2 Métodos combinatorios

Las pruebas pueden aplicarse de muchas maneras, es decir, existen diferentes formas de preparar casos de prueba. En este capítulo se presentan dos formas de prueba muy fáciles de utilizar, aún cuando no siempre resultan suficientes: la prueba de particiones (o de dominios) y la de valores a la frontera. Ambas descansan sobre el modelo combinatorio y muchas veces se usan conjuntamente. Son de utilidad especialmente en pruebas de unidad, aunque también resultan útiles en algunas pruebas a nivel de sistema.

2.1 Modelo combinatorio

El modelo combinatorio supone que el software es una gran caja negra con entradas y salidas. Las diferentes variables de entrada reciben valores desde el exterior del programa, los cuales se combinan de diversas maneras para producir salidas específicas. Este modelo considera el software a probar como un gran clasificador (ver Figura 2.1) que separa todas las posibles combinaciones de entrada en unas pocas categorías y cada una de éstas se ejecuta como un subprograma independiente de los demás; es decir como si se tuviese un conjunto reducido de funciones que se eligen al entrar y, una vez iniciado, ya no hay caminos alternos, debiendo seguir hasta terminar. Desde el punto de vista de programación es como si el software fuera un gran “case” (el case de Pascal o el switch de Java), donde las diferentes entradas determinan una opción.

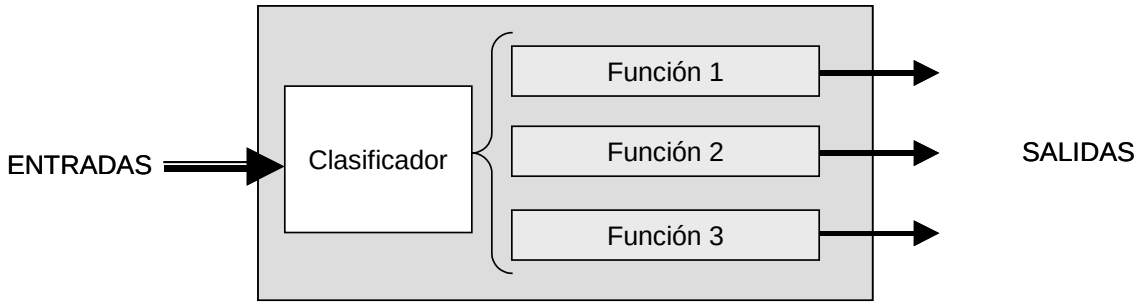

Figura 2.1 Modelo combinatorio

Por ejemplo, suponga un programa que recibe dos variables alfa y beta, ambas de tipo entero y una sola salida omega, de tipo real (punto flotante). La salida se calcula de acuerdo a la siguiente tabla según los valores de alfa y beta:

	alfa <=0	alfa >0
beta <=0	beta/alfa	-beta
beta >0	-alfa	alfa/beta

El software del ejemplo puede pensarse como una caja negra con dos entradas; al llegar datos, se analiza cuál de los cuatro casos le corresponde y según eso se produce la salida.

Por su naturaleza, este modelo se puede aplicar usando la información de los requerimientos del usuario, cuando son detallados, o las especificaciones que se establecen en el documento de diseño. Para poderse aplicar se requiere que:

a) todas las variables de entrada sean independientes entre sí, es decir, se pueden dar en cualquier orden y se les puede considerar simultáneas
b) puedan identificarse las diferentes funciones o comportamientos (es decir, tipos diferentes de salidas)
c) puedan asociarse un grupo de valores de las entradas con cada comportamiento

Una vez establecido este modelo, la suposición principal dice que basta elegir representantes de cada clase para probar el software, es decir, en vez de un gran número de combinaciones que podrían ser infinitas (como se vio en el capítulo anterior), bastará con un número relativamente pequeño de casos de prueba.

A veces se construye este modelo comenzando por las salidas, tanto las deseadas como las no deseadas. A partir de ahí, se razona hacia atrás para determinar qué entradas producirían cada salida.

