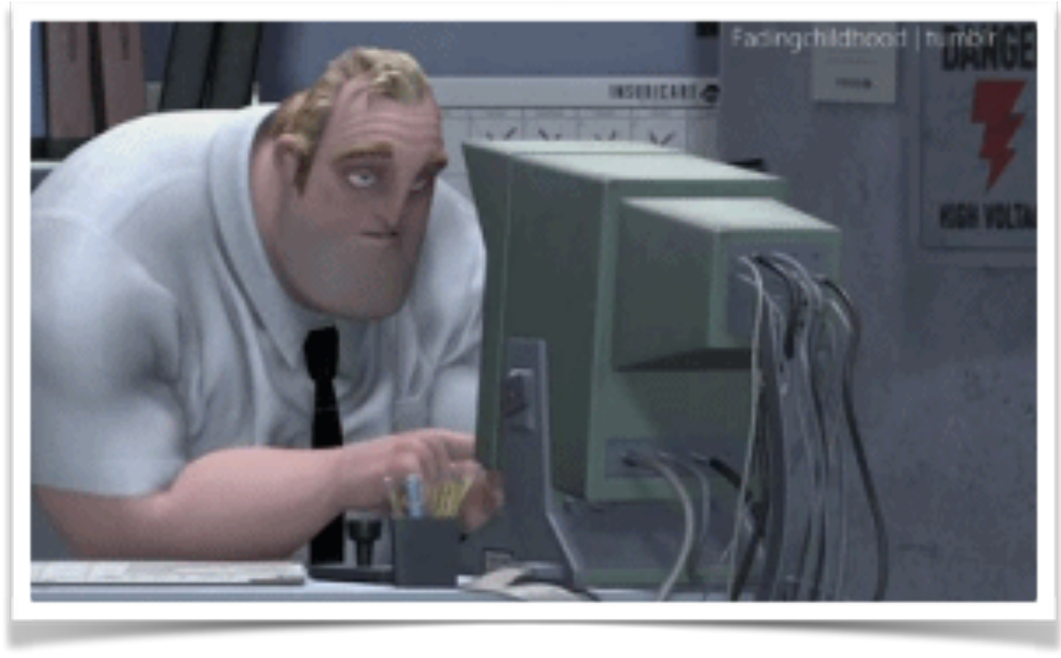
# Testing



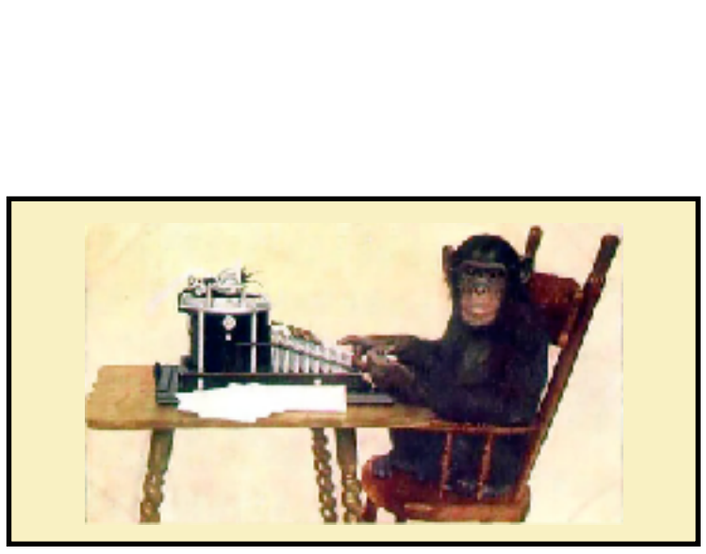
## Creación Manual

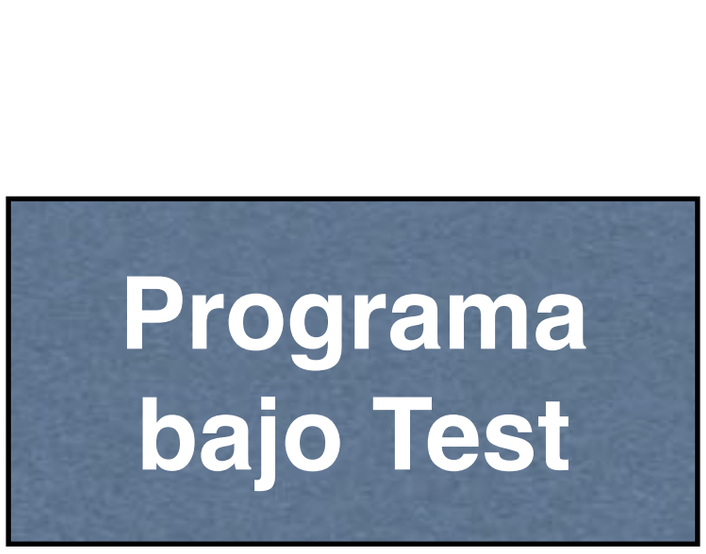
VS

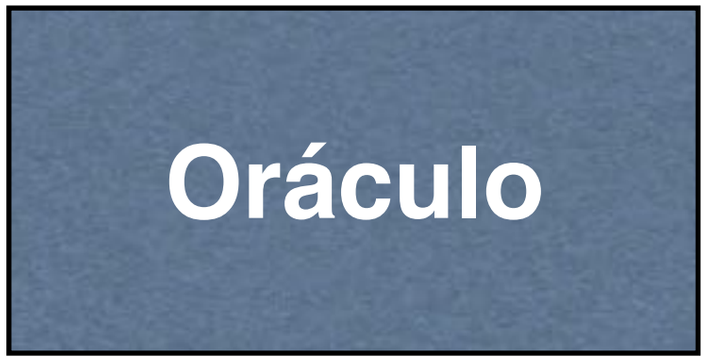
## Ejecución Automática



# Random Testing

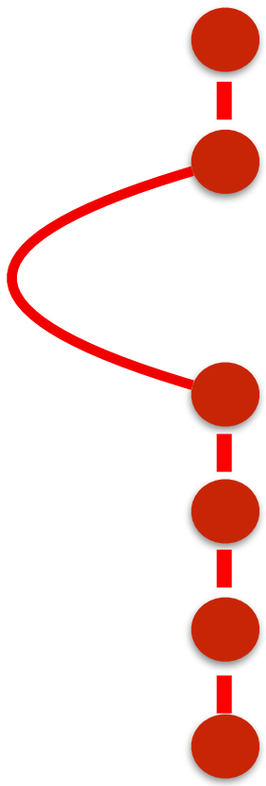





```
1. def testme(x, y):
2.   if (y<0):
3.       return -x
4.   else:
5.       z = x - y
6.       if (y+z=10000):
7.         raise Exception(“error”)
8.       else:
9.         return z
```

```
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 ~ 0%

