IDENTIFICACIÓN DE CUELLOS DE BOTELLA EN CIRCUITOS ASÍNCRONOS

(ANEXOS)

Autor: Luis Jesús Jorge Salcines
Director: Jorge Emilio Júlvez Bueno

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Zaragoza, mayo de 2010

Tabla de contenido

I. Anexos 3
I.1. Código fuente de todos los programas 3
I.1.a. readgr.m 3
I.1.b. simularArchivo.m 4
I.1.c. simular.m 5
I.1.d. avanzarTransicion.m 6
I.1.e. calculoCuellos.m 7
I.1.f. calculoCuellosConRatio.m 9
I.1.g. eleccionDelta.m 11
I.1.h. optimizacion.m 12
I.1.i. analisisCompleto.m 13
I.1.j. optimCalculoThroughput.m 15
I.1.k. simplificarCircuito.m 16
I.1.l. analisisMarcado.m 17

I. Anexos

I.1. Código fuente de todos los programas

En los próximos subapartados se muestra una copia de los códigos fuente de todos los programas, que igualmente pueden encontrarse tanto en PDF como en formato texto-Matlab dentro del CD de esta memoria.

I.1.a. readgr.m

Programa que lee un archivo de texto con el modelo de una Red de Petri y lo transforma en sus cuatro matrices características Pre, Post, M_i y Delta.

```
1 function [Pre, Post, M0, delta] = readgr(filename)
2 % This function reads a file 'filename' containing a marked graph
3 % with variable delay transitions.
4
5 % It returns:
6 % - the Pre and Post incidence matrices
7 % - the initial marking M0
8 % - the delays (and the probability of each delay)
9
10 fid = fopen(filename,'r');
11
12 Pre=sparse([]);
13 Post=sparse([]);
14 M0=sparse([]);
15
16 % Obtain Pre, Post and M0
17 nps=0; % Number of places
18 ta=fscanf(fid,'%d',1);
19 while ta>0
20 nps=nps+1;
21 tb=fscanf(fid,'%d',1);
22 Post(nps,ta)=1;
23 Pre(nps,tb)=1;
24 M0(nps,1)=fscanf(fid,'%d',1);
25 ta=fscanf(fid,'%d',1);
26 end
27
28 nts=size(Pre,2); % Number of transitions
29
30 % Obtain delta
31 % 1st column: probability for first delay; 2nd column: first delay
32 % 3rd column: probability for second delay; 4th column: second delay
33
34 delta=fscanf(fid,'%f',[4 nts]);
35 delta=delta';
```

I.1.b. simularArchivo.m

Programa que simula una Red de Petri contenida en un archivo de texto, según los parámetros que el usuario facilite. Recurre a la función 'simular.m', contenida en el apartado siguiente.

```
1 function [Rendimientos, MediaVarianza, IntervaloRendimiento] =
simularArchivo(nombreArchivo, bucles, tiempo, pconfint)
2 %Función que ejecuta un sistema esquematizado como Red de Petri,a partir
3 %de un archivo
4 %Entradas: Nombre del archivo con el esquema de la red
5 % Número de ejecuciones
6 % Tiempo de cada ejecución
7 % Intervalo de confianza en tanto por 1
8 %Salidas: Matriz Rendimientos: throughputs de cada ejecución
9 % Matriz MediaVarianza: media y varianza de todas las ejecuciones
10 % Matriz IntervaloRendimiento: valores posibles del throughput
11 % para un intervalo de confianza
12 % dado (pconfint)
13
14 %Leer mapa inicial del circuito
15 [Pre, Post, Mi, Delta] = readgr(nombreArchivo);
16
17 %Simular el circuito
18 [Rendimientos, MediaVarianza, IntervaloRendimiento] = simular(Pre,
Post, Mi, Delta, bucles, tiempo, pconfint);
```