Existen varios métodos correspondientes a este modelo, de los cuales hay tres destacados:

a) particiones equivalentes o dominios
b) valores a la frontera
c) tablas de decisión, llamado también grafo causa-efecto

En las siguientes secciones se tratarán los dos primeros.

La presencia de defectos a veces invalida las suposiciones de este modelo, haciéndolo insuficiente. En general se recomienda complementarlo con pruebas de tipo estructural.

2.2 Prueba de particiones equivalentes

La prueba de particiones equivalentes, llamada también de dominios, es una técnica basada en el modelo combinatorio, de tipo funcional o caja negra, que puede aplicarse en diferentes niveles, desde funciones y procedimientos de la programación procedural, los métodos de las clases bajo orientación a objetos, hasta los sistemas completos. Por ahora se denominará simplemente unidad al software bajo prueba.

Una unidad es una pieza de código que recibe una serie de entradas y produce una serie de salidas, que pueden incluir un valor de retorno (en el caso de funciones). Las entradas pueden ser explícitas, como los parámetros o valores leídos de consola, pero también pueden ser implícitos, como valores tomados de estructuras de datos globales, de archivos o bases de datos.

En forma similar, las salidas pueden ser explícitas, como los valores de regreso, escrituras a consola o impresora y parámetros pasados por nombre que sean

modificados, o implícitas a archivos, bases de datos o estructuras de datos globales.

Cada variable, de entrada o salida, tiene un **dominio**, es decir, pertenece a un conjunto de donde toma su valor. El dominio puede ser el conjunto de valores asociados con un tipo de datos o sólo con una parte. Por ejemplo, los números de punto flotante de 32 bits o los enteros positivos de 16 bits. En algunos casos puede expresarse como una colección discreta de valores, pero más generalmente como un intervalo. En este capítulo se considera únicamente el caso de entradas.

Formalmente, se tiene que para una variable ***x*** su dominio será ***Dx***.

Cuando la unidad bajo prueba recibe un valor en una variable, puede suceder una de dos cosas:

a) realiza la misma función o procedimiento para cualquiera de los valores del dominio
b) reacciona de manera diferente según el valor recibido

Ejemplo:

Si se tiene una función que recibe una cuerda de caracteres y regresa la longitud, siempre se tendrá un mismo funcionamiento; aún si se da una cuerda vacía, su longitud está definida (cero).

Ejemplo:

Si se tiene una función que calcula el logaritmo de un número entero positivo, el resultado siempre será calculado de la misma forma. Si el valor de entrada es negativo o cero, siempre se producirá un error o una excepción. Es decir, existen dos comportamientos diferentes.

Ejemplo:

En cambio, si se tiene una función de números reales que se evalúa de acuerdo con

$$F(x) = \begin{cases} \sqrt[3]{(x+5)} \; Si \quad x > 4 \\ 3x^3 \quad Si \quad x < -2 \\ 0 \quad en\,otro \quad caso \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Se tendrán tres comportamientos diferentes: uno para valores negativos menores a -2, otro entre -2 y 4 y otro más para valores superiores a 4.

En los últimos dos ejemplos el dominio se puede subdividir en varios subdominios, cada uno de los cuales será una clase de equivalencia respecto a la función o procedimiento que realiza la unidad. Otras clases de equivalencia se forman con los valores que no son permitidos y que son atrapados por las validaciones.

Ejemplos:

Para el caso de la función logaritmo del ejemplo anterior, se tendrán dos clases de equivalencia:

a) los valores positivos, para los cuales la función regresa el logaritmo del número recibido
b) los enteros negativos o cero, para los cuales se recibe un mensaje de error

Para la función F(x) del ejemplo anterior (Ecuación 1) se tendrán tres clases de equivalencia correspondientes a los tres casos:

a) (-∞, -2) (valores menores a –2)
b) [-2, 4] (valores entre –2 y 4, inclusive)
c) (4, +∞) (valores mayores a 4)

2.2.1 Casos de prueba

Para el caso de una variable, el método de dominios propone que se construya un caso de prueba por cada clase de equivalencia, lo cual se considera suficiente. Para mayor seguridad, se acostumbra combinar con el método de valores a la frontera, que se presenta en la siguiente sección, así como algún método estructural (caja blanca, a tratar en el Capítulo 3).

