Técnicas de testing combinatorio y de mutación
Primera parte

Macario Polo Usaola
Grupo Alarcos
Departamento de Tecnologías y Sistemas de Información
Universidad de Castilla-La Mancha
Ciudad Real, España

1

Contenidos

1. Introducción
2. Criterios de cobertura de artefactos software
3. Valores de prueba
4. Estrategias de combinación para casos de prueba
5. Técnicas para generación de casos de prueba
6. Pruebas utilizando mutación
7. Casos de prueba redundantes
8. Recapitulación

2

Introducción

- Importancia del testing y *bla bla bla*
- Necesidad de disponer de buenos casos de prueba y de medir cuantitativamente su calidad
  - Tanto desde un punto de vista de caja negra
  - Como del punto de vista de caja blanca

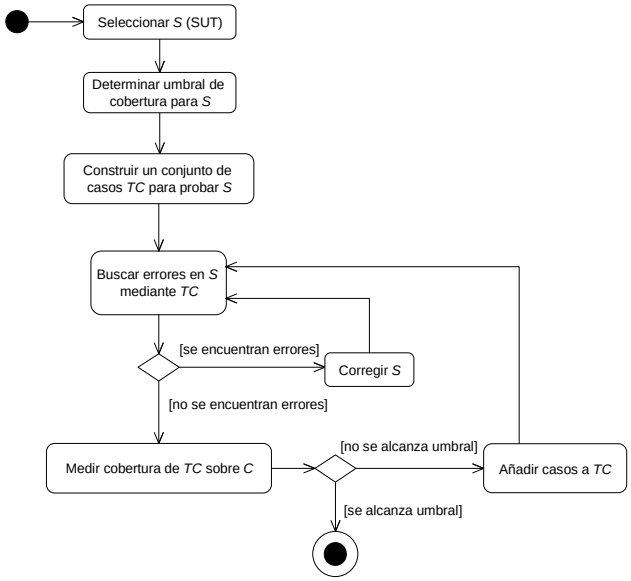
3

Criterios de cobertura

- Los criterios de cobertura se utilizan para conocer cuantitativamente la "cantidad de producto" que se ha probado
- El valor deseado se establece antes de realizar las pruebas, y el hecho de alcanzarlo puede utilizarse como condición de parada
- Así pues: si encontramos errores, los corregimos; si no, continuamos probando hasta alcanzar el valor umbral que se haya predeterminado

4

Un posible modelo de trabajo



Utilidad de los criterios de cobertura

- Determinar las porciones del sistema que no se han probado
- Escribir nuevos casos de prueba para recorrer esas áreas inexploradas
- Conocer cuantitativamente el valor de la cobertura que, indirectamente, puede ser un valor muy adecuado para determinar la calidad o la fiabilidad del sistema

Criterios de cobertura para código fuente

- Sucede que la determinación de ese valor cuantitativo puede ser medido de muchas formas
- En código fuente:
  - Sentencias
  - Decisiones
  - Condiciones
  - Condiciones/decisiones
  - Toda la tabla de verdad de la decisión
  - Condición/decisión modificada

7

Cobertura de sentencias

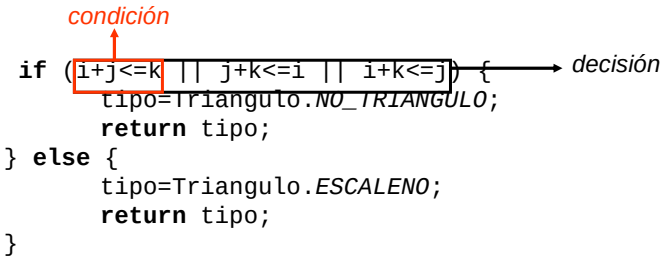
- Se alcanza cuando se ejecuta el 100% de las sentencias del programa
- Puede entenderse que una sentencia es aquello que termina en un punto y coma

- Ejemplos con Eclemma Code Coverage:
  - Triángulo.paper.TestTriángulo1
  - SudokuSolver.tests.Test1

8

Cobertura de decisiones (I)

- Se alcanza cuando se ejecutan a *true* y a *false* todas las decisiones del programa



Cobertura de decisiones (y II)

```
if (i+j<=k || j+k<=i || i+k<=j) {
      tipo=Triangulo.NO_TRIANGULO;
      return tipo;
} else {
      tipo=Triangulo.ESCALENO;
      return tipo;
}
```

- El caso (4, 6, 10) hace cierta la decisión, y (4, 6, 9) la hace falsa, con lo que se verifica el criterio
- Sin embargo, dejamos de probar las condiciones *j+k<=i* y *i+k<=j*

Cobertura de condiciones (I)

- Busca una granularidad "más fina" que la cobertura de decisiones
- Se alcanza cuando todas las condiciones de todas las decisiones se evalúan a *true* y a *false* al menos una vez:
  - *true, true, true* hacen ciertas las tres condiciones
  - *true, false, false,* *y false, true, true* las hacen falsas
  - Por tanto, se verifica el criterio

```
if (A && B && C) {
      talCosa();
} else {
      talOtra();
}
```