I.1.c. simular.m

Programa que simula una Red de Petri a partir de sus cuatro matrices características, en función de los parámetros que el usuario facilite. Recurre a la función 'avanzarTransicion.m', contenida en el apartado siguiente.

```
1 function [Rendimientos, MediaVarianza, IntervaloRendimiento] =
simular(Pre, Post, Mi, Delta, bucles, tiempo, pconfint)
2 %Función que ejecuta un sistema esquematizado como Red de Petri
3 %Entradas: Matrices Pre, Post, Mi y Delta con el esquema de la red
4 % Número de ejecuciones
5 % Tiempo de cada ejecución
6 % Intervalo de confianza en tanto por 1
7 %Salidas: Matriz Rendimientos: throughputs de cada ejecución
8 % Matriz MediaVarianza: media y varianza de todas las ejecuciones
9 % Matriz IntervaloRendimiento: valores posibles del throughput
10 % para un intervalo de confianza
11 % dado (pconfint)
12
13 %Inicialización de variables para resultados
14 %"Resultados" almacena el número de veces que se ha disparado cada
15 %transición en cada ciclo de ejecución. No se facilita el dato al acabar la
16 %simulación. Es simplemente un residuo de código para hacer pruebas
17 Resultados = [];
18 Rendimientos = [];
19
20 for i = 1 : bucles
21 %Matriz de tiempos de disparo
22 % Columna 1: tiempos remanentes o -1 si está deshabilitada
23 % Columna 2: veces que se ha ejecutado una transición
24 Transiciones = -1 * ones(size(Pre,2),1);
25 Transiciones = [Transiciones, zeros(size(Transiciones,1),1)];
26
27 tiempoEjecucion = 0;
28 while (tiempoEjecucion < tiempo)
29 %Mientras no se acabe el tiempo, se ejecuta una transición detrás
30 %de otra
31 [Mi, Transiciones, tdisparo] = avanzarTransicion(Pre, Post, Mi, Delta,
Transiciones);
32 if (tdisparo ~= -1)
33 tiempoEjecucion = tiempoEjecucion + tdisparo;
34 else
35 tiempoEjecucion = tiempo;
36 disp('No hay ninguna transición habilitada');
37 end
38 end
39
40 Resultados = [Resultados, Transiciones(:,2)];
41 %Se supone que, transcurrido un tiempo considerable de ejecución, todas
42 %las transiciones se habrán disparado un número similar de veces. Por
43 %ello, se toma sólo la primera de ellas como referencia
44 Rendimientos = [Rendimientos, (Transiciones(1,2)/tiempoEjecucion)];
45 end
46
47 %Obtención de medias y varianzas
48 Media = mean(Rendimientos,2);
49 Varianza = var(Rendimientos');
50 MediaVarianza = [Media, Varianza'];
51
52 %Rango de valores para un intervalo de confianza dado
53 confinter = icdf('t', 1-(1-pconfint)/2, bucles-1) * sqrt(Varianza/bucles);
54 IntervaloRendimiento = [Media - confinter, Media + confinter];
```

I.1.d. avanzarTransicion.m

Programa que dispara la transición indicada como parámetro de entrada, junto con las cuatro matrices características de la Red de Petri.

```
1 function [Mi, Transiciones, tdisparo] = avanzarTransicion(Pre, Post, Mi,
Delta, Transiciones)
2 %Función que avanza una transición en una Red de Petri. Se ejecuta como
3 %llamada de otra función
4 %Entradas: Mapa del circuito (Pre, Post, Mi, Delta)
5 % Matriz Transiciones según la función simular.m
6 %Salidas: Nuevo mapa del circuito (Mi)
7 % Nueva situación de las transiciones
8 % Tiempo de ejecución
9
10 %Comprobar qué transiciones están habilitadas
11 for i = 1 : size(Pre,2)
12 nodo = 0;
13 marcado = 0;
14 for j = 1 : size(Pre,1)
15 if (Pre(j,i) == 1)
16 nodo = nodo + 1;
17 if (Mi(j) > 0)
18 marcado = marcado + 1;
19 end
20 end
21 end
22
23 %Crear matriz con tiempos de disparo de transiciones habilitadas
24 if ((marcado == nodo) && (Transiciones(i,1) == -1))
25 aleatorio = rand;
26 if (Delta(i,1) >= aleatorio)
27 Transiciones(i,1) = Delta(i,2);
28 else
29 Transiciones(i,1) = Delta(i,4);
30 end
31 end
32 end
33
34 %Comprobar transición con menor tiempo de disparo
35 tdisparo = inf;
36 ndisparo = 0;
37 for i = 1 : size(Transiciones,1)
38 if ((Transiciones(i,1) < tdisparo) && (Transiciones(i,1) ~= -1))
39 tdisparo = Transiciones(i,1);
40 ndisparo = i;
41 end
42 end
43
44 %Iniciar si hay alguna transición habilitada
45 if (tdisparo ~= inf)
46 %Disparar transición más rápida
47 disparo = sparse(size(Transiciones,1),1);
48 disparo(ndisparo,1) = 1;
49 Mi = Mi + (Post-Pre) * disparo;
50
51 %Actualizar tiempos de disparo
52 for i = 1 : size(Transiciones,1)
53 if ((Transiciones(i,1) ~= -1) && (i ~= ndisparo))
54 Transiciones(i,1) = Transiciones(i,1) - tdisparo;
55 end
56 end
57 Transiciones(ndisparo,1) = -1;
58
59 %Actualizar número de veces
60 Transiciones(ndisparo,2) = Transiciones(ndisparo,2) + 1;
61 else
62 tdisparo = -1;
63 end
```

