



Control Flow Testing

Universidad Autónoma de Coahuila

Facultad de Sistemas

Carlos Nassif Trejo García

Contexto

Asignación

$X = 2 * y$

Condicionales

If

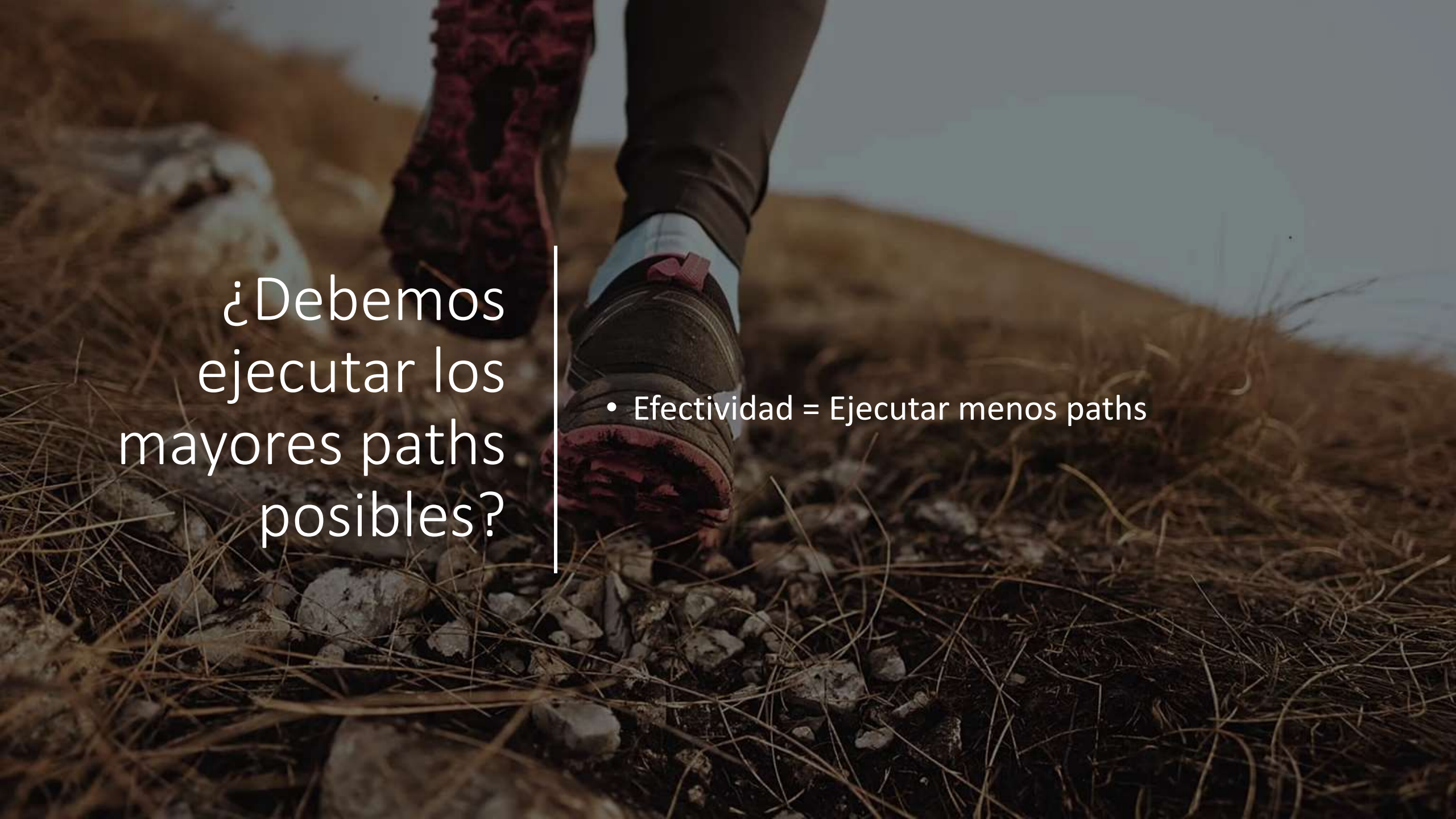
While

Funciones:
Abstracción

For

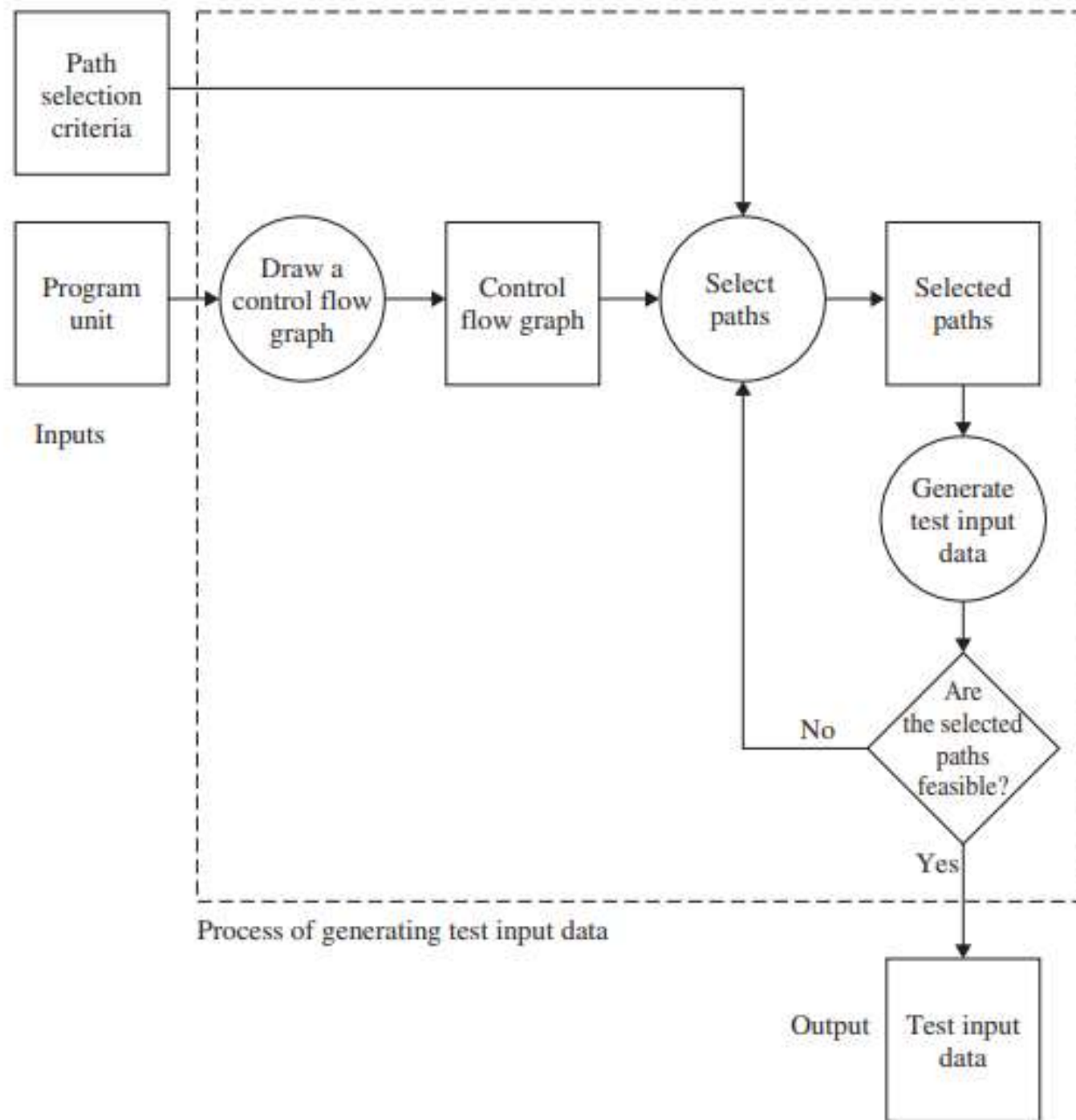
Program path

- Punto de entrada
- Punto de salida
- PP:
 - Ejecución de una secuencia de instrucciones desde la entrada hasta la salida
 - Entrada y salida esperada
 - Unas entradas en específico pueden ejecutar un path en específico



¿Debemos
ejecutar los
mayores paths
posibles?