Cobertura de condiciones (y II)

```
if (true && true && true) {
    talCosa();
} else {
    talOtra();
}

if (true && false && false) {
    talCosa();
} else {
    talOtra();
}
if (false && true && true) {
    talCosa();
} else {
    talOtra();
}
```

```
if (true) {
    talCosa();
} else {
    talOtra();
}

if (false) {
    talCosa();
} else {
    talOtra();
}
```

Cobertura de condiciones/decisiones

- Se requiere que cada condición se evalúe a *true* y a *false* al menos una vez, y que cada decisión también se evalúe a *true* y a *false* también al menos una vez.

```
if (true && true && true) {
   talCosa();
} else {
   talOtra();
}
```

```
if (false && false && false) {
      talCosa();
      } else {
      talOtra();
}
```

- Como se observa, se está verificando este criterio

13

Cobertura modificada de condición/decisión (MC/DC) (I)

- Requiere que cada posible valor de una condición determine la salida de la decisión al menos una vez
- Supongamos *if (A or B or C)*
  - Debemos ir a la rama *true* gracias a que *A* sea *true* y las demás no; a que *B* sea *true* y las demás no; a que *C* sea *true* y las demás no. Tendremos por tanto tres casos de prueba: *(true, false, false), (false, true, false), (false, false, true)*
  - Del mismo modo, debemos ir a la rama *false* gracias a la contribución que cada condición hace a la decisión independientemente. En este caso: *(false, false, false)*, ya que se trata de tres condiciones separadas por *or*.

14

Cobertura modificada de condición/decisión (MC/DC) (II)

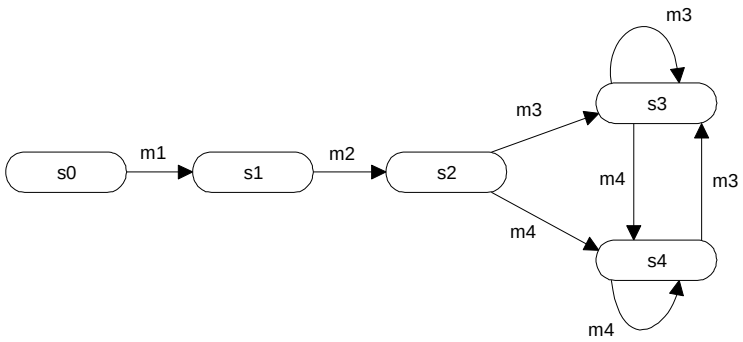
- Supongamos *if (A and B and C)*
  - Como antes, debemos ir a la rama *true* gracias a la contribución independiente de cada condición. Puesto que son tres condiciones separadas por *and*, el único caso es *(true, true, true)*
  - Para la rama *false*, tendremos *(false, true, true)*, *(true, false, true)* y *(true, true, false)*

Cobertura modificada de condición/decisión (MC/DC) (y III)

- Supongamos *if (A and (B or C))*
- Los casos serían:
  - *(true, true, false)*
  - *(true, false, true)*
  - *(false, true, true)*
  - *(true, false, false)*

Criterios de cobertura para máquinas de estado (I)

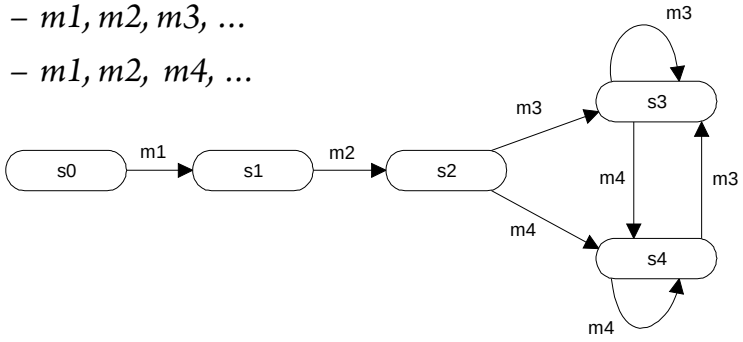
- Todos los estados: *m1, m2, m3, m4*
- Todas las transiciones: *m1, m2, m3, m4, m4, m4, m3, m3; m1, m2, m4*



17

Criterios de cobertura para máquinas de estado (II)

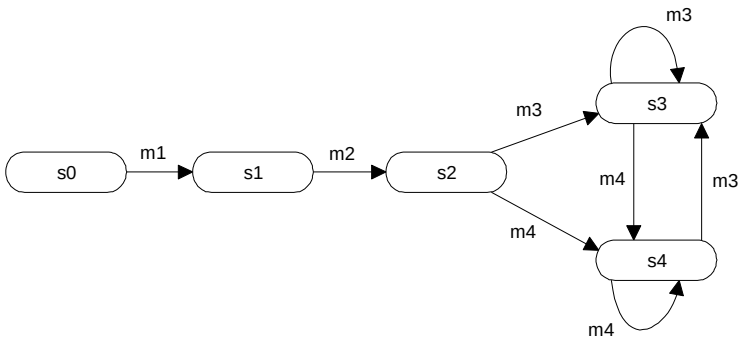
- Todos los pares de transiciones E/S de cada estado. Para *s2*, *k(m2, m3)* y *(m2, m4)*; para *s3*, debemos recorrer *(m3, m3)* y *(m3, m4)*; para *s4*, *(m4, m4)* y *(m4, m3)*. Por tanto:
  - *m1, m2, m3, …*
  - *m1, m2, m4, …*