I.1.e. calculoCuellos.m

Obtiene sucesivos cuellos de botella cambiando los deltas de disparo de cada transición uno por uno, a partir del archivo que contenga una Red de Petri. Recurre a la función 'eleccionDelta.m', contenida en el apartado I.1.g.

```
function [yFinal,thoFinal,listaNodos,evolucionThroughput] = calculoCuellos(nombreArchivo)
%Función que calcula el cuello de botella por métodos de optimización
%Entradas: Nombre de archivo con el esquema de la red
%Salidas: Matriz yFinal: cada columna es el resultado con unas Deltas
% diferentes. Los valores distintos de cero
% representan el camino más lento (cuello de
% botella)
% Matriz thoFinal: cada columna es el throughput de cada
% optimización anterior
% Matriz evolucionThroughput: muestra la evolución del
% throughput en los sucesivos cambios
% de tiempo de disparo de las
% transiciones
% Matriz listaNodos: muestra los nodos que definen los cuellos de
% botella de los casos más desfavorable, más
% favorable y más probable, por columnas
% Gráfico con la evolución del throughput en los sucesivos
% cambios

yFinal = [];
thoFinal = [];
evolucionThroughput = [];

listaNodos = [];
contFilaListaNodos = 1;

listaTransicionesProb = [];
contFilaListaTransicionesProb = 1;

%Leer mapa inicial del circuito
[Pre, Post, Mi, Delta] = readgr(nombreArchivo);

%Para que funcione correctamente la simulación, es necesario adaptar la
%matriz de los deltas de disparo para el caso de las transiciones que
%tengan sólo un tiempo de disparo. Además, se elabora una lista con todas
%las transiciones que tienen distintos tiempos de disparo
for i = 1 : size(Delta,1)
if (Delta(i,3) == 0)
Delta(i,4) = Delta(i,2);
end

if (Delta(i,1) == 0)
Delta(i,2) = Delta(i,4);
end

if ((Delta(i,1) ~= 1) && (Delta(i,1) ~= 0))
listaTransicionesProb(contFilaListaTransicionesProb,1) = i;
contFilaListaTransicionesProb = contFilaListaTransicionesProb + 1;
end
end

%Problema a resolver
for i = 1 : 3
%Elección de distintos Deltas
if (i == 1)
D = eleccionDelta(Delta, 'DeltasMaximos');
sentencia = 'Nodos cuello de botella, deltas máximos :';
end
if (i == 2)
D = eleccionDelta(Delta, 'DeltasMinimos');
sentencia = 'Nodos cuello de botella, deltas mínimos :';
end
if (i == 3)
D = eleccionDelta(Delta, 'DeltasMasProbables');
sentencia = 'Nodos cuello de botella, deltas mas probables:';
end
```

```
67
68 %Con la matriz Delta obtenida de la funcion eleccionDelta se procede a
69 %optimizar el esquema
70 [y,tho] = optimizacion(Pre, Post, D, Mi);
71
72 yFinal = [yFinal, y];
73 thoFinal = [thoFinal, tho];
74
75 for j = 1 : size(y,1)
76 if (y(j,1) > 10e-8)
77 listaNodos(contFilaListaNodos,i) = j;
78 contFilaListaNodos = contFilaListaNodos + 1;
79 sentencia = [sentencia, ' ', int2str(j)];
80 end
81 end
82 contFilaListaNodos = 1;
83 disp(sentencia);
84 end
85
86 thoAntiguo = thoFinal(1,1);
87 evolucionThroughput(1,1) = thoAntiguo;
88 indiceEvolThro = 2;
89
90 %Volvemos al peor de los casos posibles
91 D = eleccionDelta(Delta, 'DeltasMaximos');
92
93 %Se optimiza el esquema para sucesivos cambios en los tiempos de
94 %transición
95 for i = 1 : size(listaTransicionesProb,1)
96 valorAntiguo = D(listaTransicionesProb(i,1),1);
97 if (Delta(listaTransicionesProb(i,1),2) > Delta(listaTransicionesProb(i,1),4))
98 D(listaTransicionesProb(i,1),1) = Delta(listaTransicionesProb(i,1),4);
99 else
100 D(listaTransicionesProb(i,1),1) = Delta(listaTransicionesProb(i,1),2);
101 end
102 valorNuevo = D(listaTransicionesProb(i,1),1);
103
104 [yNueva,thoNueva] = optimizacion(Pre, Post, D, Mi);
105 sentencia = ['Cambiando la transicion ', int2str(listaTransicionesProb(i,1))];
106 sentencia = [sentencia, ' de ', num2str(valorAntiguo), ' a '];
107 sentencia = [sentencia, num2str(valorNuevo), ' el throughput pasa de '];
108 sentencia = [sentencia, num2str(thoAntiguo,20), ' a ', num2str(thoNueva,20)];
109 disp(sentencia);
110 thoAntiguo = thoNueva;
111 evolucionThroughput(indiceEvolThro,1) = thoAntiguo;
112 indiceEvolThro = indiceEvolThro + 1;
113 end
114
115 plot(evolucionThroughput);
```