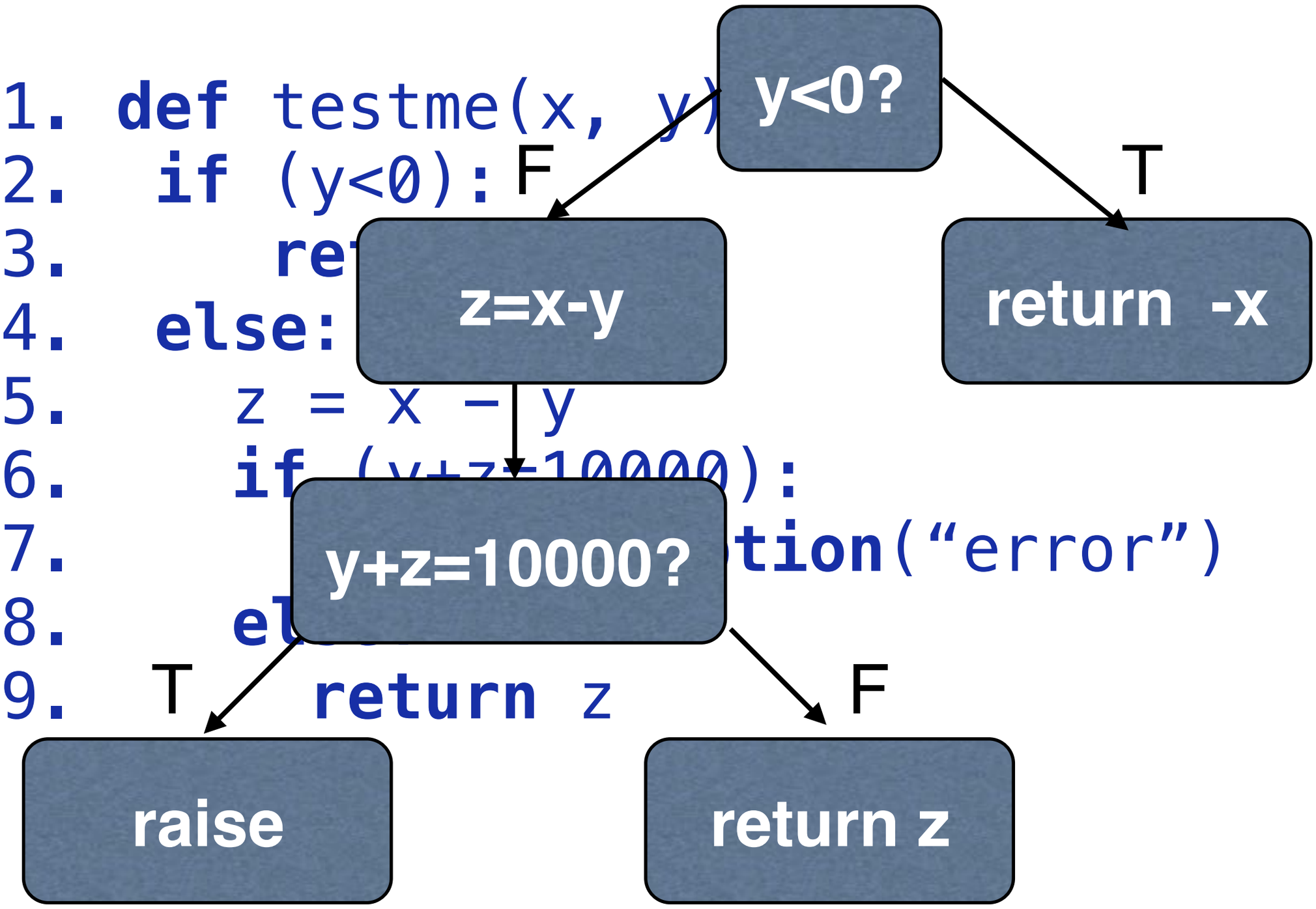
# Ejecución Simbólica
## Al Rescate!

*Necesitamos…*

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

# Ejecución Simbólica

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

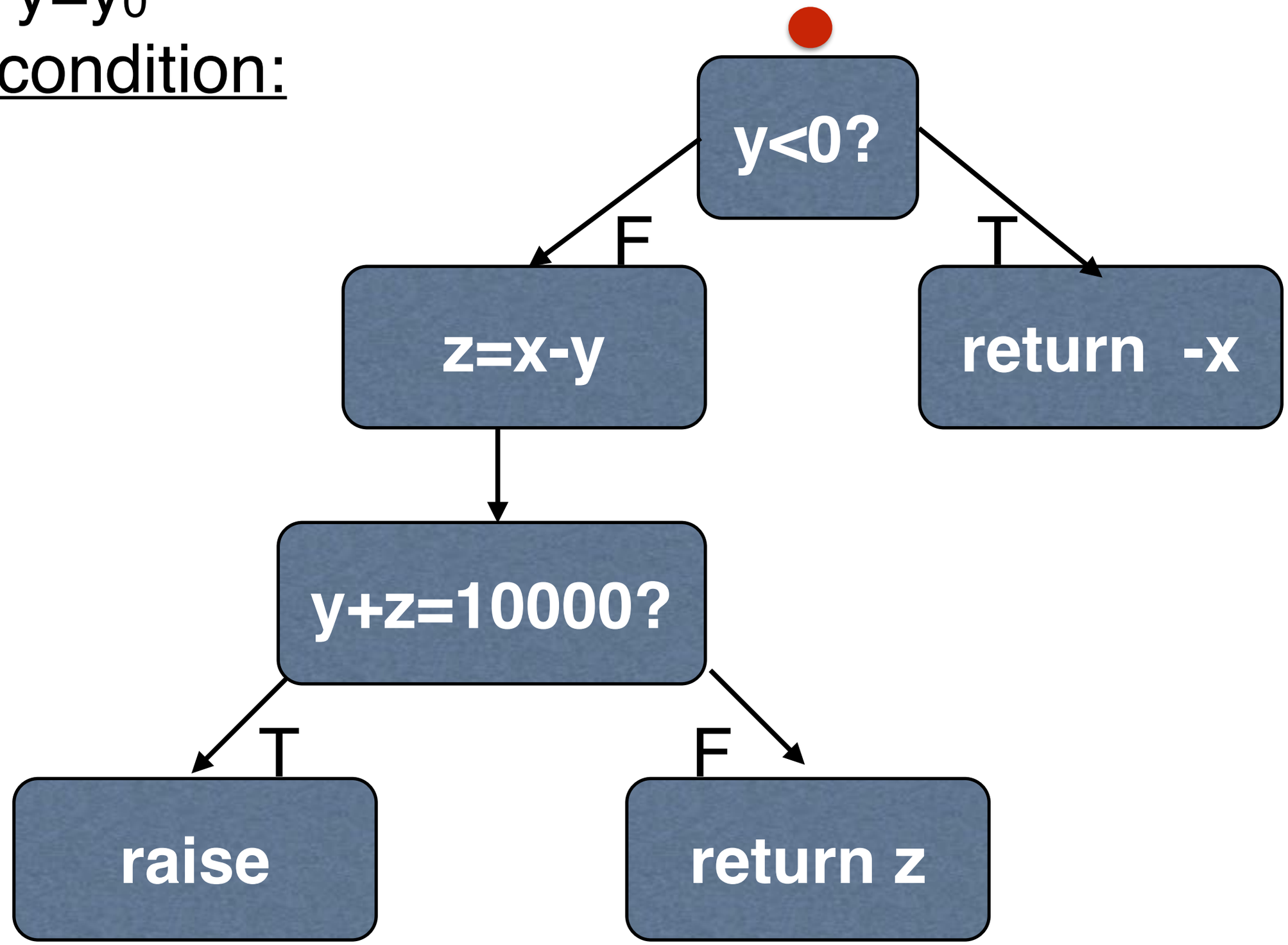


<u>symbolic state:</u>

$x=x_0$, $y=y_0$

<u>path condition:</u>

true