- Efectividad = Ejecutar menos paths



Criterios

Cada declaración es ejecutado al menos un vez

Para cada condicional, obtener un verdadero y uno falso

Símbolos

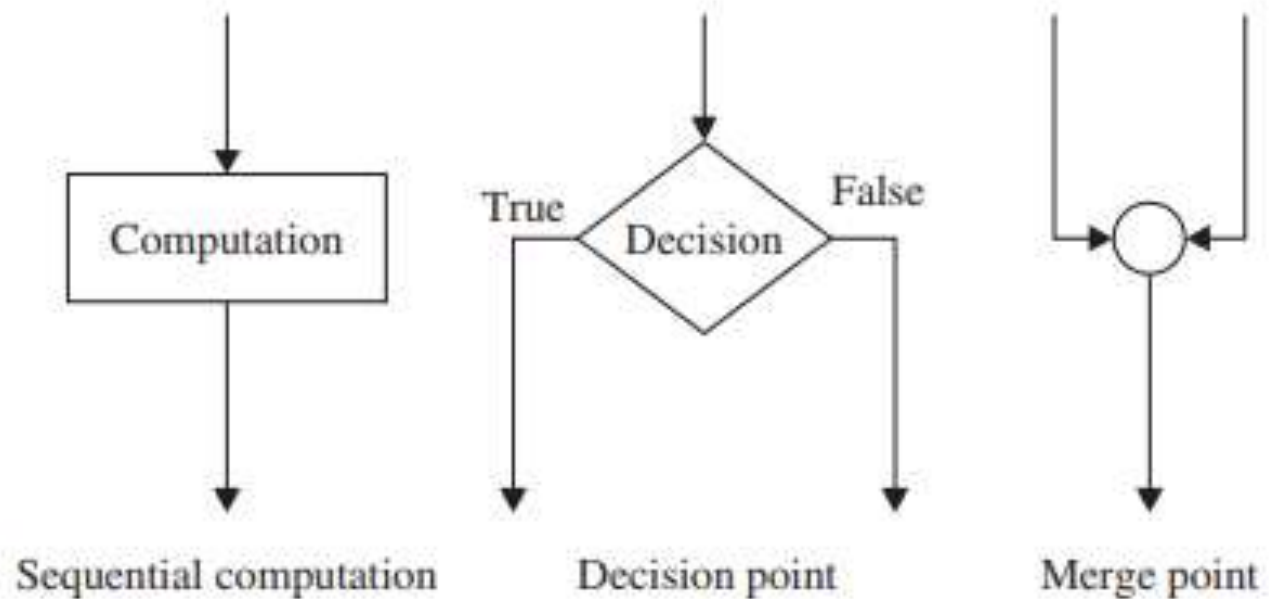


Figure 4.2 Symbols in a CFG.

```

FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */

int openfiles(){
    /*
       This function tries to open files "file1", "file2"
       "file3" for read access, and returns the number o
       successfully opened. The file pointers of the ope
       are put in the global variables.
    */
    int i = 0;
    if(
        ((( fptr1 = fopen("file1", "r")) != NULL) && (
                                                    && (
        ((( fptr2 = fopen("file2", "r")) != NULL) && (
                                                    && (
        ((( fptr3 = fopen("file3", "r")) != NULL) && (
        );
    return(i);
}

