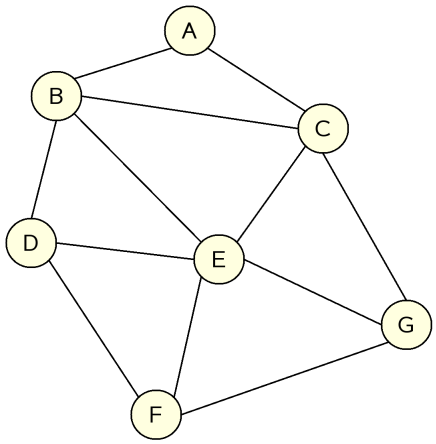
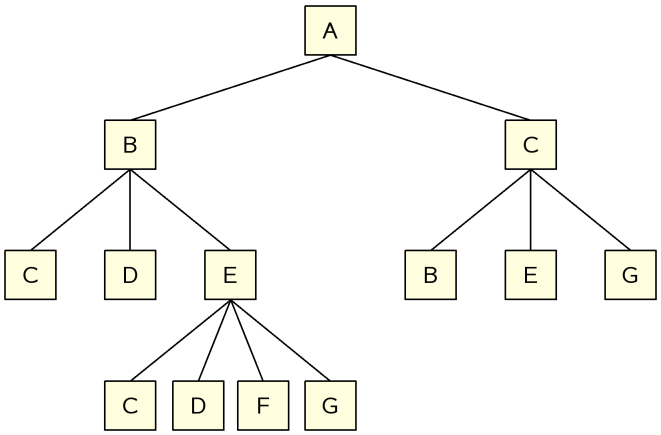
11-9-2008

1. Búsqueda de rutas en el grafo

- He dividido el proceso de búsqueda de rutas en tres partes: conversión del grafo en un árbol, podado del árbol, y recorrido del árbol para obtener los caminos finales. Lo he hecho así para poder modificar de manera más sencilla los criterios de poda, desacoplándolos de las otras dos partes del proceso.
- En la primera parte del proceso, el grafo se convierte en un árbol *n-ario*. Para ello, se parte del nodo origen de las rutas, que será la raíz del árbol, y se visitan los nodos adyacentes, que serán los nodos hijos. Se van anotando los nodos visitados, y se repite el proceso recursivamente hasta que ya no quedan más nodos por visitar. Por ejemplo, dado el siguiente grafo:



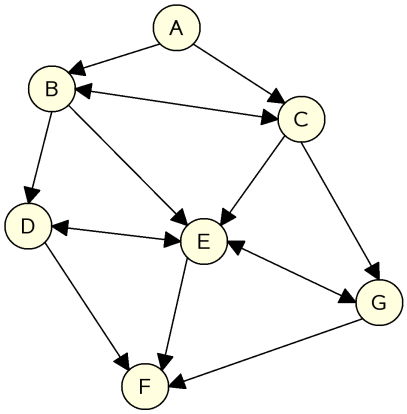
Obtendríamos un árbol como el siguiente (nota, el árbol no está completo porque sería demasiado grande, es sólo a modo de ejemplo):



- Una vez obtenido el árbol, se puede podar si se desea reducir el número de rutas posibles. Para redes pequeñas no es necesario, pero en redes grandes como la de AT&T es imprescindible, ya que de lo contrario el número de rutas entre dos nodos es tan elevado que puede tardar horas en procesarlas todas. Un criterio de poda que se ha visto que da buen resultado es el siguiente: si dos nodos A y B son adyacentes, se descartan todas aquellas rutas que conecten A y B a través de uno o más nodos intermedios. La implementación de este criterio es bastante sencilla; se realiza podando aquellas rutas que pasen por nodos adyacentes a los ya visitados.

- Por último, se recorre el árbol ya podado, buscando el nodo destino de la ruta. Se hace un recorrido recursivo sobre el árbol, acumulando los nodos visitados en una lista. Si se llega al nodo destino, se devuelve la lista con todos los nodos de la ruta. Si se llega a un nodo que no tiene adyacentes, se devuelve una lista vacía.

