Introducción al Testing de Estructural

Maximiliano Cristiá

Ingeniería de Software 2
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

2021

Resumen En este apunte de clase se introducen brevemente los conceptos básicos del testing estructural basado en flujo de control.

Índice

1. Introducción

Como ya hemos explicado, en el testing estructural los casos de prueba se seleccionan según la estructura del código fuente del programa que se está testeando.

Los conjuntos de prueba que se seleccionan siguiendo técnicas de testing estructural deben cumplir con lo que se denomina *criterio de selección de casos de prueba*. Un criterio de selección de casos de prueba, o simplemente criterio de prueba o criterio de selección, es un subconjunto de $\mathbb{F}\,ID$ donde ID es el dominio de entrada del programa que se está testeando –este concepto fue presentado en el capítulo anterior. Es decir que un criterio de selección define los conjuntos de prueba que se pueden utilizar para testear el programa. Si C es un criterio de selección y T es un conjunto de prueba que pertenece a C, se dice que T *satisface* C. Se espera que los criterios de selección sean *consistentes*. Un criterio de selección C es consistente sí y solo sí para cualesquiera conjuntos de prueba T_1 y T_2 que satisfacen C, T_1 no encuentra un error sí y solo sí T_2 tampoco lo hace [1]. De esta forma el tester debe elegir un único conjunto de prueba de un criterio puesto que todos los otros darán los mismos resultados. Aunque los criterios que se usan en la práctica no son consistentes, se trabaja como si lo fueran por lo que los testers eligen cualquier conjunto de prueba que satisface el criterio que están aplicando.

El testing estructural puede dividirse según cómo se definen los criterios de selección.

- *Testing estructural basado en el flujo de control.* En este caso los criterios de selección se definen en base a reglas que sobre el flujo de control de los programas, y en particular se tienen en cuenta los valores de verdad de las condiciones usadas en sentencias condicionales e iterativas. En otras palabras, un criterio de selección exige que los conjuntos de prueba que lo satisfacen recorran el programa siguiendo el flujo de control de una forma específica y haciendo que las condiciones asuman valores de verdad específicos.

  Esta forma de testing estructural es la más antigua y simple de aplicar, aunque la menos potente. Es la que se suele aplicar en la industria [2] y es la que estudiaremos en este capítulo.

- *Testing estructural basado en el flujo de datos.* En esta forma de testing estructural los criterios se definen de forma tal que cumplan con ciertas reglas que gobiernan el flujo de datos de los programas. No estudiaremos criterios basados en el flujo de datos pues, si bien son más poderosos que los basados en flujo de control, el cálculo de conjuntos de prueba es muy laborioso. Uno de los trabajos seminales sobre esta forma de testing estructural es el de Rapps y Weyuker [3]; en general cualquier trabajo de Elaine Weyuker sobre este tema será una excelente referencia.

En ambos casos es posible definir un orden parcial entre los criterios de forma tal que quedan ordenados según la cantidad de casos de prueba que exigen sean ejecutados. Por consiguiente los testers pueden elegir el criterio que mejor se adapte al presupuesto o tiempo disponible.

El resto del apunte está basado en [1, capítulo 6].

2. Grafo de flujo de control de un programa

Los criterios de selección basados en flujo de control se definen en base a lo que se denomina el *Grafo o Diagrama de Flujo de Control* (CFG o CFD) del programa. El CFG es una abstracción del

BasicSentence ::=
skip
| *var* := *Expr*
| call($arg_1, \ldots, arg_n$)

ConditionalSentence ::=
if *Cond* then *Program* fi
| if *Cond* then *Program* else *Program* fi
| while *Cond* do *Program*done

Sentence ::= *BasicSentence* | *ConditionalSentence*

Program ::= *Sentence* | *Program* ; *Program*

Figura 1: Gramática de un lenguaje de programación imperativo simple. Los no terminales *Cond* y *Expr* se dejan sin especificar porque no tienen influencia sobre el CFG.

programa que captura los aspectos estructurales más importantes obviando detalles específicos tanto del programa como del lenguaje de programación.

Asumamos que el lenguaje de programación con el cual se escriben los programas a testear tiene la gramática presentada en la Figura 1. En ese caso el CFG de cualquier programa se construye inductivamente como se indica más abajo, teniendo en cuenta que en los CFG las flechas representan sentencias y los nodos los puntos de entrada y salida de la sentencia respectiva.