```

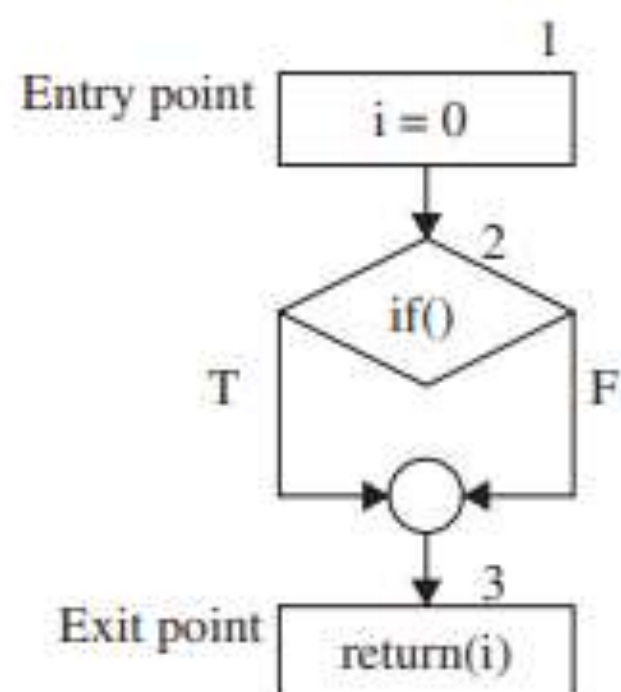
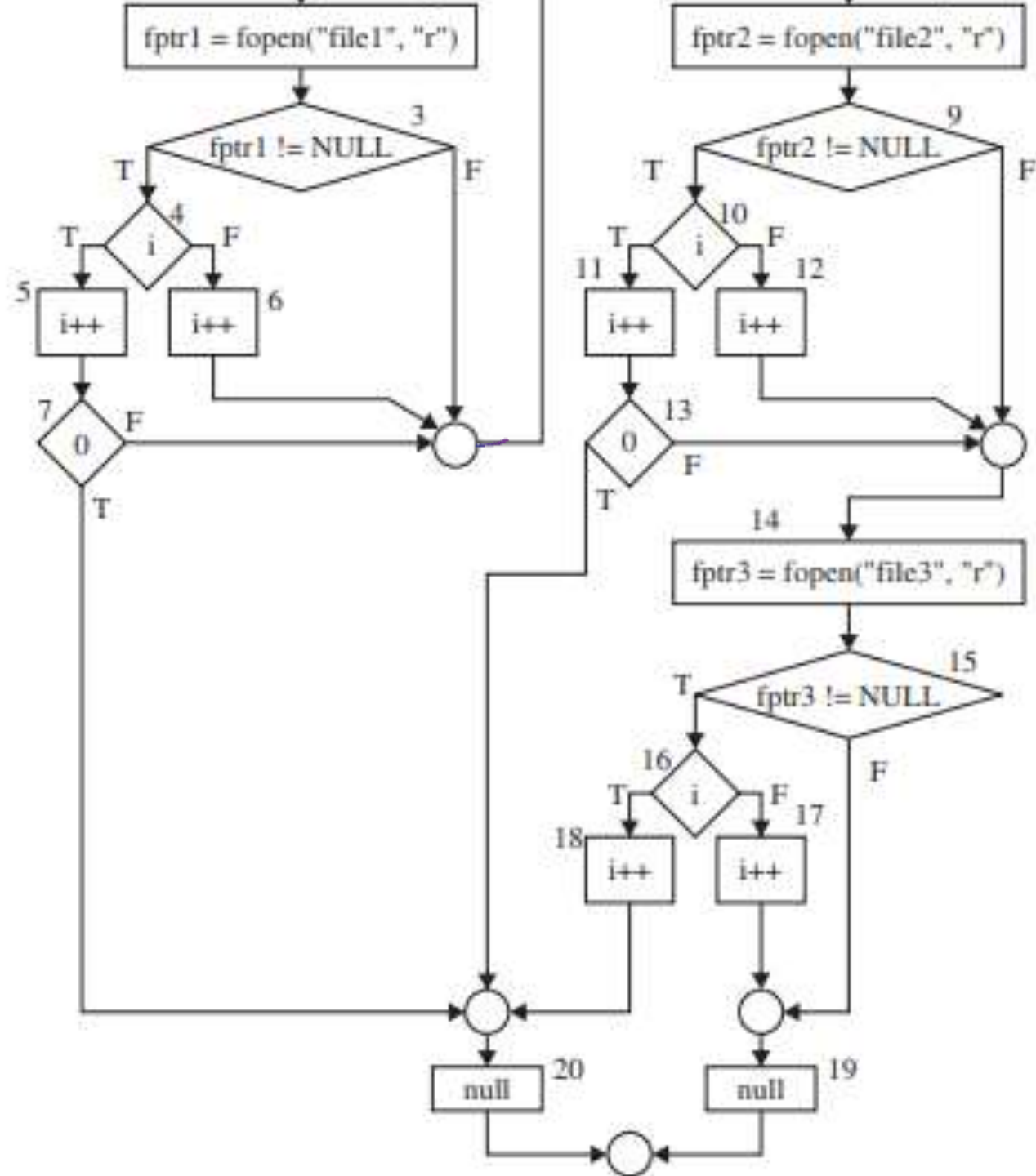