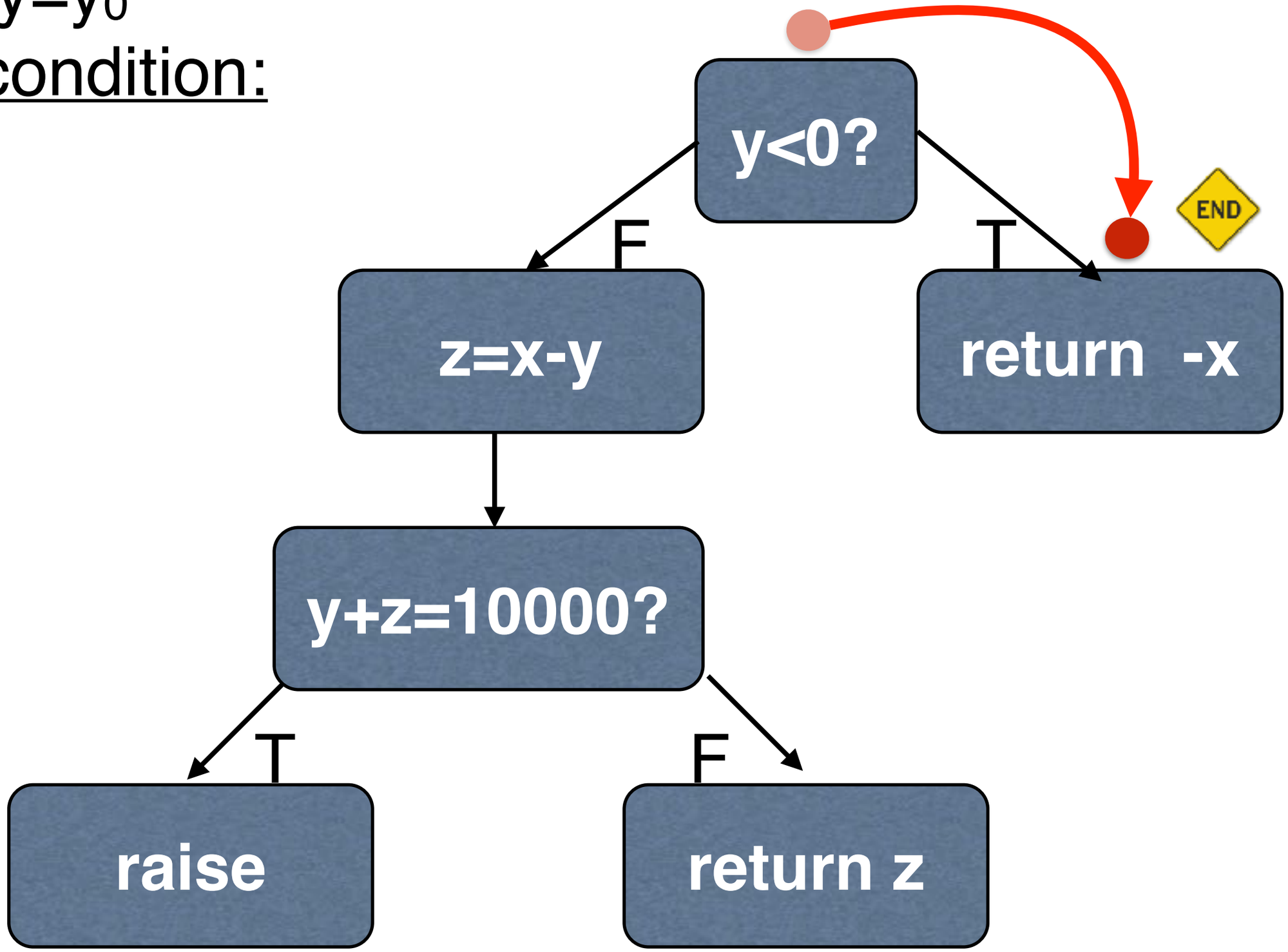


<u>symbolic state:</u>

$x=x_0$, $y=y_0$

<u>path condition:</u>

$y_0<0$

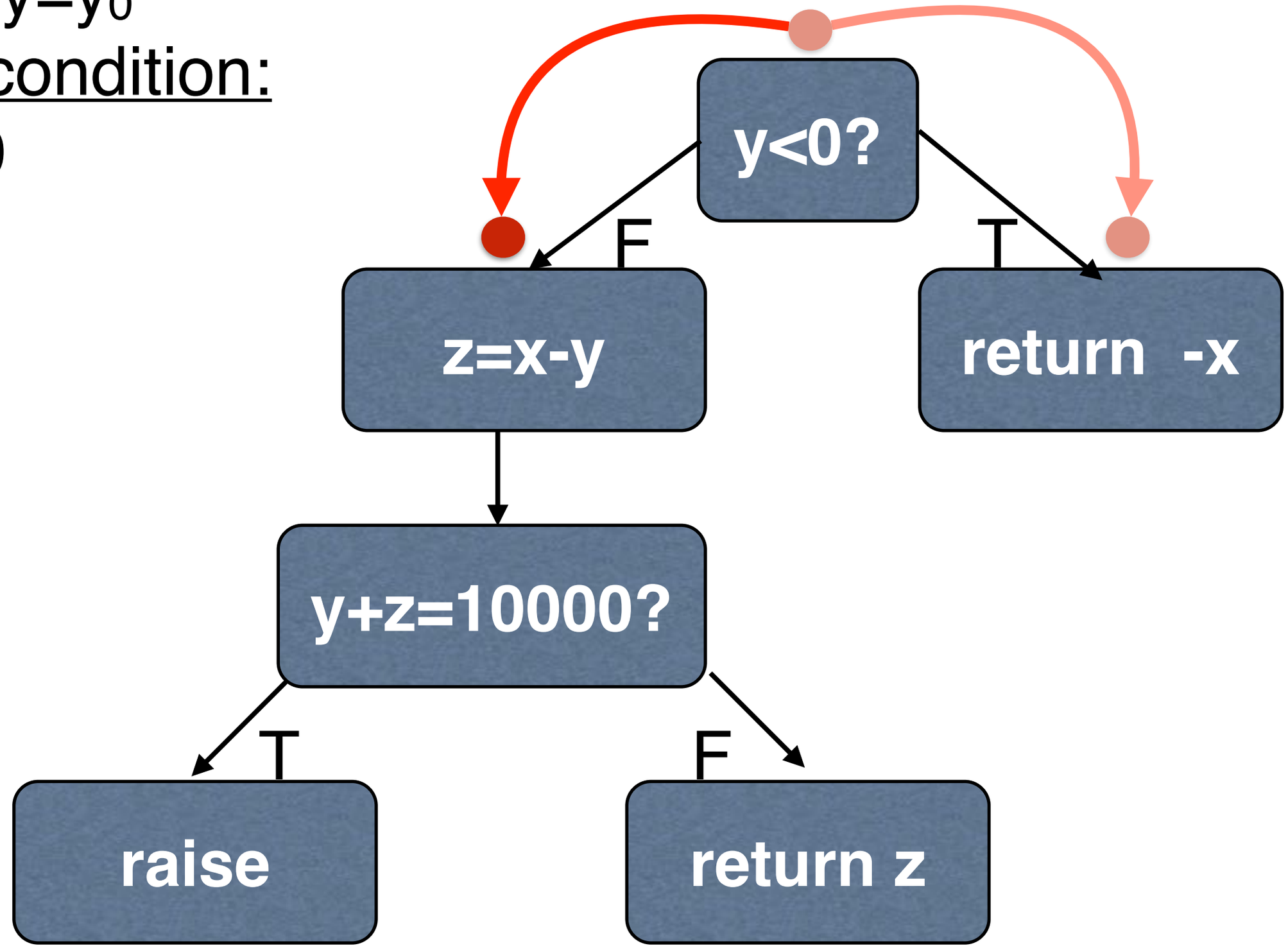


<u>symbolic state:</u>

$x=x_0$, $y=y_0$

<u>path condition:</u>

$y_0>=0$

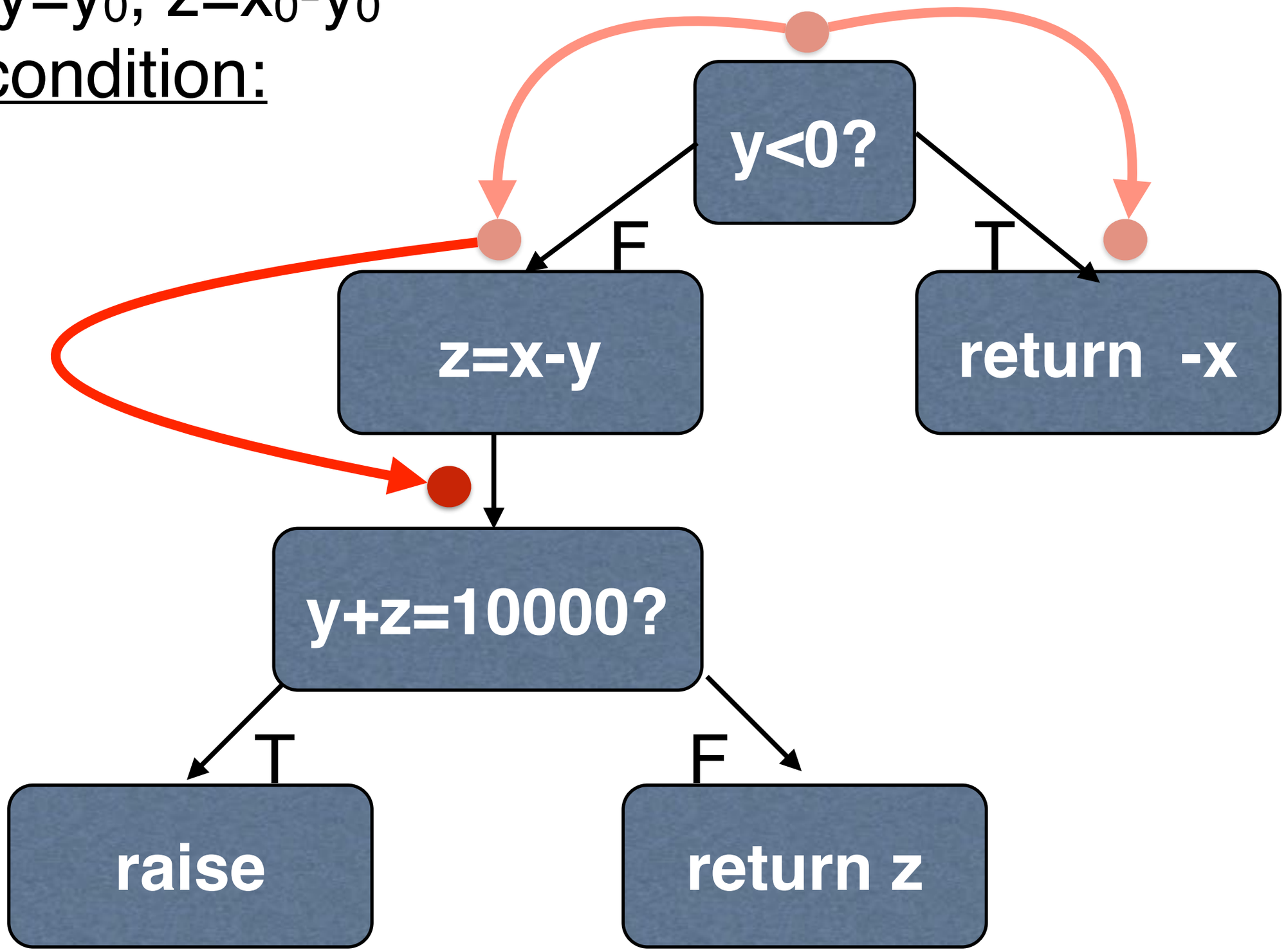


<u>symbolic state:</u>

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