1. Para cada *BasicSentence* se dibuja un grafo como el de la Figura 2a.
2. Si S es un *Program* cuyo CFG es G entonces el CFG del programa if *cond* then S fi es el de la Figura 2b.
3. Si S_1 y S_2 son dos *Program* cuyos CFG son, respectivamente, G_1 y G_2, entonces el CFG del programa if *cond* then S_1 else S_2 fi es el de la Figura 2c.
4. Si S es un *Program* cuyo CFG es G entonces el CFG del programa while *cond* do Sdone es el de la Figura 2d.
5. Si S_1 y S_2 son dos *Program* cuyos CFG son, respectivamente, G_1 y G_2, entonces el CFG del programa S_1 ; S_2 es el de la Figura 2e.

El grafo correspondiente a una secuencia de *BasicSentence* se puede abreviar con un grafo como el de la Figura 2a, es decir dos nodos y una sola flecha, puesto que el flujo de control indefectiblemente debe seguir la secuencia.

La Figura 3a es la codificación en nuestro lenguaje del algoritmo de Euclides para calcular el máximo común divisor (MCD) entre dos números naturales, y 3b es el CFG correspondiente. En este caso hemos etiquetado las flechas con las sentencias del programa para que sea más simple comprender cómo se genera el CFG a partir del programa, aunque esto no es necesario.

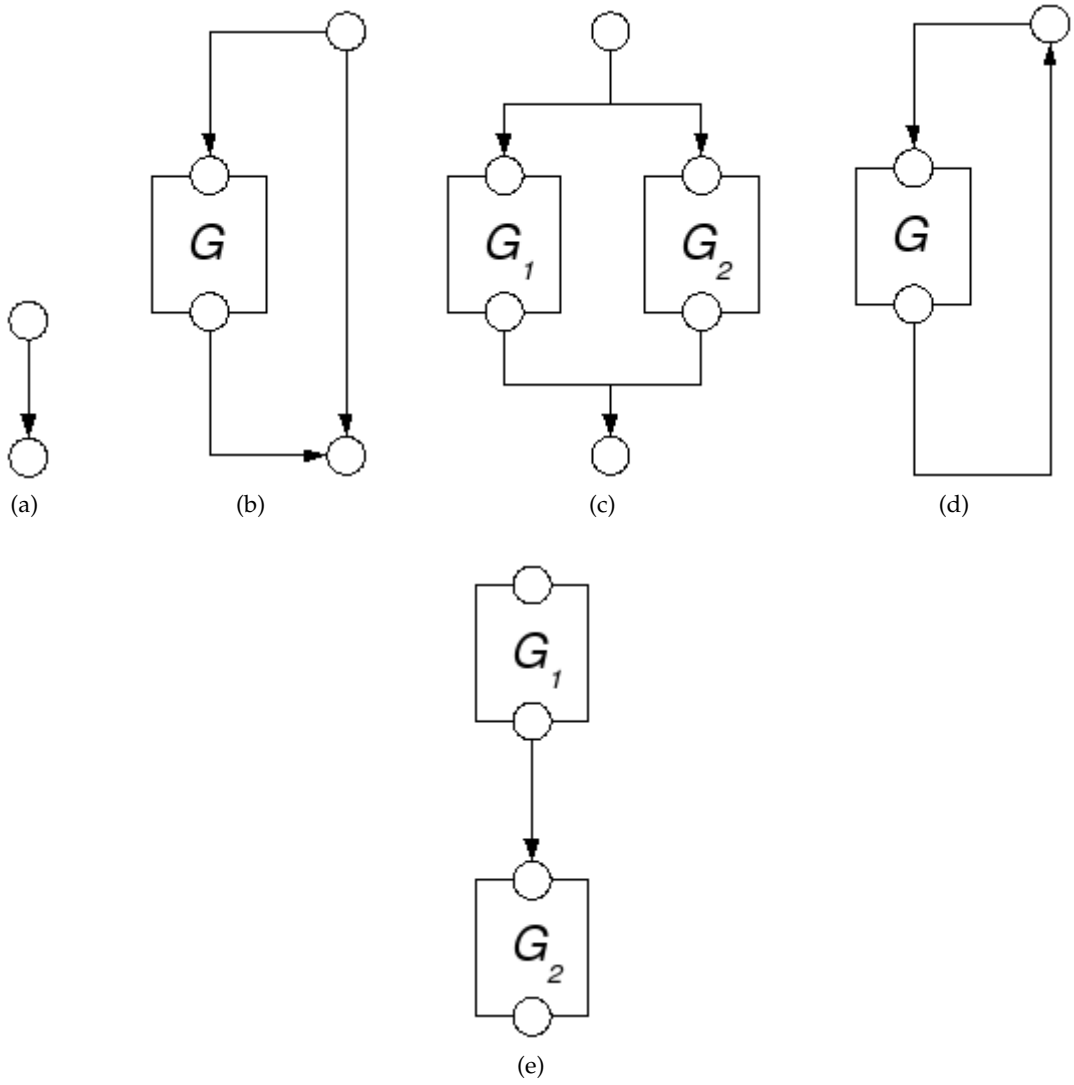
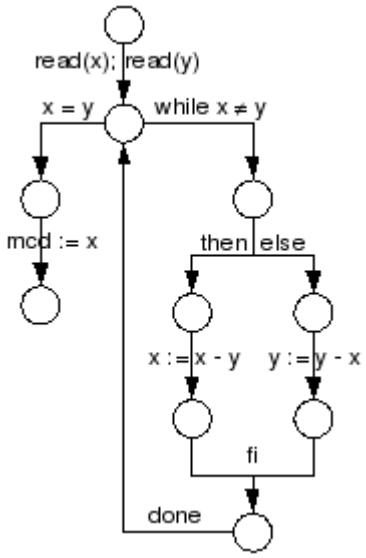


