

Resumen 2do Parcial Ingeniería de Software 1

Juan Vanecek

Junio 2015

Índice

1. Finite State Machine (FSM)	2
1.1. Máquina de Estado	2
1.2. LTS	2
2. Testing Funcional (Caja Negra)	4
3. Testing Estructural (Caja Blanca)	5

1. Finite State Machine (FSM)

1.1. Máquina de Estado

Las *Máquinas de Estado* denotan una familia de notaciones que tienen: (a) Estados; (b) Transiciones; y (c) Etiquetas .

Por ejemplo: Statecharts, FSM, LTS, Autómatas, Timed Autómatas, etc.

1.2. LTS

Definición 1 (LTS)

Sea *Estados* el universo de estados, *Act* el universo de acciones observables, y $Act_\tau = Act \cup \{\tau\}$ (τ es la acción no observable o silenciosa).

Un *LTS* es una tupla $P = \langle S, L, \Delta, s_0 \rangle$, donde $S \subseteq Estados$ es un conjunto finito, $L \subseteq Act_\tau$ es un conjunto de etiquetas, $\Delta \subseteq (S \times L \times S)$ es un conjunto de transiciones etiquetadas, y $s_0 \in S$ es el estado inicial.

Definimos el alfabeto de comunicaciones de P como $\alpha(P) = L \setminus \{\tau\}$.

Ej: $FSM_A = \langle \{1, 2, 3\}, \{a, d, c\}, \{(1, a, 2), (2, d, 3), (3, c, 1)\}, 1 \rangle$

Definición 2 (Ejecución)

Sea $P = \langle S, L, \Delta, s_0 \rangle$ una LTS. Una ejecución de P es una secuencia $w = q_0 l_0 q_1 l_1 \dots$ donde para todo $i \in [0, w/2)$, q_i es un estado y $l_i \in L$ tal que $q_0 = q$ y $(q_i, l_i, q_{i+1}) \in \Delta$.

Ej: Las ejecuciones de FSM_A son $(1, a, 2)$, $(1, a, 2)(2, d, 3)$ y $(1, a, 2)(2, d, 3)(3, c, 1)$

Definición 3 (Proyección)

Sea w una palabra $w_0 w_1 w_2 \dots$ y A un alfabeto. La proyección de w en A , que se denota como $w|_A$, es el resultado de eliminar de w todos los w_i que no están en A .

Definición 4 (Trazas)

Sea P una LTS. Una palabra w del alfabeto $\alpha(P)$ es una traza de P si hay una ejecución w' de P tal que $w = w'|_{\alpha(P)}$. Notar que las trazas no incluye acciones τ .

También se define $tr(P) = \{w | w \text{ es traza de } P\}$

Definición 5 (Transitar)

Una LTS $P = \langle S, L, \Delta, q \rangle$ transita con una etiqueta l a una LTS P' si $P' = \langle S, L, \Delta, q' \rangle$ y $(q, l, q') \in \Delta$.

Notación: $P \xrightarrow{l} P'$

Definición 6 (Composición Paralela)

Sean P_1 y P_2 LTSs, con $P_i = \langle S_i, L_i, \Delta_i, q_i \rangle$. La composición paralela $P_1 || P_2$ es una LTS $\langle S, L, \Delta, (q_1, q_2) \rangle$ con $S = S_1 \times S_2$, $L = L_1 \times L_2$ y $\Delta \subseteq (S \times \Delta_1 \cup \Delta_2 \times S)$ formada por:

- $((s, t), a, (s', t)) \in \Delta$, si $(s, a, s') \in \Delta_1$, $a \in \alpha(P_1)$ y $a \notin \alpha(P_2)$.
- $((s, t), a, (s, t')) \in \Delta$, si $(t, a, t') \in \Delta_2$, $a \notin \alpha(P_1)$ y $a \in \alpha(P_2)$.