I.1.f. calculoCuellosConRatio.m

Programa que obtiene los cuellos de botella e indica el ratio de simplificación del circuito, a partir del archivo que contenga la Red de Petri. Recurre a la función 'eleccionDelta.m', contenida en el apartado siguiente.

```
function [yFinal,thoFinal,listaNodos] = calculoCuellosConRatio(nombreArchivo)
%Función que calcula el cuello de botella por métodos de optimización
%Entradas: Nombre de archivo con el esquema de la red
%Salidas: Matriz yFinal: cada columna es el resultado con unas Deltas
% diferentes. Los valores distintos de cero
% representan el camino más lento (cuello de
% botella)
% Matriz thoFinal: cada columna es el throughput de cada
% optimización anterior
% Matriz listaNodos: muestra los nodos que definen los cuellos de
% botella de los casos más desfavorable, más
% favorable y más probable, por columnas
% Muestra por pantalla cómo evoluciona el ratio de simplificación
% del circuito en cada iteración

yFinal = [];
thoFinal = [];

listaNodos = [];
contFilaListaNodos = 1;

listaTransicionesProb = [];
contFilaListaTransicionesProb = 1;

%Leer mapa inicial del circuito
[Pre, Post, Mi, Delta] = readgr(nombreArchivo);
ratioAntiguo = size(Mi,1);
for i = 1 : size(Delta,1)
if (Delta(i,3) == 0)
Delta(i,4) = Delta(i,2);
end

if (Delta(i,1) == 0)
Delta(i,2) = Delta(i,4);
end

if ((Delta(i,1) ~= 1) && (Delta(i,1) ~= 0))
listaTransicionesProb(contFilaListaTransicionesProb,1) = i;
contFilaListaTransicionesProb = contFilaListaTransicionesProb + 1;
end
end

%Problema a resolver
for i = 1 : 3
%Elección de distintos Deltas
if (i == 1)
D = eleccionDelta(Delta, 'DeltasMaximos');
sentencia = 'Nodos cuello de botella, deltas máximos :';
end
if (i == 2)
D = eleccionDelta(Delta, 'DeltasMinimos');
sentencia = 'Nodos cuello de botella, deltas mínimos :';
end
if (i == 3)
D = eleccionDelta(Delta, 'DeltasMasProbables');
sentencia = 'Nodos cuello de botella, deltas mas probables:';
end

[y,tho] = optimizacion(Pre, Post, D, Mi);

yFinal = [yFinal, y];
thoFinal = [thoFinal, tho];

for j = 1 : size(y,1)
if (y(j,1) > 10e-8)
listaNodos(contFilaListaNodos,i) = j;
contFilaListaNodos = contFilaListaNodos + 1;
```

```
68 sentencia = [sentencia, ' ', int2str(j)];
69 end
70 end
71 contFilaListaNodos = 1;
72 disp(sentencia);
73 end
74
75 thoAntiguo = thoFinal(1,1);
76
77 D = eleccionDelta(Delta, 'DeltasMaximos');