Figura 2: Construcción inductiva del CFG [1].

```
read(x); read(y);
while x ≠ y do
  if x > y then
    x := x − y;
  else
    y := y − x;
  fi
done
mcd := x;
```

(a)



(b)

Figura 3: Algoritmo de Euclides para calcular el MCD (3a) y el CFG correspondiente (3b).

Notar que hemos abreviado las dos primeras sentencias del programa con una sola flecha y dos nodos, como explicamos más arriba.

Se supone que el CFG debe ser obtenido automáticamente a partir del código fuente del programa.

Los criterios basados en flujo de control se definen en base a los caminos del CFG que se deben recorrer para testear el programa. Es decir se deben seleccionar los casos de prueba que sean necesarios para que entre todos ellos se recorran todos los caminos que exige el criterio que se esté aplicando. En las secciones que siguen veremos los criterios de testing estructural basado en flujo de control más comunes. Los criterios están ordenados desde el menos exigente hacia el más exigente.

3. Criterio de cubrimiento de sentencias

En realidad la definición de este criterio no requiere analizar el CFG debido a que es muy simple.

> Seleccionar un conjunto de prueba T tal que, al ejecutar P para cada d en T, cada sentencia básica de P es ejecutada al menos una vez.

Por ejemplo nuestra implementación del algoritmo de Euclides puede testearse con el conjunto de prueba $CS = \{\langle x = 4, y = 3\rangle, \langle x = 3, y = 4\rangle\}$, para cumplir con el criterio.

4. Criterio de cubrimiento de flechas

Este criterio se enuncia de la siguiente manera:

> Seleccionar un conjunto de prueba T tal que, al ejecutar P para cada d en T, cada flecha del CFG de P es atravesada al menos una vez.

Importante. *En este criterio y en todos los que siguen se debe tener en cuenta que el arco que sale de un bucle cuando la condición es falsa debe ser recorrido al menos una vez sin haber recorrido el interior del bucle en la misma ejecución. En otras palabras es necesario un caso que no lleve el flujo de control al interior del bucle.*

Un conjunto de prueba que satisface el criterio es $CF = \{\langle x = 4, y = 3\rangle, \langle x = 3, y = 4\rangle, \langle x = 3, y = 3\rangle\}$. Observar que el último caso fue agregado debido a la aclaración efectuada más arriba porque de otra forma sin ese caso igual se hubieran recorrido todas las flechas.

El cubrimiento de flechas es un criterio más fuerte que el de cubrimiento de sentencias lo cual se evidencia cuando el programa contiene sentencias if-then-else o bucles, pues hay flechas que no representan sentencias básicas y que deben ser recorridas con casos específicos.