- $((s, t), a, (s', t')) \in \Delta$, si $(s, a, s') \in \Delta_1$, $(t, a, t') \in \Delta_2$, $a \in \alpha(P_1) \cap \alpha(P_2)$

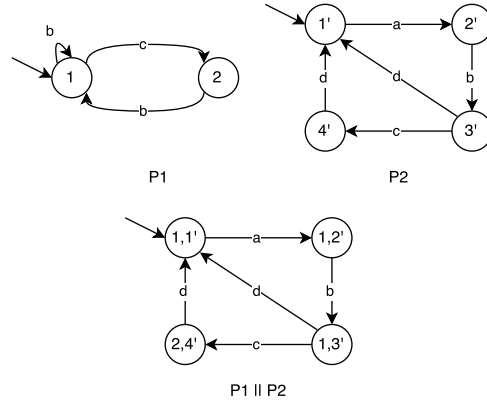


Figura 1: Ejemplo de composición 1

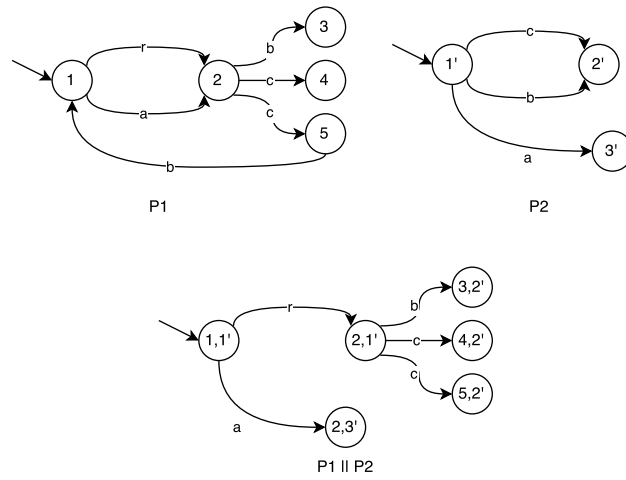


Figura 2: Ejemplo de composición 2

Propiedad 1 (Leyes algebraicas)

En la composición en paralelo valen las leyes algebraicas:

- **Conmutatividad:** $P || Q = Q || P$
- **Asociatividad:** $P || (Q || R) = (P || Q) || R = P || Q || R$

2. Testing Funcional (Caja Negra)

Para el testing funcional hay que seguir los pasos:

1. Elegir la **funcionalidad** a testear (de todas las que vengan con los requerimientos).
2. Determinar los **factores** (parámetros).
3. Determinar las **categorías** (lo que yo quiero observar de los factores).
4. Determinar las **elecciones** (particionar el dominio de los factores según lo que se quiera testear).
5. Poner las **características** (“Error” o “Único”)

3. Testing Estructural (Caja Blanca)

Sirve para testear programas secuenciales, con un único punto de ingreso y un único punto de terminación.

Una ejecución del programa que termina satisfactoriamente, está asociada a un **camino completo** en el flowgraph del programa (va desde el nodo inicial hasta el nodo asociado a la terminación).

Un **camino no factible** es aquel que no puede ser realizado porque es lógicamente imposible. Proceso para hacer un test estructural:

1. Con el código como base, dibujamos el grafo de flujo de control
2. Determinamos el conjunto de todos los caminos completos.
3. Preparamos los datos de test que forzarán la ejecución de cada camino. Para ello hay que conseguir conjunto de valores de entrada tal que:
 - (a) Cubra todas las sentencias (nodos del flowgraph).
 - (b) Cubra todos los branches (ejes del flowgraph).
 - (c) Cubra todos los casos (True y False) de las condiciones.
 - (d) Cubra todas las duas (terna $[d, u, x]$ tal que: d es un nodo que tiene la definición de la variable x , u es un nodo o arco en donde se usa x , y hay al menos un camino desde d hasta u que no contiene otra definición de x).
 - (e) Cubra todos los demás caminos que los incisos anteriores no cubren.
4. Evaluamos que cada test cumpla con los criterios.
5. Eventualmente, iteramos hasta terminar con todos los tests hasta tener cubiertos todos los caminos.

Lo que buscamos testear acá son todos los caminos que puede tomar un programa. Para ello hay que graficar el CFG, un grafo de flujo o *flowgraph* que representa el flujo de control de un programa.

Cuadro 1: Ejemplo de código y CFG asociado

```

1 void calculo(int x, int y) {
2   int w = 0;
3   if( x%2 == 0 ){
4     w = 21;
5   }else{
6     w = 31;
7   }
8   int z = x + y;
9   if( z==0 || z==1 ){
10    while( z<w ){
11      z = z+10;
12    }
13  }
14  if( z!=30 ){
15    print( "no llegamos a 30" );
16  }
17 }

```