18

Criterios de cobertura para máquinas de estado (III)

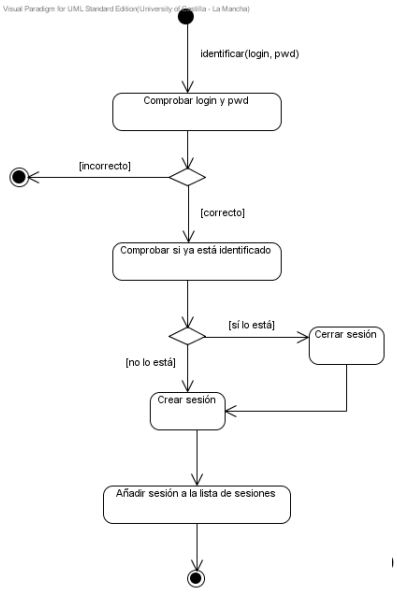
- Cobertura de *secuencia completa* (secuencias que el tester considera interesantes)
  - Por ejemplo: *m1, m2, m3, m3, m3, m3, m4*



19

Criterios de cobertura para máquinas de estado (y IV)

- Las máquinas de estado tienen dos usos principales en UML:
  - Describir comportamiento de instancias de clases
  - Describir pasos en la ejecución de casos de uso: cada estado se considera un paso en la ejecución. En la figura, se está describiendo un requisito
- En general: alta cobertura a nivel de artefactos de alto nivel supone alta cobertura en código fuente



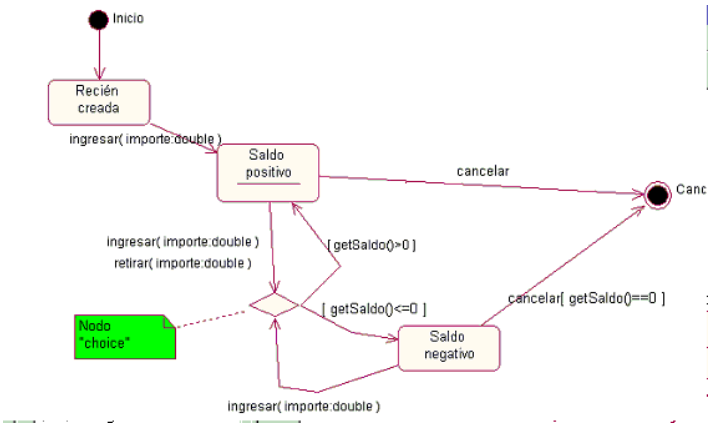
Cobertura CRUD (I)

- Para cada elemento persistente, se prueban todas sus operaciones *CRUD*
- Cada caso de prueba comienza por una *C*, seguida por todas las *U* y se termina por una *D*
- Tras cada *C*, *U* o *D*, se ejecuta una *R* que nos sirve de *oráculo*

21

Cobertura CRUD (y II)

- new·[[ingresar|retirar]·getSaldo]*·cancelar·getSaldo



22

Valores de prueba

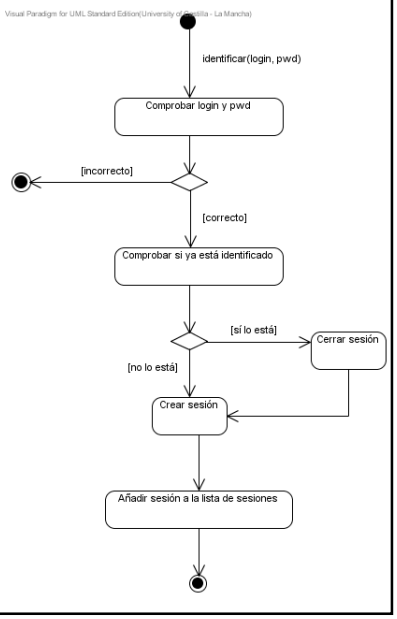
- Los valores de prueba (*test data*) son aquellos valores que el tester considera que son interesantes para probar el SUT
- Pueden proceder de técnicas como particiones de equivalencia, valores límite o conjetura de errores, del conocimiento, experiencia, etc. que el tester tenga del SUT o de otros proyectos, de la experiencia que nos transmite el usuario, etc.

Particiones de equivalencia (I)

- Para cada parámetro, se divide su dominio de entrada en conjuntos disjuntos (clases de equivalencia), asumiéndose que el SUT se comportará igual para cualquier elemento de la clase
- Igualmente, puede pensarse desde el punto de vista de la salida

Particiones de equivalencia (y II)

- Para *identificar(login, pwd)*:
  - *login* existente, *login* inexistente, etc.
  - *password* correspondiente al login, *password* no correspondiente, etc.
- Pero ojo, podemos aplicar otras técnicas de *perturbación de datos,* inyectar SQL, etc*:*
  - *login* inexistente AND *true*
  - …