78 for i = 1 : size(listaTransicionesProb,1)
79 valorAntiguo = D(listaTransicionesProb(i,1),1);
80 if (Delta(listaTransicionesProb(i,1),2) > Delta(listaTransicionesProb(i,1),4))
81 D(listaTransicionesProb(i,1),1) = Delta(listaTransicionesProb(i,1),4);
82 else
83 D(listaTransicionesProb(i,1),1) = Delta(listaTransicionesProb(i,1),2);
84 end
85 valorNuevo = D(listaTransicionesProb(i,1),1);
86
87 [yNueva,thoNueva] = optimizacion(Pre, Post, D, Mi);
88 listaNodos = [];
89 contFilaListaNodos = 1;
90 for j = 1 : size(yNueva,1)
91 if (yNueva(j,1) > 10e-8)
92 listaNodos(contFilaListaNodos,1) = j;
93 contFilaListaNodos = contFilaListaNodos + 1;
94 sentencia = [sentencia, ' ', int2str(j)];
95 end
96 end
97 disp(sentencia);
98 sentencia = [];
99 disp(sentencia);
100
101 tablaCuellosBotella = [];
102 for j = 1 : size(listaNodos,1)
103 tablaCuellosBotella(listaNodos(j,1),1) = 1;
104 end
105
106 guardarPre = Pre;
107 guardarPost = Post;
108 guardarMi = Mi;
109 guardarDelta = Delta;
110 [Pre, Post, Mi, Delta] = simplificarCircuito(Pre, Post, Mi, Delta,
tablaCuellosBotella);
111 disp(['Relación de simplificación: ',num2str(100-100*size(Mi,1)
/ratioAntiguo),'%']);
112 Pre = guardarPre;
113 Post = guardarPost;
114 Mi = guardarMi;
115 Delta = guardarDelta;
116
117 thoAntiguo = thoNueva;
118 end
```

I.1.g. eleccionDelta.m

Programa que selecciona los deltas máximos, mínimos o más probables, según la elección del usuario o del programa que recurra a esta función.

```
1 function D = eleccionDelta(Delta, tipoEleccion)
2 %Función auxiliar. Se recurre a ella desde otras funciones
3 %Entradas: Matriz Delta: matriz con valores de probabilidades y tiempos
4 % tipoEleccion: selección de Deltas
5 %Salidas: Matriz D: valores Delta según la elección pedida
6
7 for i = 1 : size(Delta,1);
8 if (Delta(i,1) == 1)
9 Delta(i,4) = Delta(i,2);
10 end
11
12 if (Delta(i,1) == 0)
13 Delta(i,2) = Delta(i,4);
14 end
15 end
16
17 switch tipoEleccion
18 case {'DeltasMaximos'}
19 Delta = [Delta(:,2),Delta(:,4)];
20 D = (max(Delta'))';
21 case {'DeltasMinimos'}
22 Delta = [Delta(:,2),Delta(:,4)];
23 D = (min(Delta'))';
24 case {'DeltasMasProbables'}
25 for i = 1 : size(Delta,1)
26 if (Delta(i,1) >= Delta(i,3))
27 nuevaDelta(i,1) = Delta(i,2);
28 else
29 nuevaDelta(i,1) = Delta(i,4);
30 end
31 end
32 D = nuevaDelta;
33 otherwise
34 disp('Error inesperado');
35 D = 0;
36 end
```