Continuando con los ejemplos anteriores, se prepararían casos de prueba como los siguientes:

Para el caso de la longitud de una cuerda:

Entrada	Salida esperada
“”	0
“G”	1
“Este es un ejemplo largo”	24

Para la función logaritmo:

entradas	Salidas esperadas
-10	"Error"
0	"Error"
1	0
100	2

Para la función F(x):

Entradas	Salidas esperadas
-80	-1536000
-2.1	-27.783
-2	0
0	0
4	0
4.1	2.08
60	4.02

2.2.2 Entradas múltiples

En la práctica es poco probable tener una unidad que recibe únicamente una entrada. En general hay más. Cuando se tiene más de una variable, hay que identificar los dominios de cada variable y sus subdominios; luego se deben seleccionar valores de cada subdominio para todas las variables y combinarlos formando los casos de prueba. Matemáticamente, el proceso equivale a obtener el dominio de todas las entradas utilizando el producto cartesiano de los dominios individuales, los subdominios del dominio resultante y de ahí los casos de prueba. Es decir: para las variables *x*, *y*, *z*, con dominios *Dx*, *Dy*, *Dz*, el dominio conjunto será: ***D*** = *Dx* X *Dy* X *Dz*. Un ejemplo con dos variables es la identificación de usuarios que se presentó en el capítulo anterior.

Ejemplo:

Para una función de identificación de usuarios que recibe un nombre y una contraseña, las parejas de nombre registrado y contraseña correcta formarán una clase de equivalencia que recibirá la aceptación del sistema; las parejas con nombre registrado y contraseña inválida formarán otra que recibirá un mensaje "contraseña equivocada"; y aquellas parejas de nombre no registrado forman otra clase de equivalencia que recibirá un mensaje "nombre desconocido". Además existen la clase donde la contraseña está vacía y la clase donde el nombre está vacío, las cuales deben dar error.

Para el caso de las claves, supóngase que se tienen registradas las parejas siguientes:

Nombre	**Contraseña**
Julián	1K809vX
Margarita	8n662zW
Elsa	V456JhP
Roberto	72wW55X
Guillermina	B87hG34

Los casos de prueba adecuados serían:

Entradas	**Salidas esperadas**
(Margarita,8n662zW)	Aceptada
(Roberto,72Ww55x)	Contraseña equivocada
(Pedro,7R45Ty2)	Nombre desconocido
(Julián,)	Contraseña equivocada
(,)	Nombre desconocido

2.2.3 Procedimiento de preparación de casos de prueba

Formalizando el procedimiento para preparar casos de prueba con el método de particiones es como sigue:

1. Hallar el dominio y subdominios de cada variable
2. Formar el dominio resultante de todas las variables y sus subdominios
3. Seleccionar valores de cada subdominio, formando los casos de prueba

Este procedimiento puede automatizarse, de modo que se pueda reducir el trabajo, ya que el número de casos puede crecer enormemente. También se emplean métodos estadísticos para seleccionar un conjunto de casos que cubra la mayor parte de las combinaciones, pero no se tratarán por ahora.

En algunas circunstancias, junto con los casos de prueba debe establecerse un procedimiento de prueba. Ésto es necesario cuando debe emplearse una base de datos o una estructura de datos global que debe encontrarse en cierto estado (por ejemplo un árbol que deba tener al menos dos ramas de longitud mayor a dos), o cuando se requieren determinadas condiciones del ambiente (hardware, sistema operativo, software auxiliar, persona, etc.). De hecho estos elementos muestran que la unidad depende de un estado global. Por estado se entiende el conjunto de variables con sus valores en un instante dado.

El procedimiento de prueba indicará los preparativos necesarios, condiciones que deben satisfacerse y la forma de aplicar cada caso de prueba, la cual puede ser en dos formas:

a) restaurando el estado inicial antes de cada caso de prueba, o
b) sin restaurar el estado inicial, como si los casos se encadenaran uno tras otro.