Valores límite

- Para dar un préstamo hay tres categorías: menores de 18 años, de 18 a 55 y mayores de 55:
  - Teníamos 3 clases de eq.: 0-17, 18-55, >=56
  - En este ejemplo, el intervalo conflictivo es [18,55]. Por tanto, podemos usar el *valor exactamente en el límite*, el valor *inmediatamente superior* y el valor *inmediatamente inferior*: 17, 18, 19; 54, 55, 56.
- En la *variante ligera,* se toman sólo dos valores: el propio límite y los valores adyacentes de las clases de equivalencia adyacentes: 17, 18; 55, 56
- Recordar que todo esto es aplicable no sólo a los datos de entrada, sino que puede pensarse desde el punto de vista de la salida

Criterios de cobertura para valores (I)

- Una vez que, para cada parámetro, se han identificado sus *valores interesantes*, debemos combinarlos para construir casos de prueba.
- Existen multitud de estrategias de combinación. En general, todas ellas persiguen obtener *test suites* reducidos pero que alcancen alta cobertura en el SUT.
- Con los *criterios de cobertura para valores* intentamos conocer cuantitativamente el grado en que estamos utilizando los valores de prueba

Criterios de cobertura para valores (II)

- Cada valor de prueba debería utilizarse al menos una vez
- ¿Lo utilizamos una vez y ya? ¿Dos veces? ¿Tres?...
- Bueno, pues depende…
- Criterios:
  - *Each use* o *1-wise*
  - *Pairwise*
  - *N-wise*

Criterios de cobertura para valores: *Each use* (III)

- Utilizamos cada valor interesante al menos una vez en un caso de prueba

	A	B	C	D	E	F
0	Ludo	Dice	Person	2	Visa	Quiz
1	Trivial	null	Computer	3	Master card	-
2	Checkers				American express	
3	Chess					

- Para este sistema y con este criterio, bastarían cuatro casos de prueba:
  - *{Trivial, null, Person, 3, Master card, Quiz}*
  - *{Checkers, Dice, Computer, 2, Visa, -}*
  - *{Chess, Dice, Computer, 2, American express, Quiz}*
  - *{Ludo, null, Person, 2, Master card, -}*

29

Criterios de cobertura para valores: *Pairwise* (IV)

- Utilizamos cada par de valores interesantes al menos una vez en un caso de prueba: *Ludo* con *Dice, Ludo* con *null, Ludo* con *Person, Ludo* con *Computer,* ... pero también *Dice* con *Person, Dice* con *Computer,* etc.
- Se basa en la idea de que muchos errores aparecen cuando interactúan dos ciertos valores de dos parámetros

	A	B	C	D	E	F
0	Ludo	Dice	Person	2	Visa	Quiz
1	Trivial	null	Computer	3	Master card	-
2	Checkers				American express	
3	Chess					

Criterios de cobertura para valores: *Pairwise* (V)

- Construimos las tablas de pares, proponemos casos de prueba y vamos marcando los pares visitados. Terminamos cuando se han visitado todos al menos una vez.

8 pairs in (0, 1)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0
(2, 0)	0
(2, 1)	0
(3, 0)	0
(3, 1)	0

8 pairs in (0, 2)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0
(2, 0)	0
(2, 1)	0
(3, 0)	0
(3, 1)	0

8 pairs in (0, 3)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0
(2, 0)	0
(2, 1)	0
(3, 0)	0
(3, 1)	0

12 pairs in (0, 4)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(0, 2)	0
(1, 0)	0
(1, 1)	0
(1, 2)	0
(2, 0)	0
(2, 1)	0
(2, 2)	0
(3, 0)	0
(3, 1)	0
(3, 2)	0

8 pairs in (0, 5)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0
(2, 0)	0
(2, 1)	0
(3, 0)	0
(3, 1)	0

4 pairs in (1, 3)	
Elements	# of visits
(0, 0)	0
(0, 1)	0

4 pairs in (1, 2)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0

6 pairs in (1, 4)	
Elements	# of visits
(0, 0)	0
(0, 1)	0
(0, 2)	0
(1, 0)	0
(1, 1)	0
(1, 2)	0

- Hay $n \cdot (n-1)/2$ tablas de pares, siendo n el nº de parámetros: $6 \cdot 5/2 = 15$

31

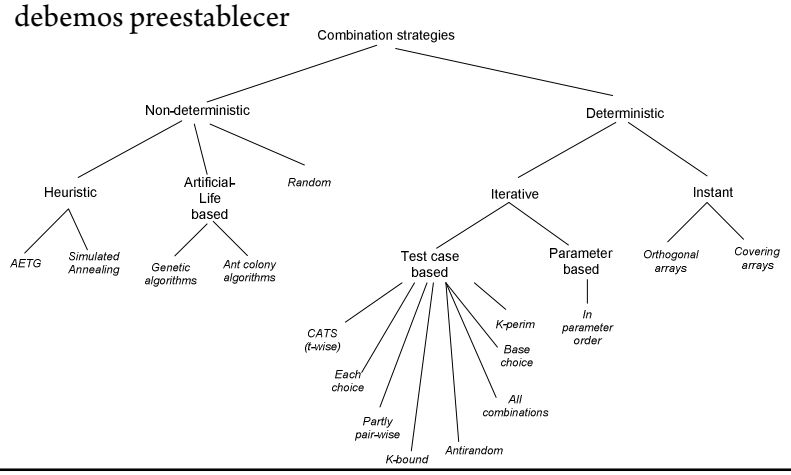
Criterios de cobertura para valores: *n-wise* (y VI)

- En este caso no es que interese probar con todos los pares, sino con todas las ternas (*3-wise*), cuartetos (*4-wise*), quintetos (*5-wise*), etc.