<u>path condition:</u>

$y_0>=0$



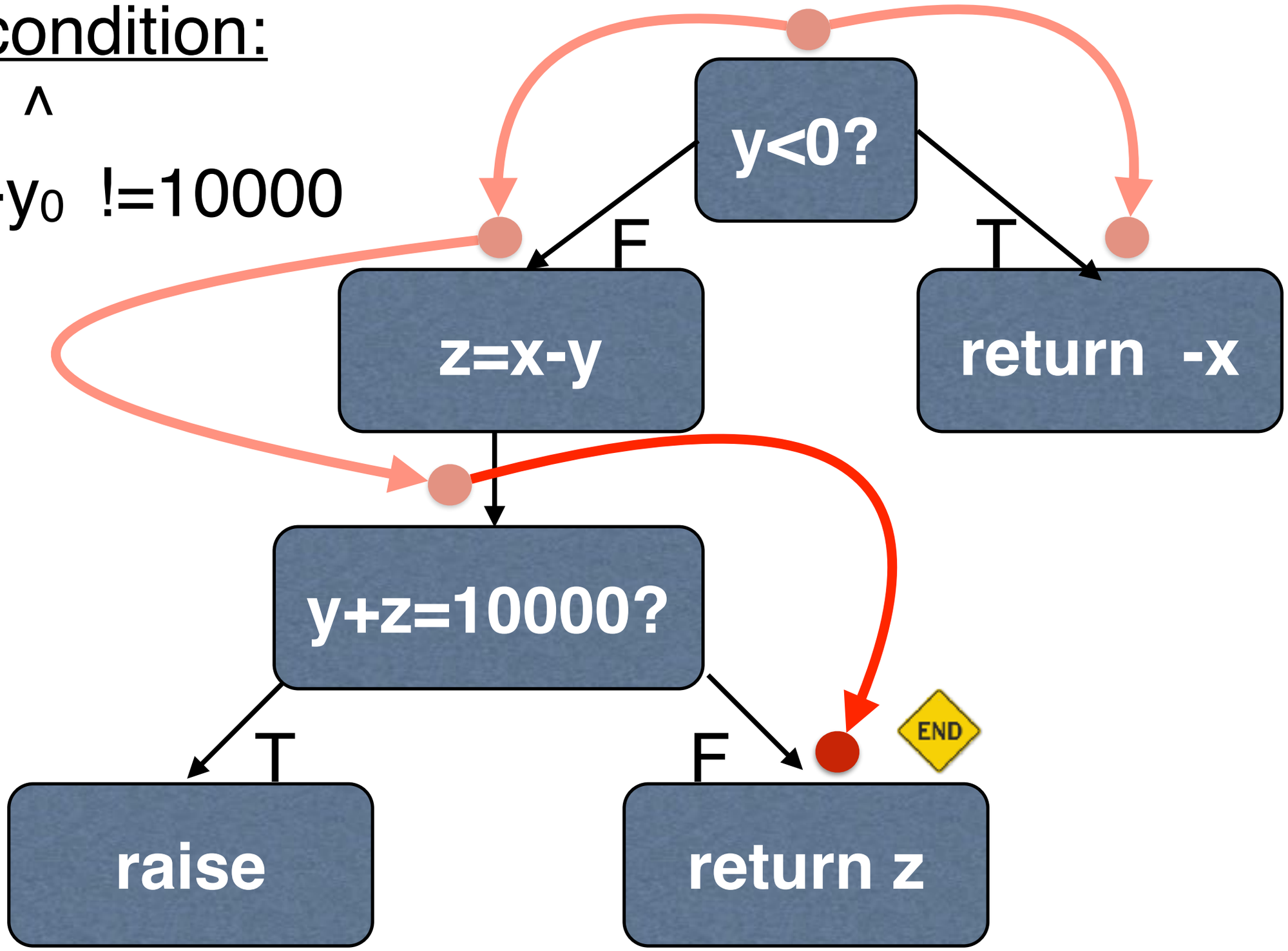
<u>symbolic state:</u>

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

<u>path condition:</u>

$y_0>=0$ ^

$y_0+x_0-y_0$ !=10000



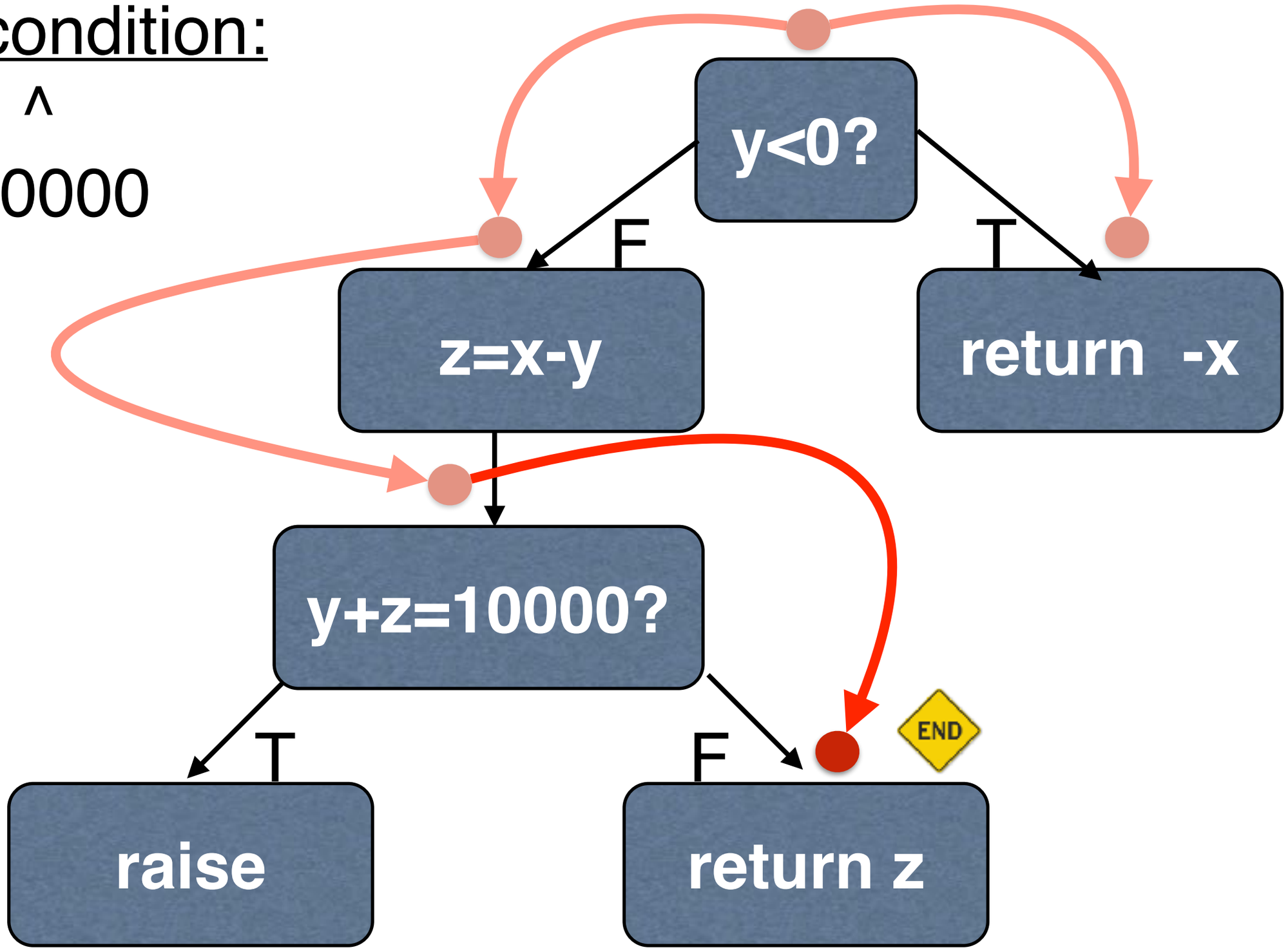
<u>symbolic state:</u>

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