2.2.4 Usos del método de particiones

El método de dominios resulta de utilidad para pruebas:

- de unidad, especialmente de algoritmos numéricos
- pruebas de sistema donde las entradas se dan conjuntamente

El método puede utilizarse con software tradicional y también orientado a objetos, siempre que éste no dependa de la historia. En el caso de objetos se aplica a nivel de métodos individuales.

En el caso de software de objetos donde el comportamiento depende de su historia de manera sencilla, el método puede adaptarse como sigue:

- Para cada variable del estado se toman los valores del estado inicial como si fueran entradas,
- Se usan los valores que se espera se produzcan en las variables de estado como parte de las salidas

De esa manera se puede tratar como un problema tradicional.

Si la dependencia del estado es compleja debe usarse un método orientado a objetos, como el de rebanadas, que se tratará en un capítulo posterior.

De cualquier forma, debe recordarse que el método supone que las entradas se dan de una vez o que el orden de entrada no es relevante y además que son pocas. Cuando el número de variables crece, resulta más práctico emplear el

método de tablas de decisión, llamadas también de grafo causa-efecto (ver el libro de R. Binder 1999).

A continuación se presentan tres ejemplos donde se aplica el método de dominios.

2.2.5 Ejemplo sin estructuras globales: la fecha del día siguiente

Se tiene el siguiente problema: se desea probar una unidad con las especificaciones siguientes:

a) Recibe tres datos numéricos: el día, el mes y el año, para una fecha posterior al año 1899 y anterior al año 2200, con las convenciones del calendario juliano.
b) Escribe, también como tres datos enteros, la fecha del día siguiente al que leyó.
c) Si recibe una fecha errónea, debe avisar del error, indicando si faltó un dato, alguno está equivocado o fuera del intervalo propuesto.

Para este problema tenemos lo siguiente:

1. Dominios y subdominios: el dominio en los tres casos es el de los enteros; sin embargo hay un dominio de interés donde son relevantes los datos, que se muestra en la segunda columna. Además se muestran todos los subdominios en la tercera columna.

variable	**dominio**	**subdominios**
Mes	{1, ... ,12}	(- ∞, 0), {2}, {4, 6, 9, 11}, {1, 3, 5, 7, 8, 10, 12} (12, +∞)
Día	{1, 2, ..., 31}	(- ∞, 0), {1, 2, ... , m} (m, +∞), m= 28, 29, 30 ó 31 según el caso
Año	[1900, 2199]	(- ∞, 1899], {a | a ∈ [1900, 2199] y a bisiesto} {a | a ∈ [1900, 2199] y a no es bisiesto}, [2200, +∞)

2. Casos de prueba

Entradas			**Salidas esperadas**		
día	mes	año	día	mes	año
-10	-2	1803	Datos fuera de intervalo		
-9	-3	1956	Datos fuera de intervalo		
-11	4	2002	Datos fuera de intervalo		
15	6	1650	Datos fuera de intervalo		

15	6	2300	Datos fuera de intervalo		
1	2	1957	2	2	1957
12	2	1957	13	2	1957
28	2	1957	1	3	1957
29	2	1957	Datos fuera de intervalo		
28	2	2000	29	2	2000
29	2	2000	1	3	2000
1	4	1987	2	4	1987
17	4	1988	18	4	1988
30	4	1988	1	5	1988
31	4	1988	Datos fuera de intervalo		
1	8	1999	2	8	1999
20	8	1999	21	8	1999
30	8	1999	31	8	1999
31	8	1999	1	9	1999

2.2.6 Ejemplo con estructura de datos global: búsqueda binaria en una lista

En muchos problemas se tiene una lista de datos que se usa para identificar valores. El programa, o fragmento de él, lee un valor, lo busca en la lista, almacenada en un arreglo en las posiciones cero a **n - 1** y regresa la posición que ocupa, si lo encontró; de otro modo regresa un menos uno. Para la búsqueda se supone que la lista está ordenada y se emplea el método de búsqueda binaria.

Suponga que la lista está en el arreglo **Lista** con los valores de abajo y la variable de entrada es **llave**.