32

Estrategias de combinación para casos de prueba

- Ya que conocemos los criterios de cobertura para valores de prueba, necesitamos combinarlos de manera que los valores de nuestros test suites alcancen la cobertura de valores deseada, que también debemos preestablecer



Estructura de un caso de prueba

- Antes de continuar, debemos conocer la estructura de un caso de prueba:
  - Especificación de las condiciones iniciales
  - Ejecución de servicios
  - Determinación del resultado de la prueba (oráculo), de manera que conozcamos si lo obtenido se corresponde con lo esperado
- La escritura del oráculo es habitualmente un proceso manual (su generación automática es un intenso tema de investigación). Por ello interesa tener test suites buenos pero que no sean demasiado grandes.

All combinations (I)

- Se trata de determinar el producto cartesiano de todos los valores de todos los parámetros
  - Supongamos: A={1, 2, 3, 4}, B={5, 6, 7}, C={8, 9}
  - El número de combinaciones (cardinal del producto cartesiano) es |A|·|B|·|C|=4·3·2=24

i	A*i*	B*i*	C*i*	Índices
0	1	5	8	{0, 0, 0}
1	1	5	9	{0, 0, 1}
2	1	6	8	{0, 1, 0}
3	1	6	9	{0, 1, 1}
4	1	7	8	{0, 2, 0}
5	1	7	9	{0, 2, 1}
6	2	5	8	{1, 0, 0}
7	2	5	9	{1, 0, 1}
8	2	6	8	{1, 1, 0}
9	2	6	9	{1, 1, 1}
10	2	7	8	{1, 2, 0}
11	2	7	9	{1, 2, 1}
12	3	5	8	{2, 0, 0}
13	3	5	9	{2, 0, 1}
14	3	6	8	{2, 1, 0}
15	3	6	9	{2, 1, 1}
16	3	7	8	{2, 2, 0}
17	3	7	9	{2, 2, 1}
18	4	5	8	{3, 0, 0}
19	4	5	9	{3, 0, 1}
20	4	6	8	{3, 1, 0}
21	4	6	9	{3, 1, 1}
22	4	7	8	{3, 2, 0}
23	4	7	9	{3, 2, 1}

35

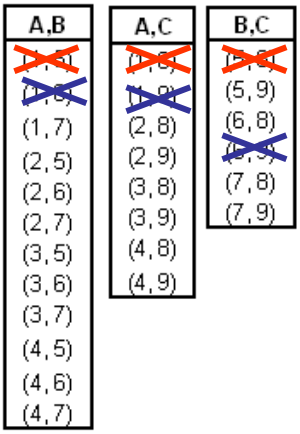
All combinations (y II)

- Obviamente, requerimos la escritura de 24 oráculos, lo que supone el seguimiento del programa para conocer su comportamiento en esas 24 situaciones, con el análisis, seguimiento, etcétera, que todo ello conlleva
- Además, muchos de estos casos de prueba serán redundantes, en el sentido de que recorrerán las mismas porciones del SUT: en otras palabras, habrá casos que no incrementen la cobertura de sentencias, MC/DC… del SUT
- Por ello, la estrategia AC se utiliza, en investigación, como *baseline*, como elemento de comparación
- Bien es cierto que, para un mismo conjunto de datos de prueba, AC es la estrategia que alcanza mayor cobertura, pues cualquier caso obtenido con otra estrategia está contenido en los casos producidos por AC

36

Un algoritmo básico de *pairwise*

- Teníamos: A={1, 2, 3, 4}, B={5, 6, 7}, C={8, 9}



- Tomamos un par no visitado (p. ej., *(1,5)*)y, a partir de él, escribimos un caso: (1, 5, 8)
- Y marcamos los pares (1, 5), (1, 8) y (5,8)
- Tomamos (1, 6) de (A,B), un par compatible en (A, C) y completamos con (B, C): (1, 6, 9). Marcamos los pares.
- …y así continuamos

37

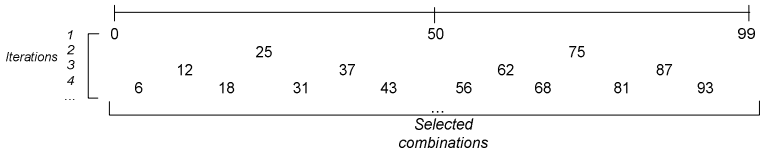
Antirandom

- Partiendo de un caso de prueba cualquiera, *Antirandom* se basa en la selección de los casos de prueba que sean *más diferentes* a los que ya están elegidos
- La diferencia entre dos casos de prueba se corresponde con la *distancia Hamming* (número de bits diferentes)
- La diferencia entre un caso de prueba *t* y los casos de prueba en un *TS* es la suma de las distancias Hamming a cada caso
- En caso de empate, se toma la *distancia Cartesiana* (suma de las raíces cuadradas de las distancias Hamming)
- Coste elevadísimo, porque en cada iteración se recalculan las distancias

38

Comb (peine)

- Inspirado en Antirandom, ordena todos los posibles casos de prueba (su número es conocido, recuérdese *All combinations*), y:
  - Selecciona el primero
  - Selecciona el último
  - Selecciona el del medio
  - Selecciona los dos centrales
  - etcétera