I.1.h. optimizacion.m

Programa que organiza los datos provenientes de las matrices Pre, Post, Mi y Deltas seleccionados, para introducirlos como parámetros a la función de optimización de Matlab.

```
1 function [y,tho] = optimizacion(Pre,Post,D,Mi)
2
3 f = Pre * D;
4 f = (-1)*f';
5 %Desigualdades
6 A = [];
7 b = [];
8 %Igualdades
9 Aeq = Post - Pre;
10 Aeq = [Mi,Aeq]';
11 beq = zeros(size(Aeq,1),1);
12 beq(1,1) = 1;
13 %Límites
14 lb = zeros(size(Aeq,2),1);
15
16 [y,funObjetivo] = linprog(f,A,b,Aeq,beq,lb);
17 tho = 1/(-funObjetivo);
```

I.1.i. analisisCompleto.m

Programa que optimiza y simula todos los archivos contenidos en el directorio de trabajo de Matlab. Se introducen los mismos parámetros de simulación que en 'simular.m'. Recurre a las funciones 'simular.m', contenida en el apartado I.1.c; a la función 'simplificarCircuito.m', contenida en el apartado I.1.k; a la función 'eleccionDelta.m', contenida en el apartado I.1.g y a la función 'optimCalculoThroughput.m', contenida en el apartado I.1.j.

```
1 function [listaArchivos, resulThroughputs] = analisisCompleto(bucles,
tiempo, pconfint)
2
3 %Crear listado de circuitos
4 lista = dir('*.g');
5 listaArchivos = lista;
6
7 %Crear tabla de resultados
8 % Columna 1: Throughput mínimo, para deltas máximos
9 % Columna 2: Throughput de simulación, para circuito optimizado
10 % Columna 3: Throughput máximo, para deltas mínimos
11 % Columna 4: Error de simulación, respecto de la media de mínimo y máximo
12 % Columna 5: Varianza del throughput de simulación
13 % Columna 6: Valor inferior del intervalo de confianza especificado
14 % Columna 7: Valor superior del intervalo de confianza especificado
15 resulThroughputs = [];
16 indiceResultados = 1;
17
18 %Crear tabla de nodos con los cuellos de botella calculados
19 tablaCuellosBotella = [];
20
21 for i = 1 : size(lista,1)
22 nombreArchivo = lista(i).name;
23 disp(['Analizando: ', nombreArchivo]);
24
25 [Pre, Post, Mi, Delta] = readgr(nombreArchivo);
26 for j = 1 : size(Delta,1)
27 if (Delta(j,3) == 0)
28 Delta(j,4) = Delta(j,2);
29 end
30
31 if (Delta(j,1) == 0)
32 Delta(j,2) = Delta(j,4);
33 end
34 end
35
36 [resulThroughputs(indiceResultados, 1), listaNodos] =
optimCalculoThroughput(Pre, Post, Mi, Delta, 'DeltasMaximos');
37 for j = 1 : size(listaNodos,1)
38 tablaCuellosBotella(listaNodos(j,1),1) = 1;
39 end
40
41 [resulThroughputs(indiceResultados, 3), listaNodos] =
optimCalculoThroughput(Pre, Post, Mi, Delta, 'DeltasMinimos');
42 for j = 1 : size(listaNodos,1)
43 tablaCuellosBotella(listaNodos(j,1),1) = 1;
44 end
45
46 [datoInnecesario, listaNodos] = optimCalculoThroughput(Pre, Post, Mi,
Delta, 'DeltasMasProbables');
47 for j = 1 : size(listaNodos,1)
48 tablaCuellosBotella(listaNodos(j,1),1) = 1;
49 end
50
51 ratioAntiguo = size(Mi,1);
52 [Pre, Post, Mi, Delta] = simplificarCircuito(Pre, Post, Mi, Delta,
tablaCuellosBotella);
53 disp(['Relación de simplificación: ',num2str(100-100*size(Mi,1)
/ratioAntiguo),'%']);
54
55 [Rendimientos, MediaVarianza, IntervaloRendimiento] = simular(Pre,
Post, Mi, Delta, bucles, tiempo, pconfint);
56 resulThroughputs(indiceResultados, 2) = MediaVarianza(1,1);
57 resulThroughputs(indiceResultados, 5) = MediaVarianza(1,2);
```

```
58 resulThroughputs(indiceResultados, 6) = IntervaloRendimiento(1,1);
59 resulThroughputs(indiceResultados, 7) = IntervaloRendimiento(1,2);
60
61 indiceResultados = indiceResultados + 1;
62 tablaCuellosBotella = [];
63 end
64
65 for i = 1 : size(resulThroughputs,1)
66 resulThroughputs(i,4) = 100*((resulThroughputs(i,1)+resulThroughputs(i,3))/2-
resulThroughputs(i,2))/resulThroughputs(i,2);
67 end
```