Figure 4.3 Function to open three files.



Selección de paths

- Cada declaración es ejecutado al menos una vez
- No usar entradas que activen el mismo path repetidamente
- Conocer las características de lo que ya fue probado y lo que no

Todos los paths

- Statment Coverage
- Branch Coverage
- Predicate Coverage

Todos los paths

TABLE 4.2 Input Domain of openfiles()

| Existence of file1 | Existence of file2 | Existence of file3 |
|--------------------|--------------------|--------------------|
| No | No | No |
| No | No | Yes |
| No | Yes | No |
| No | Yes | Yes |
| Yes | No | No |
| Yes | No | Yes |
| Yes | Yes | No |
| Yes | Yes | Yes |

TABLE 4.3 Inputs and Paths in openfiles()

| Input | Path |
|-------------------|---|
| < No, No, No > | 1-2-3(F)-8-9(F)-14-15(F)-19-21 |
| < Yes, No, No > | 1-2-3(T)-4(F)-6-8-9(F)-14-15(F)-19-21 |
| < Yes, Yes, Yes > | 1-2-3(T)-4(F)-6-8-9(T)-10(T)-11-13(F)-14-15(T)-16(T)-18-20-21 |

Statement Coverage

Cada declaración del código es ejecutado

Seleccionar paths pequeños

Seleccionar paths un poquito mas grandes

Seleccionar arbitrariamente paths complejos

Branch Coverage



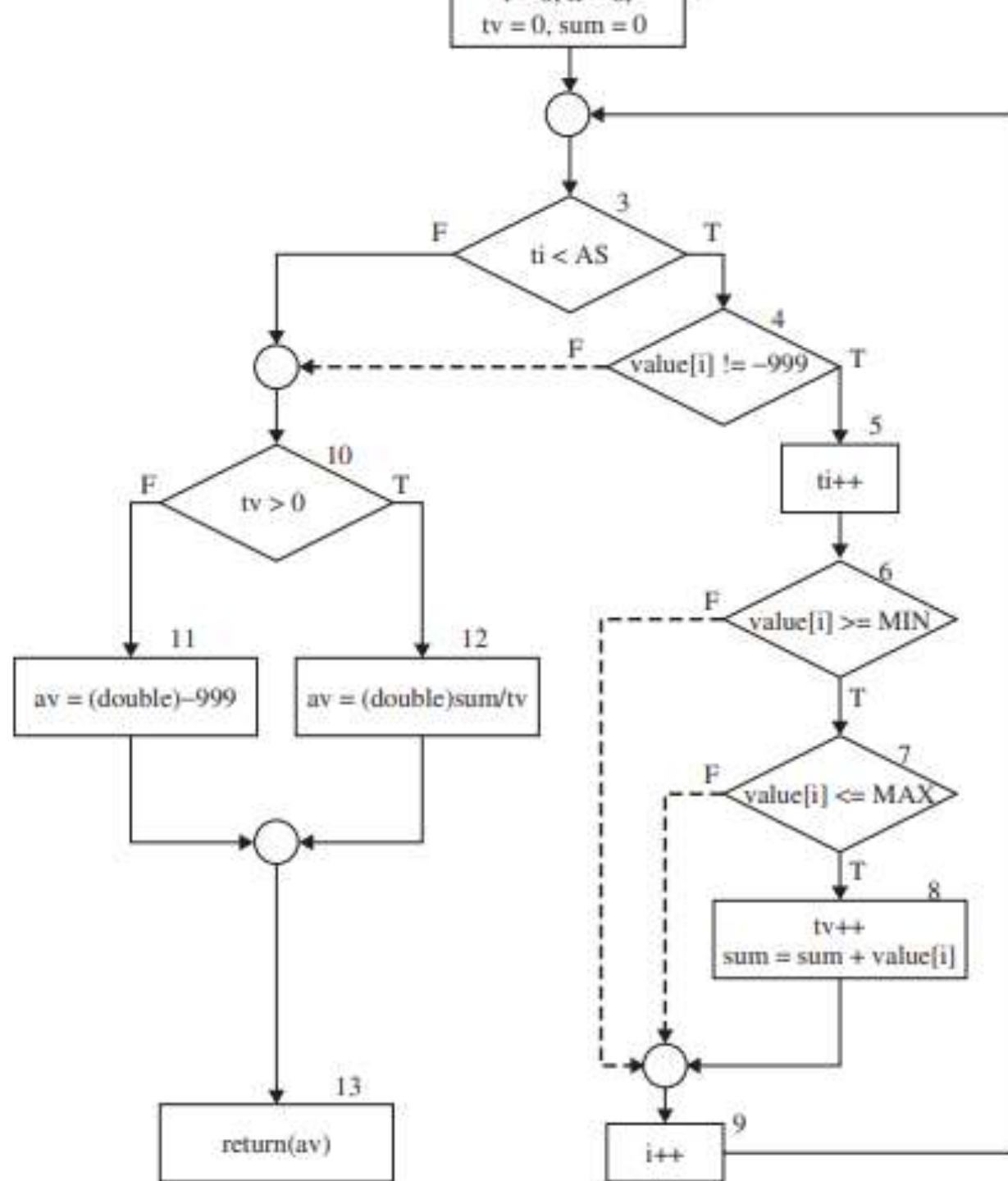