39

AETG (*Automatic Efficient Test Generator*) (I)

- Genera cobertura pairwise a un coste razonable
  - Se construyen las tablas de pares
  - Se añade la primera combinación (la número 0)
  - Se actualizan las tablas de pares
  - Mientras queden pares sin visitar:
    - Se crea una nueva combinación *c*, colocándose en la posición que corresponda el valor que visita más pares
    - Se completa *c* con el resto de valores, de tal manera que se visite el mayor número de pares
    - Se añade *c*
    - Se actualizan las tablas de pares

40

AETG (*Automatic Efficient Test Generator*) (II)

- Por ejemplo, para $A=\{1, 2, 3\}$, $B=\{4, 5\}$, $C=\{6, 7\}$
- Elegimos (1, 4, 6)
- Actualizamos las tablas:

6 pairs in (A, B)	
Elements	**# of visits**
(1, 4)	1
(1, 5)	0
(2, 4)	0
(2, 5)	0
(3, 4)	0
(3, 5)	0

6 pairs in (A, C)	
Elements	**# of visits**
(1, 6)	1
(1, 7)	0
(2, 6)	0
(2, 7)	0
(3, 6)	0
(3, 7)	0

4 pairs in (B, C)	
Elements	**# of visits**
(4, 6)	1
(4, 7)	0
(5, 6)	0
(5, 7)	0

41

AETG (*Automatic Efficient Test Generator*) (III)

- Creamos una nueva combinación. De los tres conjuntos, el 5 es el que visita más pares no visitados, así que elegimos (-, 5, -)

	6 pairs in (A, B)	
	Elements	**# of visits**
	(1, 4)	1
→	(1, 5)	0
	(2, 4)	0
→	(2, 5)	0
	(3, 4)	0
→	(3, 5)	0

6 pairs in (A, C)		
Elements	**# of visits**	
(1, 6)	1	
(1, 7)	0	
(2, 6)	0	
(2, 7)	0	→
(3, 6)	0	→
(3, 7)	0	

4 pairs in (B, C)	
Elements	**# of visits**
(4, 6)	1
(4, 7)	0
(5, 6)	0
(5, 7)	0

42

AETG (*Automatic Efficient Test Generator*) (IV)

- Debemos completar (-, 5, -) con un valor de $A=\{1, 2, 3\}$ y otro de $C=\{6, 7\}$. Y, claro, los valores que elijamos serán aquellos que visiten más pares no visitados, teniendo en cuenta que ya tenemos elegido el *5*.
- Respecto de *A*, consideramos *(1, 5)*, *(2, 5)* y *(3, 5)*. Podemos quedarnos con cualquiera, así que nos quedamos con el *1*.

6 pairs in (A, B)	
Elements	**# of visits**
(1, 4)	1
(1, 5)	0
(2, 4)	0
(2, 5)	0
(3, 4)	0
(3, 5)	0

6 pairs in (A, C)	
Elements	**# of visits**
(1, 6)	1
(1, 7)	0
(2, 6)	0
(2, 7)	0
(3, 6)	0
(3, 7)	0

4 pairs in (B, C)	
Elements	**# of visits**
(4, 6)	1
(4, 7)	0
(5, 6)	0
(5, 7)	0

43

AETG (*Automatic Efficient Test Generator*) (y V)

- Ya tenemos (1, 5, -). Completemos ahora con un valor compatible de $C=\{6, 7\}$.
- Tenemos los pares *(1, 6)* y *(1, 7)* como compatibles, pero el *(1,6)* ya se visitó. Así pues, nos quedamos con el 7 y completamos así:

$$(1, 5, 7)$$

- …y así continuamos hasta visitar todos los pares.

6 pairs in (A, B)	
Elements	**# of visits**
(1, 4)	1
(1, 5)	0
(2, 4)	0
(2, 5)	0
(3, 4)	0
(3, 5)	0

6 pairs in (A, C)	
Elements	**# of visits**
(1, 6)	1
(1, 7)	0
(2, 6)	0
(2, 7)	0
(3, 6)	0
(3, 7)	0

4 pairs in (B, C)	
Elements	**# of visits**
(4, 6)	1
(4, 7)	0
(5, 6)	0
(5, 7)	0

44

Weighted AETG (I)

- Se trata de un AETG con eliminación de pares y asignación de pesos, desarrollado por Beatriz Pérez Lamancha, del INCO/CES
- Permite eliminar aquellos pares no interesantes, y asignar pesos para que se introduzcan más veces los que interesen especialmente

45

Weighted AETG (II)

- Se trata de un AETG con eliminación de pares y asignación de *factores de selección*, desarrollado por Beatriz Pérez, del INCO/CES
- Permite eliminar aquellos pares no interesantes, y asignar un *factor de selección* para que se introduzcan más veces los pares que interesen especialmente

	A	B	C	D	E	F
0	Ludo	Dice	Person	2	Visa	Quiz
1	Trivial	null	Computer	3	Master card	-
2	Checkers				American express	
3	Chess					

46

Weighted AETG (y III)