<u>path condition:</u>

$y_0>=0$ ^

$x_0$ !=10000



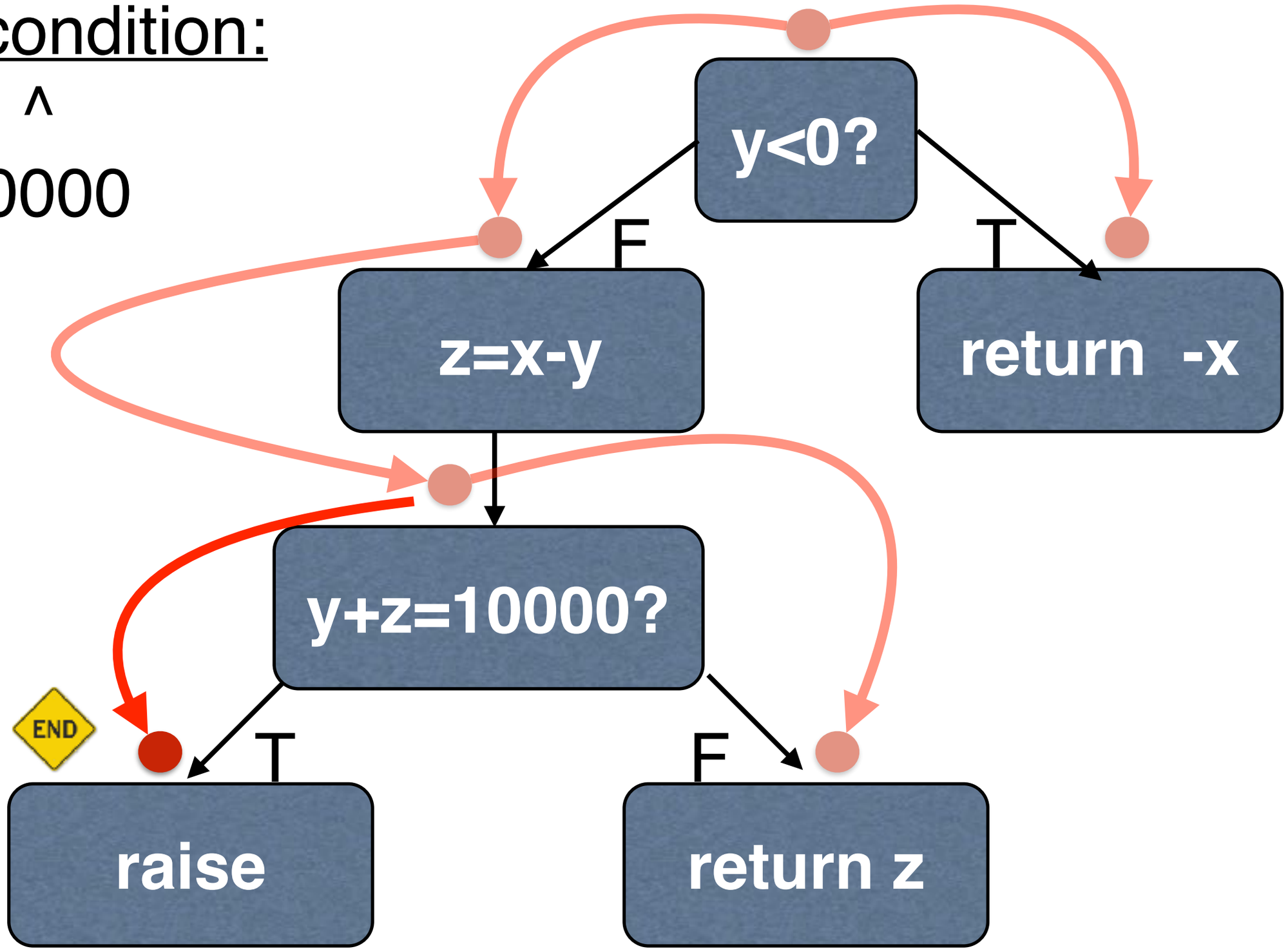
<u>symbolic state:</u>

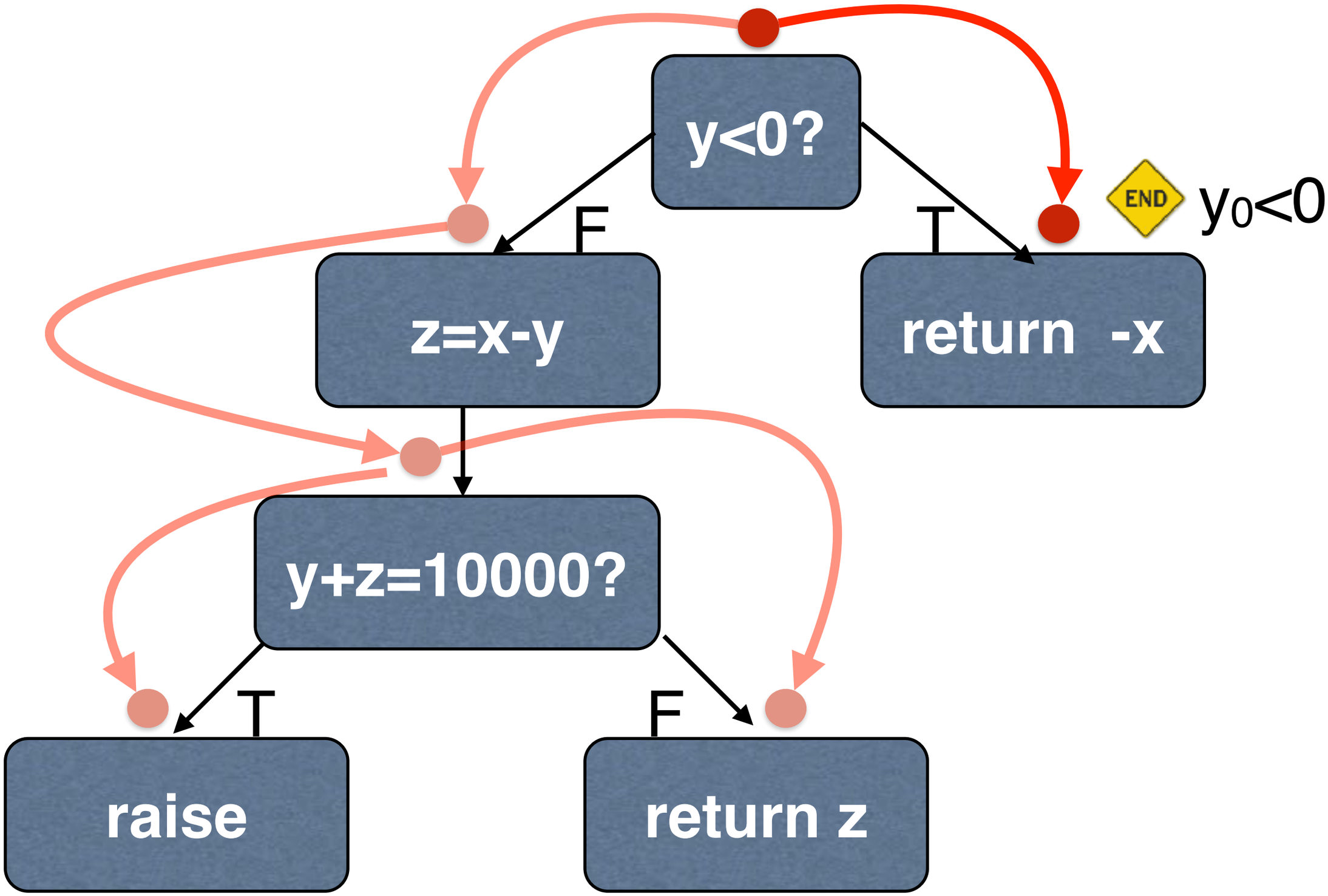
$x=x_0$, $y=y_0$, $z=x_0-y_0$

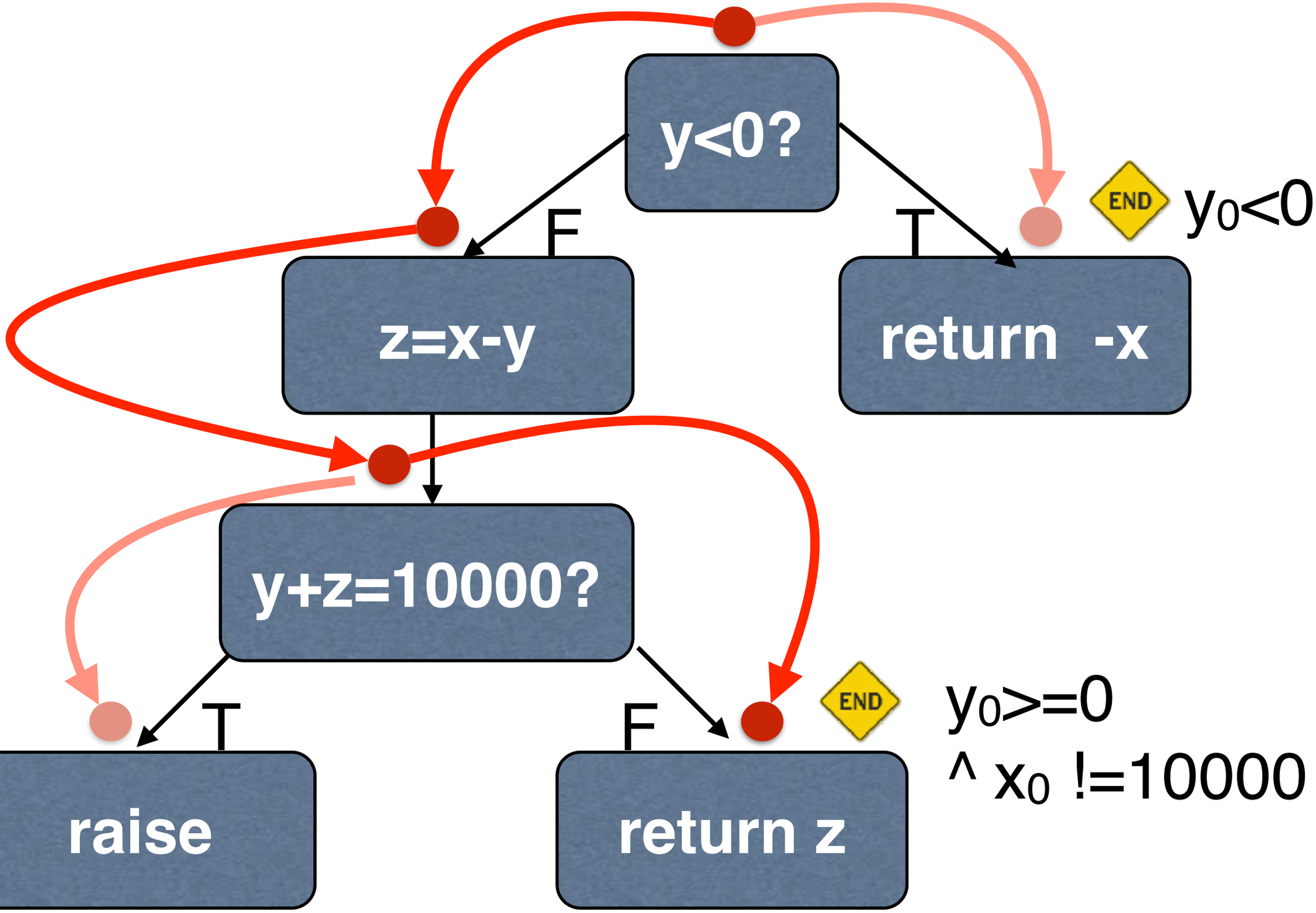
<u>path condition:</u>

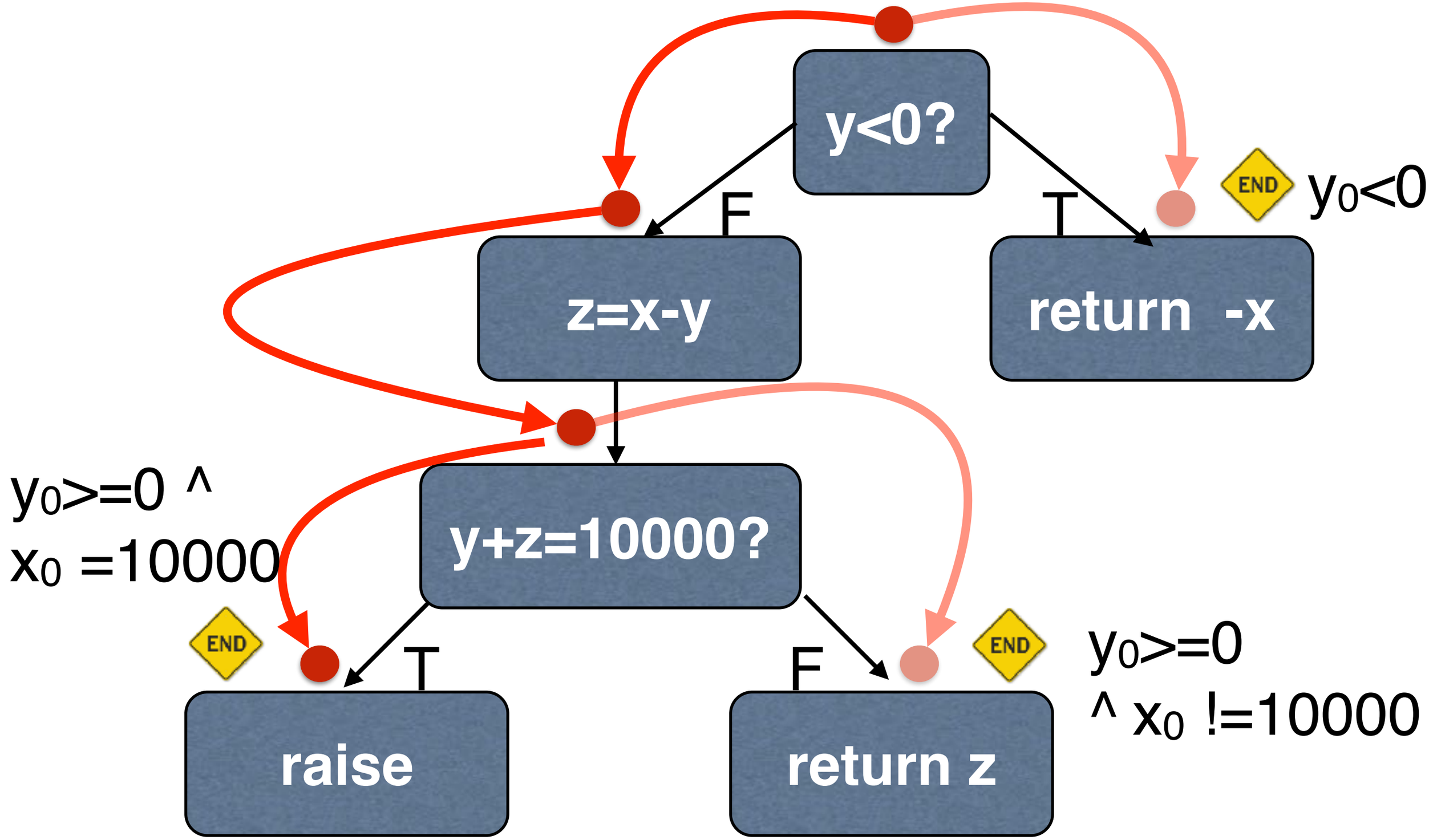
$y_0>=0$ ^

$x_0$ =10000




y<0?