Lista:

-12	-5	1	4	17	39	78	90	107	200	202	299

Los valores de la lista constituyen el dominio de interés y todos los demás valores enteros deben dar error. Entonces, algunos casos de prueba serán:

Entrada	Salida esperada
-9	-1
39	5
299	11
600	-1

2.2.7 Ejemplo de objetos con estado: retiro de una cuenta bancaria

Suponga que tiene una clase Cuenta que tiene dos atributos: identificador y saldo, así como dos métodos: deposita y retira, ambas con un parámetro cantidad. Se desea probar el método retira. Estrictamente, la clase tiene estado (el saldo) que

influye sobre el resultado de la operación. Si se desea utilizar el método de particiones, se puede suponer que el saldo es una variable de entrada y también de salida, en dos momentos diferentes: antes y después de aplicar el método. Para la variable cantidad se tienen dos subdominios: de cero al valor del saldo se puede efectuar el retiro regresando el valor del retiro, para valores menores a cero se tiene un error, regresando -1, y para valores mayores del saldo se rechaza la operación por fondos insuficientes regresando -2. El saldo, como es atributo, sólo tiene un dominio de cero a infinito.

Los casos de prueba serán:

saldo (antes)	cantidad	salida	saldo (después)
1000	2000	-2	1000
1000	500	500	500
1000	-100	-1	1000
0	100	-2	0

2.3 Valores a la frontera

Si se emplea el método de dominios, cualquier valor dentro de una clase de equivalencia, o subdominio, será igualmente bueno para fines de prueba. Al menos eso dice la teoría. Sin embargo, la experiencia indica que algunos valores son más propensos a generar problemas. Un grupo de valores problemáticos son aquellos que se encuentran en los límites de un subdominio. Esta situación puede deberse a

a) la implementación de proposiciones condicionales o de fin de ciclos que pueden tener pequeños errores (piense, por ejemplo, que el límite debía decir x<=20 y accidentalmente dice x<20)
b) problemas de precisión debida a la representación digital de números de punto flotante, que pueden volver iguales valores que realmente son diferentes.

Considerando esta situación, surgió una variante del método de dominios, conocida como “valores a la frontera”. En la práctica, este método es complementario del método de dominios y usualmente se aplican juntos.

En el método de valores a la frontera, para cada subdominio, se eligen los casos de prueba como sigue:

a) el valor en la frontera
b) un valor cercano a la frontera, pero dentro del subdominio
c) un valor cercano a la frontera pero fuera del subdominio

Al aplicar estas reglas a un conjunto de dominios, resultará que los valores cercanos a la frontera dentro de un subdominio serán los externos a otro y algo semejante ocurre con los externos. Así pues, en realidad no habrá necesidad de aumentar un gran número de casos de prueba.

La elección de valores “cercanos” es muy importante cuando se trata de números de punto flotante, ya que la imprecisión debida a la representación

binaria vuelve borrosa la frontera, lo cual influirá sobre los posibles errores. En cambio, para números enteros y conjuntos discretos no hay mucho problema, eligiéndose el valor anterior o siguiente del valor a la frontera.

A veces la definición de la frontera no es clara. Por ejemplo, cuando se maneja un conjunto de cuerdas alfanuméricas, a veces la frontera se refiere al número de caracteres, que no debe ser cero ni exceder un valor que indica la capacidad de un atributo o el límite que maneja un sistema.

Ejemplos

En la sección 2.2.1 se mostró un ejemplo con logaritmos en el cual había dos subdominios: el de números enteros mayores a cero (donde sí está definido el logaritmo) y el de los números enteros menores o iguales a cero (donde no está definido y debe mostrarse un defecto. Así pues, la frontera está en cero y los casos de prueba recomendables serían:

Entrada	Salida esperada
-1	“Error”
0	“Error”
1	0

En la misma sección se presentó la función F(x), con x real, que tiene tres subdominios:

a) $(-\infty, -2)$
b) $[-2, 4]$
c) $(4, +\infty)$

Aquí se tienen dos fronteras: -2 y 4, así que los casos de prueba serían:

Entrada	Salida esperada
-2.001	-24.036
-2.0	0
-1.999	0
3.999	0
4.0	0
4.001	2.080

(actualizado al 15/11/2006)