5. Criterio de cubrimiento de condiciones

El criterio de cubrimiento de condiciones se enuncia de la siguiente forma:

> Seleccionar un conjunto de prueba T tal que, al ejecutar P para cada d en T, cada flecha del CFG de P es atravesada al menos una vez y las proposiciones simples que forman condiciones toman los dos valores de verdad.

Importante. *En este criterio y en todos los que siguen se debe tener en cuenta que la condición que gobierna a un bucle debe ser falsa de todas las formas indicadas por cada criterio sin antes haber sido verdadera (en el momento de evaluarla) en la misma ejecución.*

Este criterio no puede ser claramente ejemplificado con la implementación del algoritmo de Euclides debido a que no contiene condiciones compuestas. Por este motivo en la Figura 4a introducimos una implementación de la búsqueda secuencial en un arreglo de longitud *max* cuyo CFG se muestra en la Figura 4b. Un conjunto de prueba que satisface el criterio es $CC = \{\langle max = 0, a = [\,], elem = 2\rangle\ \langle max = 2, a = [5,2], elem = 2\rangle, \langle max = 2, a = [5,2], elem = 7\rangle\}$. Observar que el criterio exige testear el programa según algunas de las alternativas funcionales más importantes:

- El arreglo es vacío.
- El arreglo no es vacío y el elemento que se busca está en el arreglo.
- El arreglo no es vacío pero el elemento que se busca no está en el arreglo.

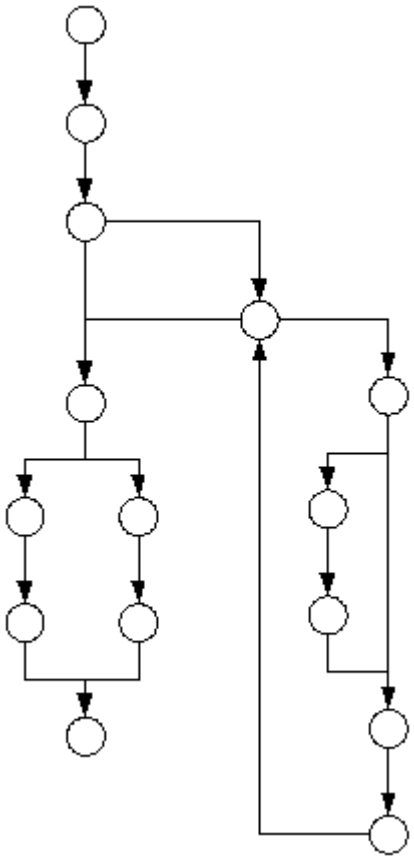
Sin embargo, no se exige testear el programa, por ejemplo, con un arreglo donde el primer elemento sea el elemento buscado, o con un arreglo donde el último elemento sea el elemento buscado, etc.

6. Criterio de cubrimiento de caminos

Este criterio en general es impracticable pero se lo suele incluir en las presentaciones por una simple cuestión de completitud teórica.

```
search(int a[ ], int max, int elem) : int
found := false;
if max ≠ 0 then
  counter := 1;
  while not found and counter ≤ max do
    if a[counter] = elem then
      found := true;
     fi
    counter := counter + 1;
  done
 fi
if found then
  return counter − 1;
 else
  return  − 1;
 fi
```

(a)



(b)

Figura 4: Búsqueda secuencial.

> Seleccionar un conjunto de prueba T tal que, al ejecutar P para cada d en T, se recorren todos los caminos completos posibles del CFG P.

Dado que es impracticable testear la mayoría de los programas siguiendo este criterio, se lo debe tener como una referencia para tratar de cubrir los caminos más críticos. La heurística mínima que debe seguirse cuando aparecen bucles es la siguiente:

- Iterar sobre cada bucle cero veces.
- Iterar sobre cada bucle el máximo número de veces posible.
- Iterar sobre cada bucle un número promedio de veces.

7. Otros criterios

En la práctica se verán dos criterios más que surgen a partir del criterio de cubrimiento de condiciones.

Referencias

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering (2nd ed.)*. Prentice Hall, 2003.

[2] A. Page, K. Johnston, and B. Rollison, *How We Test Software at Microsoft*. Microsoft Press, 2008.

[3] S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection," in *ICSE '82: Proceedings of the 6th international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1982, pp. 272–278.