8 pairs in (0, 1)		
Elements	Remove	Sel. factor
(Trivial, null)	☑	0
(Trivial, Dice)	☐	0
(Ludo, null)	☑	0
(Ludo, Dice)	☐	0
(Chess, null)	☐	0
(Chess, Dice)	☑	0
(Checkers, null)	☐	0
(Checkers, Dice)	☑	0

8 pairs in (0, 2)		
Elements	Remove	Sel. factor
(Trivial, Computer)	☐	2
(Trivial, Person)	☐	0
(Ludo, Computer)	☐	2
(Ludo, Person)	☐	0
(Chess, Computer)	☐	2
(Chess, Person)	☐	0
(Checkers, Computer)	☐	2
(Checkers, Person)	☐	0

8 pairs in (0, 3)		
Elements	Remove	Sel. factor
(Trivial, 3)	☐	0
(Trivial, 2)	☐	0
(Ludo, 3)	☐	0
(Ludo, 2)	☐	0
(Chess, 3)	☑	0
(Chess, 2)	☐	0
(Checkers, 3)	☑	0
(Checkers, 2)	☐	0

12 pairs in (0, 4)		
Elements	Remove	Sel. factor
(Trivial, Master card)	☐	0
(Trivial, Visa)	☐	0
(Trivial, American express)	☐	0
(Ludo, Master card)	☐	0
(Ludo, Visa)	☐	0
(Ludo, American express)	☐	0
(Chess, Master card)	☐	0
(Chess, Visa)	☐	0

8 pairs in (0, 5)		
Elements	Remove	Sel. factor
(Trivial, -)	☑	0
(Trivial, Quiz)	☐	0
(Ludo, -)	☐	0
(Ludo, Quiz)	☑	0
(Chess, -)	☐	0
(Chess, Quiz)	☑	0

4 pairs in (1, 2)		
Elements	Remove	Sel. factor
(null, Computer)	☐	0
(null, Person)	☐	0
(Dice, Computer)	☐	0
(Dice, Person)	☐	0

47

Customized pairwise

- Permite eliminar los pares no interesantes
- Realmente, es un *All combinations* al que le quitamos los pares que no nos hacen falta
- El problema de este algoritmo es su coste, que es exponencial, porque es exponencial (necesariamente) el *All combinations*

48

Algoritmos evolutivos (I)

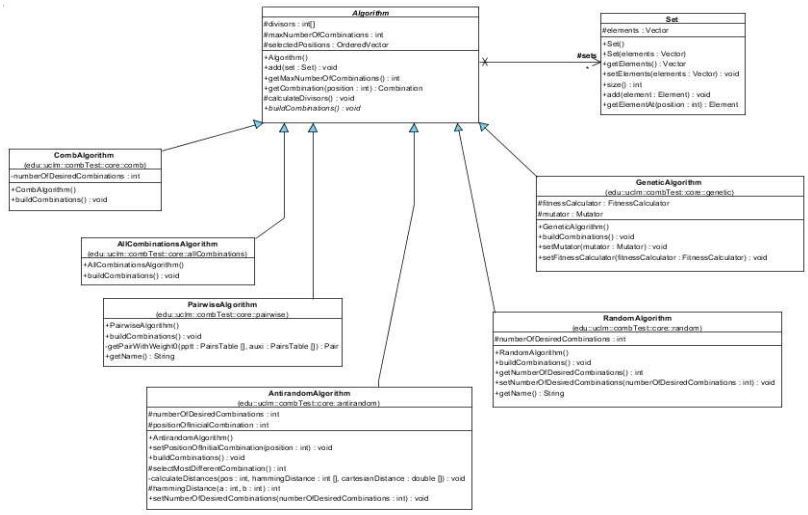
- A partir de una población inicial de casos de prueba, generan progresivamente poblaciones mejores en función del *fitness* de los individuos
- El *fitness* es una medida de la calidad de cada caso, y puede calcularse en función de… prácticamente cualquier cosa
- Una posible forma de calcular el *fitness* es mediante el cálculo del número de pares visitados

Algoritmos evolutivos (y II)

1. To decide the number of individuals composing the initial population.
2. To decide the number of individuals composing the elite.
3. Loop while the fitness of the population does not reach a certain threshold of goodness and not exceeding a number of iterations.
    1. To generate the elite from the population.
    2. To generate pmating from the population.
    3. To evaluate the elite, to put the best individuals in the result, and to mutate the worst individuals.
    4. To cross the individuals in pmating in order to obtain a new population.
    5. If the new population has not a minimum number of individuals then add the elite to the population.
    6. If the new population has not a minimum number of individuals then add several individuals from the previous population.
4. To show the results.