F
T
END
y0<0
z=x-y
return -x
y+z=10000?
T
F
raise
return z

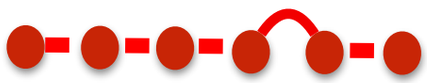

y<0?
F
T
END
$y_0$<0
z=x-y
return -x
y+z=10000?
T
F
END
raise
return z
$y_0$>=0
^ $x_0$ !=10000


y<0?
F
T
END
y0<0
z=x-y
return -x
y0>=0 ^
x0 =10000
y+z=10000?
END
T
F
END
y0>=0
^ x0 !=10000
raise
return z

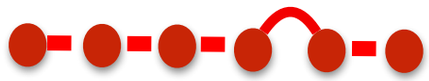
# Constraint Solving


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

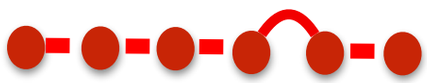
# Constraint Solving



$y_0<0$



$y_0>=0 \wedge x_0 \neq 10000$



$y_0>=0 \wedge x_0 = 10000$


$y_0>=0 \wedge x_0 \neq 10000$

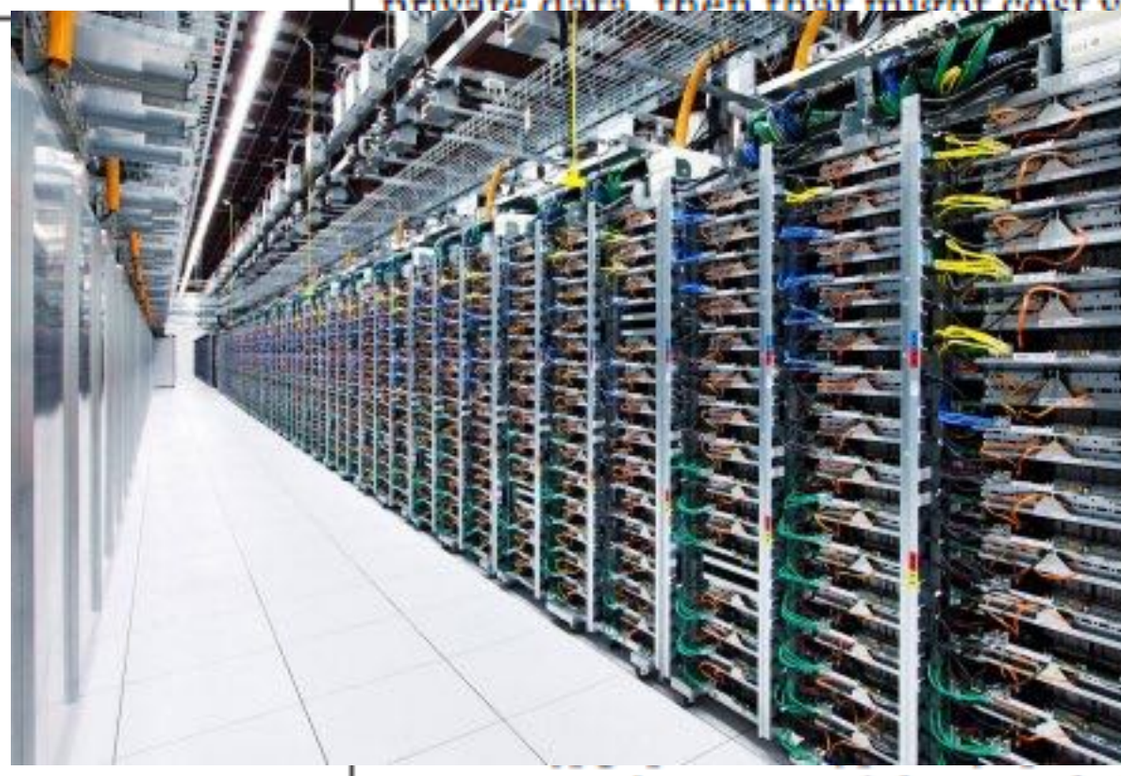
DOI:10.1145/2093548.2093564

Article development led by acmqueue
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

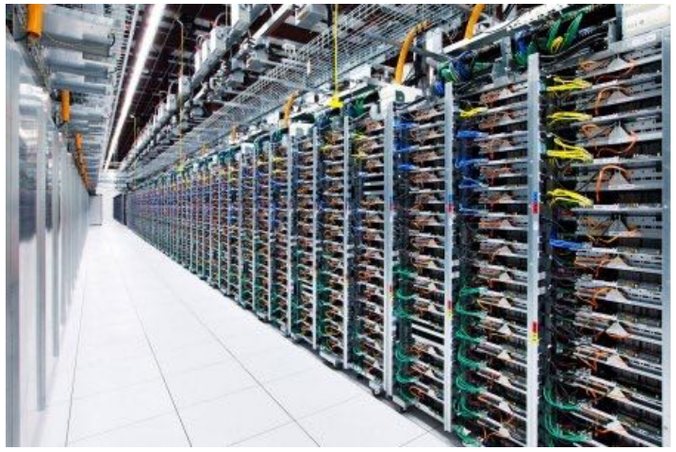


and its users millions of dollars. I
monthly security update costs y
$0.001 (one tenth of one cent) in ju
electricity or loss of productivity, th
this number multiplied by one b
lion people is $1 million. Of course
malware were spreading on your m
chine, possibly leaking some of yo