I.1.j. optimCalculoThroughput.m

Programa reducido de 'calculoCuellos.m', destinado exclusivamente al código 'analisisCompleto.m'. Recurre a las funciones 'eleccionDelta.m', contenida en el apartado I.1.g y a la función 'optimizacion.m', contenida en I.1.h.

```
1 function [valor, listaNodos] = optimCalculoThroughput(Pre, Post, Mi,
Delta, metodo)
2 %Función que devuelve el throughput del circuito optimizado según los
3 %deltas elegidos y la lista de lugares que conforman el cuello de botella
4 %Entradas: Matrices Pre, Post, Mi, Delta
5 % Deltas elegidos (DeltasMaximos o DeltasMinimos o
6 % DeltasMasProbables)
7 %Salidas: Throughput del circuito optimizado con los deltas elegidos
8 % Lista (matriz) con los lugares que conforman el cuello de
9 % botella
10
11 %Elección de distintos Deltas
12 switch metodo
13 case {'DeltasMaximos'}
14 D = eleccionDelta(Delta, 'DeltasMaximos');
15 case {'DeltasMinimos'}
16 D = eleccionDelta(Delta, 'DeltasMinimos');
17 case {'DeltasMasProbables'}
18 D = eleccionDelta(Delta, 'DeltasMasProbables');
19 otherwise
20 disp('Error inesperado');
21 D = 0;
22 end
23
24 [y,valor] = optimizacion(Pre, Post, D, Mi);
25
26 contFilaListaNodos = 1;
27 for i = 1 : size(y,1)
28 if (y(i,1) > 10e-8)
29 listaNodos(contFilaListaNodos,1) = i;
30 contFilaListaNodos = contFilaListaNodos + 1;
31 end
32 end
```

I.1.k. simplificarCircuito.m

Programa que simplifica las cuatro matrices de una Red de Petri según la lista de lugares que se facilite.

```
1 function [Pre, Post, Mi, Delta] = simplificarCircuito(Pre, Post, Mi,
Delta, tablaCuellosBotella)
2
3 nuevaPre = [];
4 nuevaPost = [];
5 nuevaMi = [];
6 nuevaDelta = [];
7
8 % Simplificar filas (nodos) según lista de nodos
9 for i = 1 : size(tablaCuellosBotella,1)
10 if (tablaCuellosBotella(i,1) == 1)
11 nuevaPre = [nuevaPre; Pre(i,:)];
12 nuevaPost = [nuevaPost; Post(i,:)];
13 nuevaMi = [nuevaMi; Mi(i,:)];
14 end
15 end
16
17 Pre = nuevaPre;
18 Post = nuevaPost;
19 Mi = nuevaMi;
20 C = Post - Pre;
21
22 nuevaPre = [];
23 nuevaPost = [];
24
25 % Simplificar columnas (transiciones) según nueva matriz C
26 for i = 1 : size(C, 2)
27 hayTransicionesVivas = false;
28 for j = 1 : size(C, 1)
29 if (C(j,i) ~= 0)
30 hayTransicionesVivas = true;
31 end
32 end
33
34 if hayTransicionesVivas
35 nuevaPre = [nuevaPre, Pre(:,i)];
36 nuevaPost = [nuevaPost, Post(:,i)];
37 nuevaDelta = [nuevaDelta; Delta(i,:)];
38 end
39 end
40
41 Pre = nuevaPre;
42 Post = nuevaPost;
43 Delta = nuevaDelta;
```