Un poco de práctica con CTWeb

- http://alarcosj.esi.uclm.es/CTWeb/



51

Técnicas para generación de casos de prueba

- Existen tres enfoques principales:
  - Generación de casos a partir de código para alcanzar un nivel dado de cobertura
  - Desarrollar algoritmos para generar casos a partir del conocimiento del *tester*
  - Generación a partir de especificaciones formales o semiformales (modelos)
- Veremos tres técnicas:
  - A partir de máquinas de estado
  - A partir de expresiones regulares
  - A partir de requisitos descritos en forma de tablas de decisión

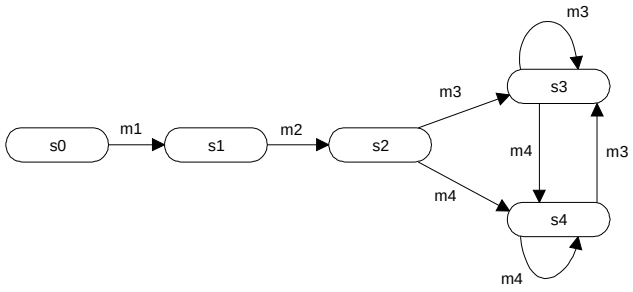
52

Obtención de casos a partir de máquinas de estado

- Se trata, simplemente, de proponer casos de prueba con el objetivo de cubrir la máquina de estados respecto de alguno de sus criterios de cobertura que, recordando, eran:
  - Estados
  - Transiciones
  - Pares de E/S en transiciones
  - Secuencia completa (secuencias *interesantes*)

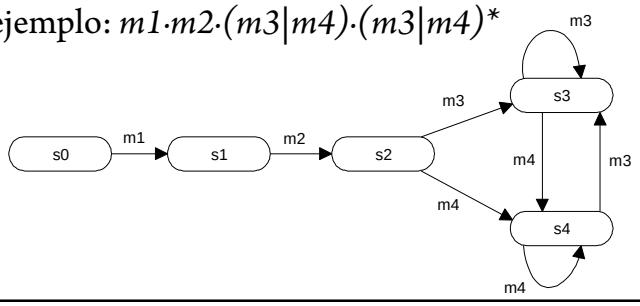
Generación de casos a partir de expresiones regulares (I)

- Una máquina de estados puede entenderse como un autómata finito, el cual describe una expresión regular
- La expresión regular estará formada por las operaciones que disparan las transiciones entre estados
- Por ejemplo: $m1 \cdot m2 \cdot (m3|m4) \cdot (m3|m4)^*$



Generación de casos a partir de expresiones regulares (II)

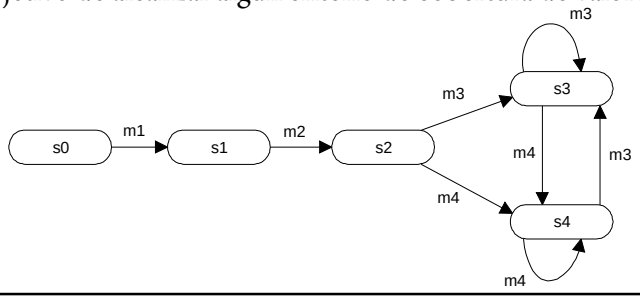
- Lo único que haremos, por tanto, será expandir la expresión regular para obtener palabras del *lenguaje* de hasta una determinada longitud
- Cada palabra se corresponde con una *test template* (plantilla de prueba), que después combinamos con *valores de prueba*
- Por ejemplo: *m1·m2·(m3|m4)·(m3|m4)**



55

Generación de casos a partir de expresiones regulares (y III)

- Si la signatura de los métodos es, por ejemplo, *m1(int, int), m2(String), m3(double, Project), m4()*, una posible plantilla de prueba sería:
  - *m1(int, int)·m2(String)·m3(double, Project)·m4()*
  - A partir de esa *template*, asignamos valores a los parámetros de acuerdo con una de las estrategias de combinación y con el objetivo de alcanzar algún criterio de cobertura de valores



56

Generación de casos a partir de requisitos descritos en forma de tablas de decisión (I)

- Una tabla de decisión muestra un conjunto de condiciones, reglas y consecuencias (acciones o *effects)*, las cuales deben verificarse en función de que se verifiquen o no las reglas

Generación de casos a partir de requisitos descritos en forma de tablas de decisión (II)

- Supongamos un sistema bancario en el que hay una función de retirada de efectivo: se hace una comprobación del saldo de la cuenta y del crédito concedido a dicha cuenta. La función conoce el importe que se desea retirar, el saldo de la cuenta y el crédito concedido (el crédito es un número negativo, que representa el saldo negativo que se autoriza alcanzar a este cliente). Las posibles acciones son las siguientes:
  - Autorizar la retirada del efectivo y enviar una carta.
  - Autorizar la retirada del efectivo.
  - Suspender la cuenta y enviar una carta.
- Además:
  - Si saldo-importe>=0, se autoriza la retirada de efectivo.
  - Si saldo-importe<0 ^ saldo-importe>=crédito, se autoriza la retirada y se envía una carta.
  - Si saldo-importe<0 ^ saldo-importe<crédito, se suspende la cuenta y se envía una carta

Generación de casos a partir de requisitos descritos en forma de tablas de decisión (y III)

- Consecuencias:
  - E1: autorizar la retirada de efectivo.
  - E2: suspender la cuenta.
  - E3: enviar la carta.
- Condiciones:
  - C1: *saldo-importe>=0*
  - C2: *saldo-importe>=crédito*

	Reglas			
	1	2	3	4
C1: *saldo-importe>=0*	0	0	1	1
C2: *saldo-importe>=crédito*	0	1	0	1
E1: autorizar retirada	0	1	-	1
E2: suspender la cuenta	1	0	-	0
E3: enviar la carta	1	1	-	0

y… ya está. Podemos aplicar aquí criterios de cobertura de decisiones, condiciones, MC/DC, etc.

59