Rectángulos: 1



Rombo: 2

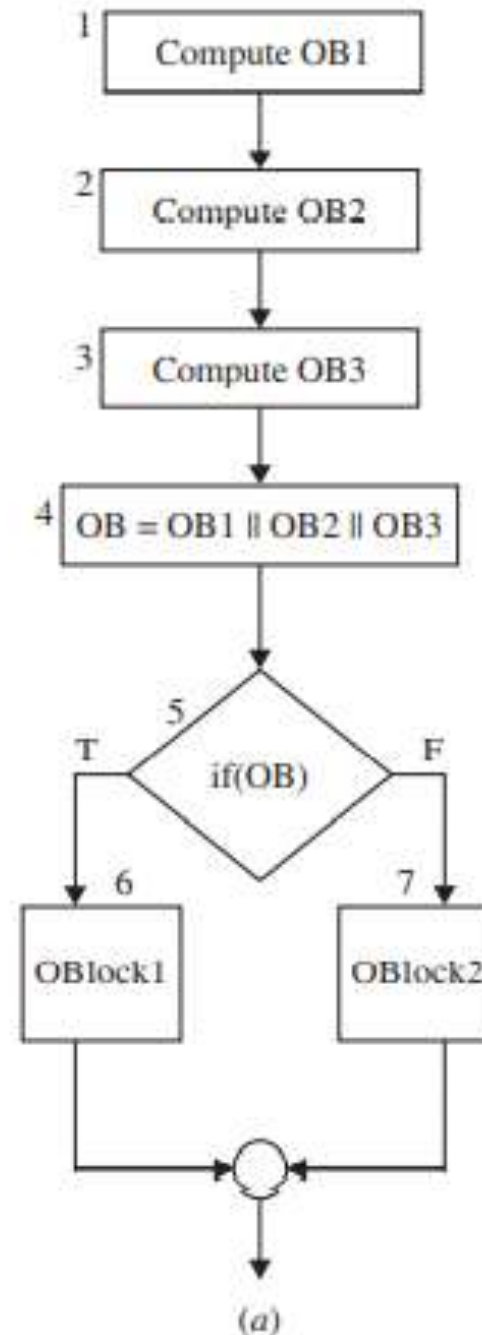


Salida: 0



Predicate Coverage

- También conocido como condition coverage
- Cada expresion booleana es evaluado como verdadero y como falso



Generando datos de entrada

Vector de entrada:

- Parámetros
- Variables y constantes globales
- Archivos
- Conexión internet
- Timers

Predicado: Función lógica evaluada en un punto de decisión

- For (...)
- If (...)
- Booleano

Path Predicate

- Conjunto de predicados asociados a un path

1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Figure 4.10 Example of a path from Figure 4.7.

```
ti < AS    ≡ True
value[i] != -999 ≡ True
value[i] >= MIN ≡ True
value[i] <= MAX ≡ True
ti < AS    ≡ False
tv > 0     ≡ True
```

Figure 4.11 Path predicate for path in Figure 4.10.

Interpretación de predicados

- Proceso de sustituir operaciones simbólicamente en un path determinado, para así poder expresar el predicado solamente en términos de vector de entrada y vector de constantes

```
public static int SymSub(int x1, int x2){  
    int y;  
    y = x2 + 7;  
    if (x1 + y >= 0)  
        return (x2 + y);  
    else return (x2 - y);  
}
```