private data, then that might cost y

your operating system) has read t
image data, decoded it, created n
data structures with the decoded da
and passed those to the graphics ca
in your computer. If the code imp
menting that jpg parser contains
bug such as a buffer overflow that c
be triggered by a corrupted jpg imag
then the execution of this jpg pars
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 alocar 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

### Battle tested tech

The same state-of-the-art tools and practices used inside

### Fast, Consistent Roll-out

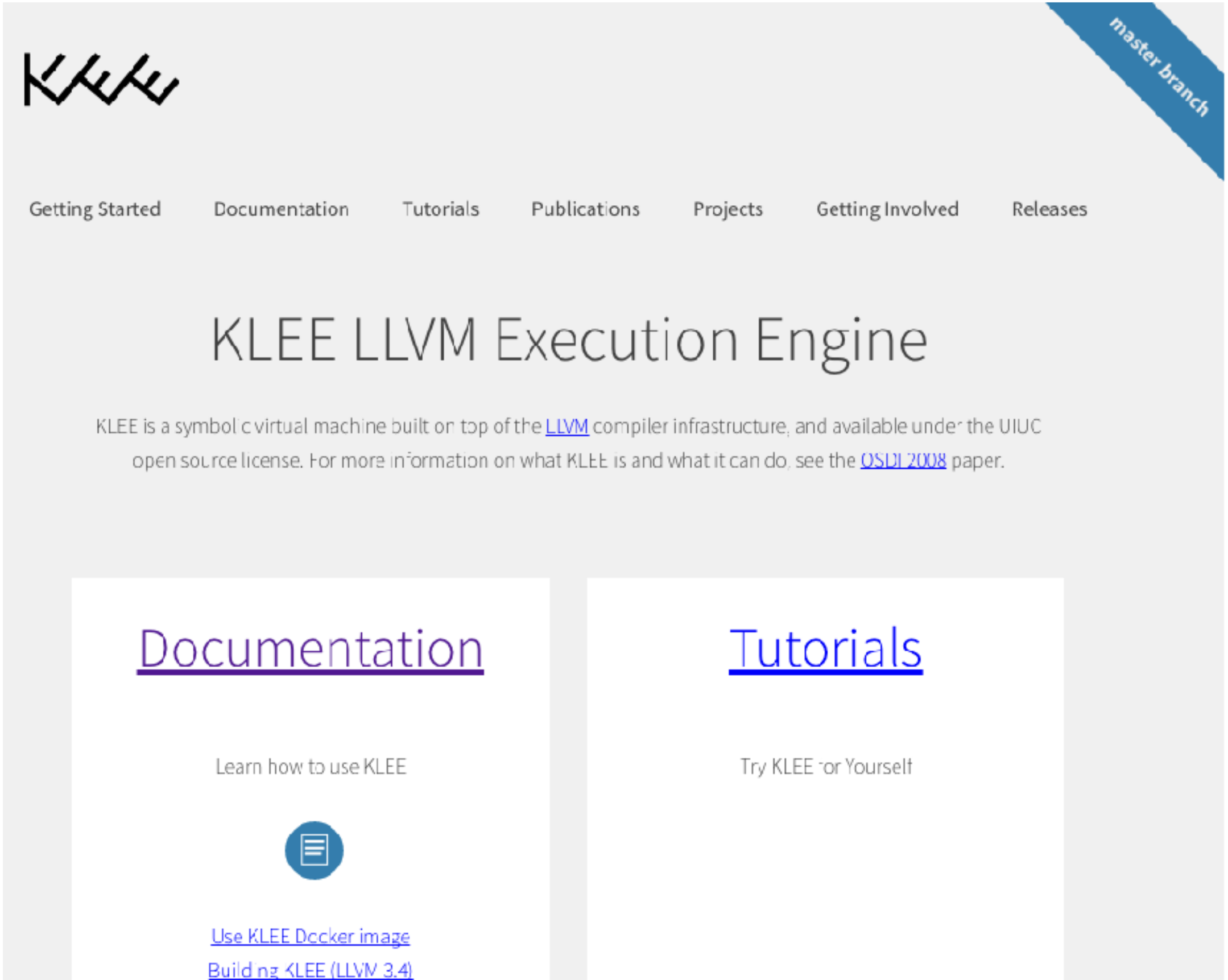
Project Springfield provides the

### Available now

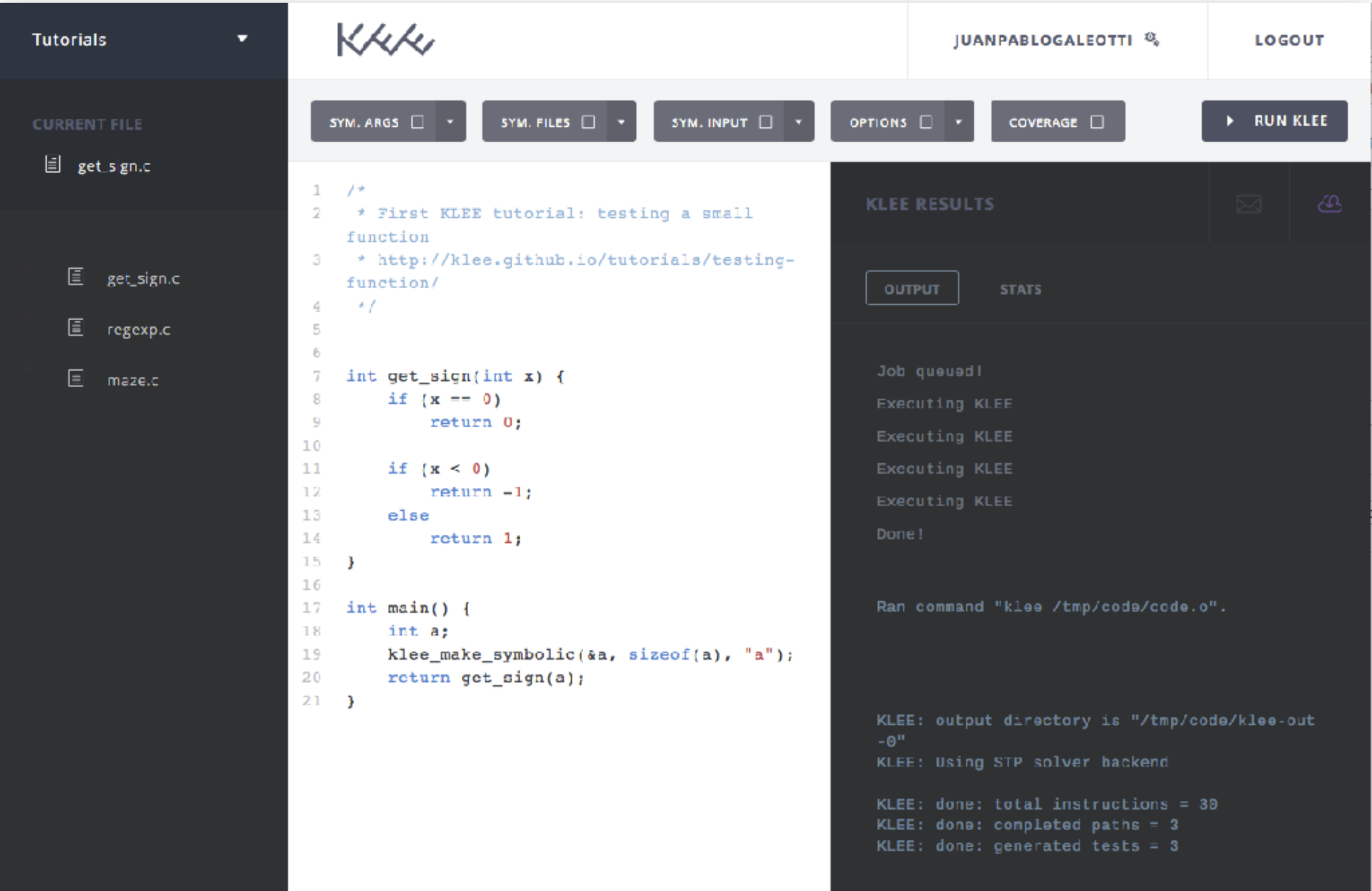
Enterprise customers are currently using Project


Microsoft Azure

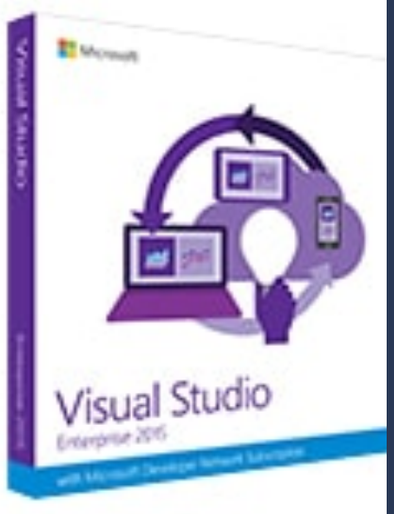
# http://klee.github.io/klee/

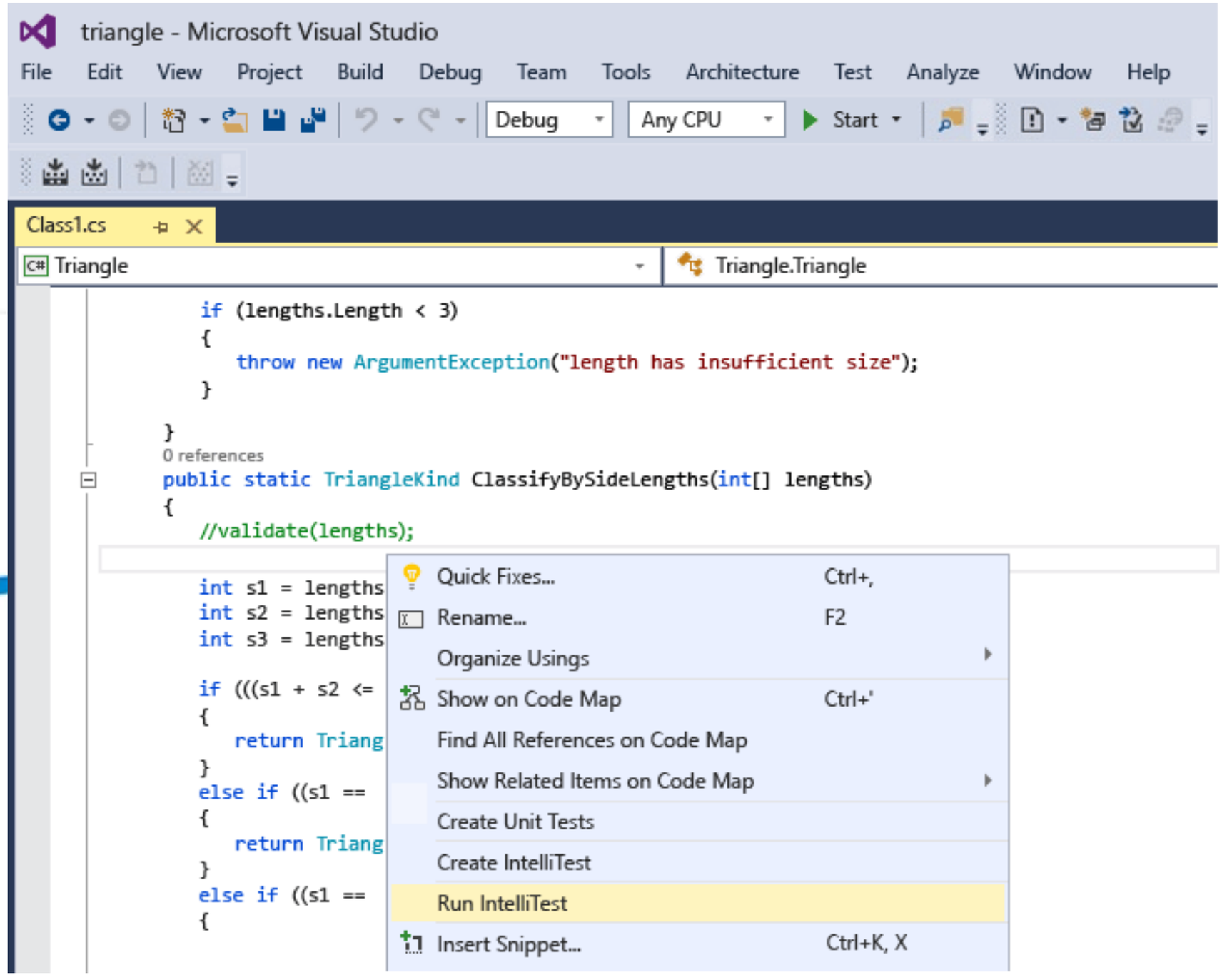


# http://klee.doc.ic.ac.uk/



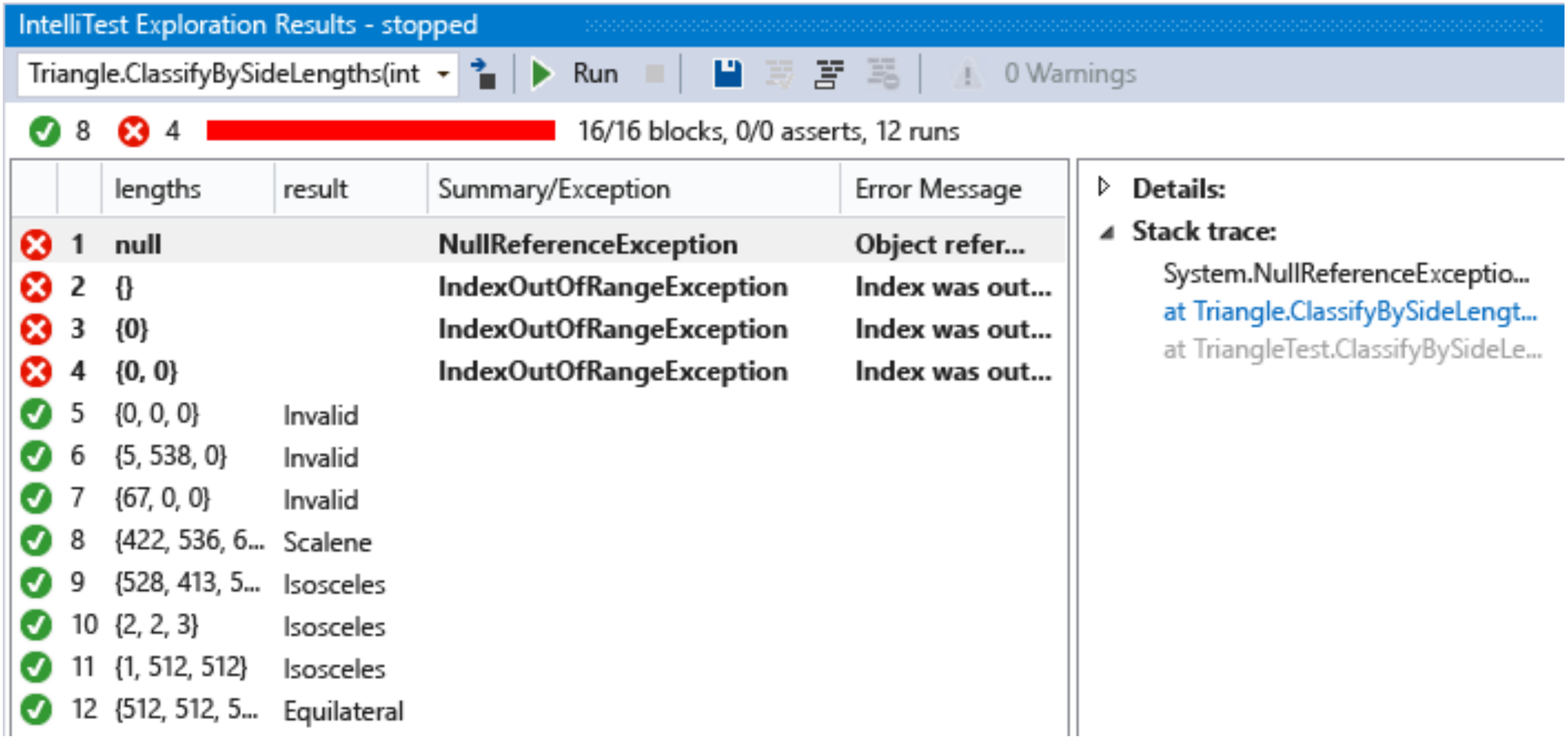
# Microsoft Visual Studio 2015 IntelliTest




https://msdn.microsoft.com/en-us/library/dn823749.aspx