I.1.l. analisisMarcado.m

Programa que analiza la evolución del marcado del circuito. Recurre a las funciones 'eleccionDelta.m', contenida en el apartado I.1.g y a la función 'avanzarTransicion.m', contenida en I.1.d.

```
1 function evolucionMarcado = analisisMarcado(nombreArchivo, avances,
tipoDelta, testDeEstres, graficoNodo)
2 % Función que analiza cómo evoluciona el circuito, en base a la
3 % modificación en el marcado de la red
4 % Entradas: Nombre de archivo con el esquema de la red
5 % Número de avances para hacer la simulación
6 % Elección de los deltas: 'DeltasMinimos'
7 % 'DeltasMaximos'
8 % 'DeltasMasProbables'
9 % 'DeltasAleatorios' (deltas por defecto)
10 % Test de estrés: 1, activado; 0, desactivado
11 % Este test impone un marcado inicial de 30 en
12 % todos los nodos
13 % Gráfica de la evolución en el tiempo de un nodo
14 % Salidas: Matriz evoluciónMarcado:
15 % 1ª columna: Marcado inicial (30, en caso de test de estrés)
16 % 2ª columna: Marcado mínimo de cada nodo en todo el test
17 % 3ª columna: Marcado máximo de cada nodo en todo el test
18 % 4ª columna: Incremento máximo de tokens (para observar
19 % acumulaciones o cuellos de botella)
20 % 5ª columna: Decremento máximo de tokens (para observar las
21 % zonas donde más carga de trabajo se libera)
22 % 6ª columna: Diferencia entre el incremento y decremento
23 % máximos (para observar si cada nodo trabaja bien siempre, mal
24 % siempre, o alternativamente en el caso de deltas aleatorios)
25 % Gráfica con la evolución del nodo elegido
26
27 %Leer mapa inicial del circuito
28 [Pre, Post, Mi, Delta] = readgr(nombreArchivo);
29
30 if (strcmp(tipoDelta, 'DeltasAleatorios') == false)
31 D = eleccionDelta(Delta, tipoDelta);
32 Delta(:,2) = D;
33 Delta(:,4) = D;
34 end
35
36 if (testDeEstres == 1)
37 Mi(:,1) = 30;
38 end
39
40 marcadoInicial = Mi;
41 marcadoMinimo = Mi;
42 marcadoMaximo = Mi;
43 maximaAcumulacion = [];
44 maximaLiberacion = [];
45 maximaDiferencia = [];
46
47 datosGraficaNodo = [];
48
49 %Matriz de tiempos de disparo
50 % Columna 1: tiempos remanentes o -1 si está deshabilitada
51 % Columna 2: veces que se ha ejecutado una transición
52 Transiciones = -1 * ones(size(Pre,2),1);
53 Transiciones = [Transiciones, zeros(size(Transiciones,1),1)];
54
55 tiempoEjecucion = 0;
56 datosGraficaNodo(1,1) = tiempoEjecucion;
57 datosGraficaNodo(1,2) = Mi(graficoNodo,1);
58 contGraficaNodo = 2;
59
60 for i = 1 : avances
61 [Mi, Transiciones, tiempoDisparo] = avanzarTransicion(Pre, Post, Mi,
Delta, Transiciones);
62 tiempoEjecucion = tiempoEjecucion + tiempoDisparo;
63 datosGraficaNodo(contGraficaNodo,1) = tiempoEjecucion;
64 datosGraficaNodo(contGraficaNodo,2) = Mi(graficoNodo,1);
65 contGraficaNodo = contGraficaNodo + 1;
```

```
66 for j = 1 : size(Mi,1)
67 if (Mi(j,1) > marcadoMaximo(j,1))
68 marcadoMaximo(j,1) = Mi(j,1);
69 end
70
71 if (Mi(j,1) < marcadoMinimo(j,1))
72 marcadoMinimo(j,1) = Mi(j,1);
73 end
74 end
75 end
76
77 maximaAcumulacion(:,1) = marcadoMaximo(:,1) - marcadoInicial(:,1);
78 maximaLiberacion(:,1) = marcadoMinimo(:,1) - marcadoInicial(:,1);
79 maximaDiferencia(:,1) = marcadoMaximo(:,1) - marcadoMinimo(:,1);
80 evolucionMarcado = full([marcadoInicial, marcadoMinimo, marcadoMaximo,
maximaAcumulacion, maximaLiberacion, maximaDiferencia]);
81 plot(datosGraficaNodo(:,1), datosGraficaNodo(:,2))
```