Path Predicate Expression

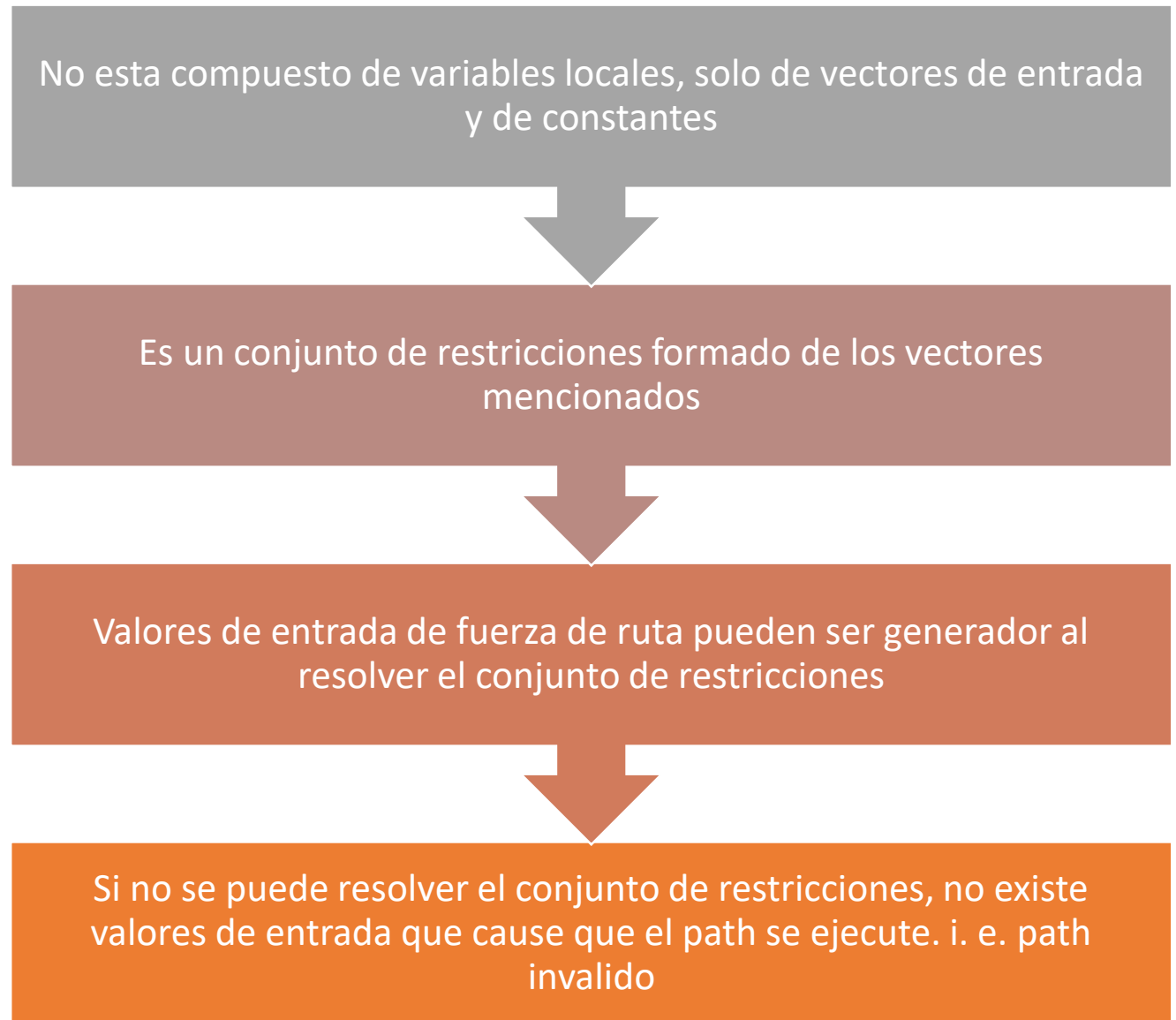


TABLE 4.7 Interpretation of Path Predicate of Path in Figure 4.10.

| Node | Node Description | Interpreted Description |
|--------------|--|--|
| 1 | Input vector: < value[], AS, MIN, MAX > | |
| 2 | i = 0, ti = 0, tv = 0, sum = 0 | |
| 3(T) | ti < AS | 0 < AS |
| 4(T) | value[i] != - 999 | value[0] != - 999 |
| 5 | ti++ | ti = 0 + 1 = 1 |
| 6(T) | value[i] >= MIN | value[0] >= MIN |
| 7(T) | value[i] <= MAX | value[0] <= MAX |
| 8 | tv++ sum = sum + value[i] | tv = 0 + 1 = 1 sum = 0 + value[0] = value[0] |
| 9 | i++ | i = 0 + 1 = 1 |
| 3(F) | ti < AS | 1 < AS |
| 10(T) | tv > 0 | 1 > 0 |
| 12 | av = (double) sum/tv | av = (double) value[0]/1 |
| 13 | return(av) | return(value[0]) |

Note: The bold entries in column 1 denote interpreted predicates.

| | | | | |
|----------|---------|---------|-------|-----|
| | 0 < AS | ≡ True | | (1) |
| value[0] | != -999 | ≡ True | | (2) |
| value[0] | >= MIN | ≡ True | | (3) |
| value[0] | <- MAX | ≡ True | | (4) |
| | 1 < AS | ≡ False | | (5) |
| | 1 > 0 | ≡ True | | (6) |

Ejemplo de path invalido

1-2-3(T)-4(F)-10(T)-12-13.

Generar datos de entrada desde la Path Predicate Expression

Resolver el path predicate expression

| | | | | |
|----------|---------|---------|-------|-----|
| | 0 < AS | ≡ True | | (1) |
| value[0] | != -999 | ≡ True | | (2) |
| value[0] | >= MIN | ≡ True | | (3) |
| value[0] | <= MAX | ≡ True | | (4) |
| | 1 < AS | ≡ False | | (5) |
| | 1 > 0 | ≡ True | | (6) |

Figure 4.13 Path predicate expression for path in Figure 4.10.

```
AS    = 1  
MIN   = 25  
MAX   = 35  
value[0] = 30
```

Posible
solución

Posibles errores

Generar datos de prueba para satisfacer el criterio de selección

Generar pruebas adicionales mas grandes

Injectar errores