# Microsoft Visual Studio 2015
# IntelliTest




https://msdn.microsoft.com/en-us/library/dn823749.aspx

http://www.codehunt.com

# SELECT SECTOR

CHANGE ZONE

00 TRAINING 00/03

01 ARITHMETIC

02 LOOPS

03 LOOPS 2

04 CONDITIONALS

05 CONDITIONALS 2

06 STRINGS

07 STRINGS 2

08 NESTED LOOPS

09 1D ARRAYS

10 JAGGED ARRAYS

11 ARRAYS 2

12 SEARCH SORT

13 CYPHERS

14 PUZZLES

HUNTER **STATS**

01/15 SECTORS **UNLOCKED**

00/15 SECTORS **COMPLETED**

Discover the arithmetic operation applied to 'x'.

CAPTURE CODE

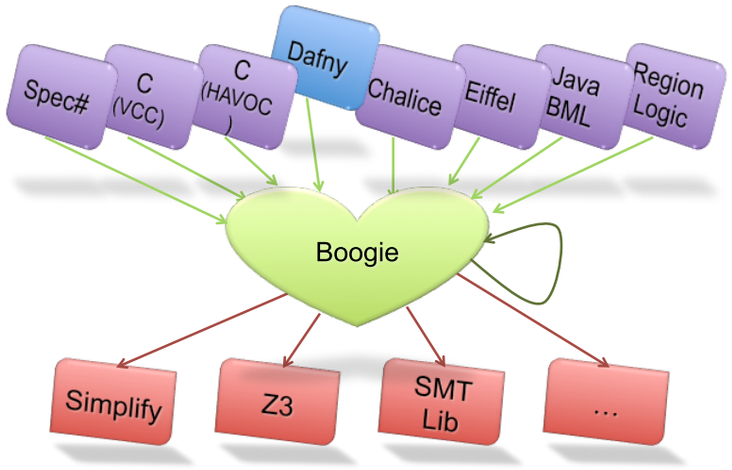
RESET LEVEL

Java

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

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

# 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

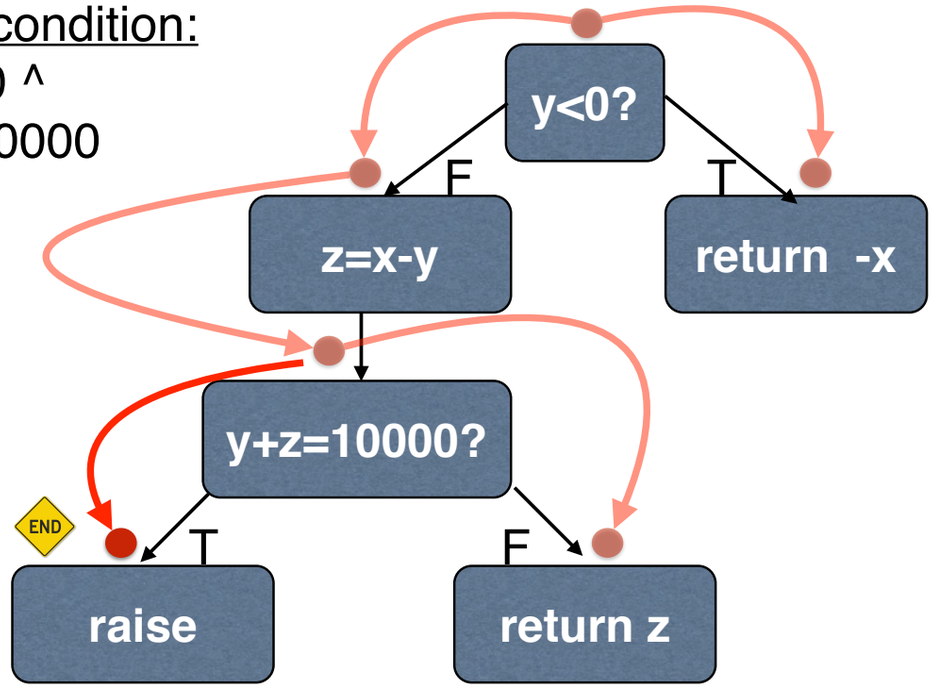
<u>symbolic state:</u>

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

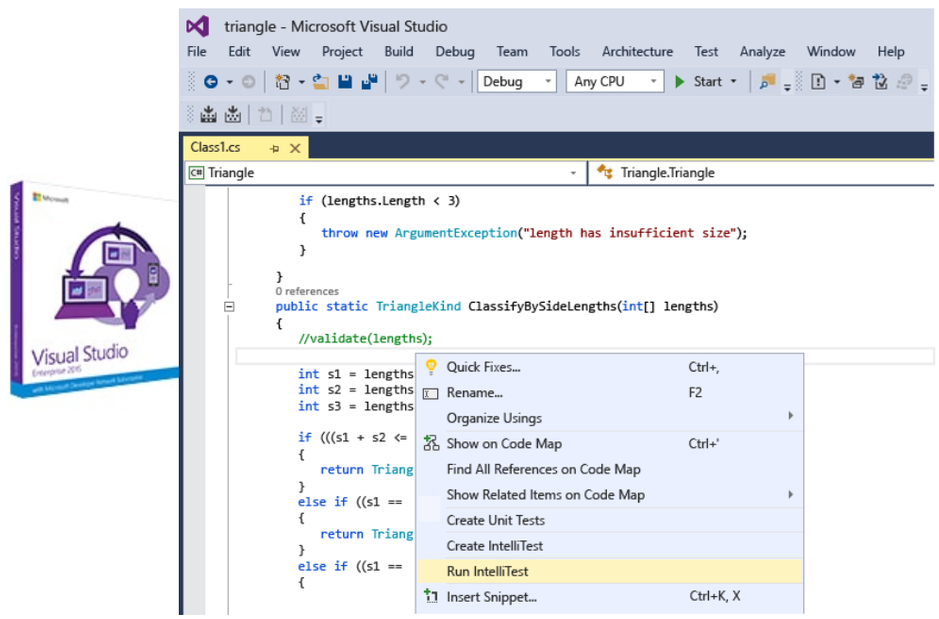
<u>path condition:</u>

$y_0>=0$ ^

$x_0$ =10000




# Microsoft Visual Studio 2015 IntelliTest




https://msdn.microsoft.com/en-us/library/dn823749.aspx