- Otra opción para acelerar la búsqueda de rutas sería preprocesar el grafo, eliminando nodos y enlaces innecesarios. Por ejemplo, se podrían borrar todos los nodos que forman caminos sin salida. Una idea que he probado y parece que también funciona bien es la de convertir el grafo en dirigido, clasificando los nodos por su distancia al origen y eliminando aquellos enlaces que impliquen retroceder en el grafo, es decir, volver a un nodo más cercano al origen. De esta manera se eliminan gran parte de los bucles. Por ejemplo, para el grafo anterior, si buscáramos la rutas entre los nodos A y F, el resultado sería:



2. Aplicación de reglas

- Como ya se comentó, es necesario clasificar las reglas en dos categorías: aquellas que sólo tienen en cuenta la red de interconexión entre los nodos finales, y las que procesan el escenario entero. Para ello he creado dos tipos de datos, a los que llamo *ReglaC* y *ReglaP* (de "completa" y "parcial").

- Las reglas completas sólo se pueden aplicar de una manera posible, pero puede ocurrir que una regla parcial se pueda aplicar de varias maneras a un escenario. Por ejemplo, la regla de canonización $Z_4 \otimes_d Z_4 = Z_4$ aplicada al escenario $Z_4 \otimes_d Z_4 \leftrightarrow Z_6 \leftrightarrow Z_4 \otimes_d Z_4$ podría dar dos resultados:

  - $Z_4 \leftrightarrow Z_6 \leftrightarrow Z_4 \otimes_d Z_4$
  - $Z_4 \otimes_d Z_4 \leftrightarrow Z_6 \leftrightarrow Z_4$

- Para resolver estos casos he escrito una función llamada *aplicarReglaP*, que toma una regla y un escenario y devuelve una lista con el resultado de aplicar la regla en cada posible posición del escenario.

- He modificado la función de generación del árbol. Ahora toma una lista de reglas completas y otra de reglas parciales, e intenta aplicarlas todas. He probado el escenario $A_6 \diamond N_6 \leftrightarrow Z_4 \otimes_d Z_4 \otimes_d Z_4 \leftrightarrow R_d^t \leftrightarrow Z_6 \leftrightarrow N_6 \diamond A_6$ con los operadores de conexión y de canonización, obteniendo el siguiente árbol:

```
*Main> main
A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6 ==> []
        A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Operador3,2,2,1)
                A6 ⋄ N6 ↔ Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,0,3,1)
                A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,1,2,1)
                        A6 ⋄ N6 ↔ Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,0,2,1)
        A6 ⋄ N6 ↔ Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,0,3,1)
                A6 ⋄ N6 ↔ Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Operador3,0,2,1)
        A6 ⋄ N6 ↔ Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,1,2,1)
                A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Operador3,1,2,1)
                        A6 ⋄ N6 ↔ Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,0,2,1)
                A6 ⋄ N6 ↔ Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6 ==> (Canonizacion1,0,2,1)
                        A6 ⋄ N6 ↔ Z4 ⊖d Z6 ↔ N6 ⋄ A6 ==> (Operador3,0,2,1)
```

- A partir del árbol anterior se obtienen los siguientes escenarios finales, con su respectiva lista de cambios:

```
A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6
                        |________|
                        Operador3
           |____________|
           Canonizacion1

--> A6 ⋄ N6 ↔ Z4 ⊙d Z6 ↔ N6 ⋄ A6

================================================================================

A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6
                        |________|
                        Operador3
                 |______|
                 Canonizacion1
           |______|
           Canonizacion1

--> A6 ⋄ N6 ↔ Z4 ⊙d Z6 ↔ N6 ⋄ A6

================================================================================

A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6
           |____________|
           Canonizacion1
           |_____________________|
           Operador3

--> A6 ⋄ N6 ↔ Z4 ⊙d Z6 ↔ N6 ⋄ A6

================================================================================

A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6
                 |______|
                 Canonizacion1
                 |_____________|
                 Operador3
           |______|
           Canonizacion1

--> A6 ⋄ N6 ↔ Z4 ⊙d Z6 ↔ N6 ⋄ A6

================================================================================

A6 ⋄ N6 ↔ Z4 ⊗d Z4 ⊗d Z4 ↔ Rd_td ↔ Z6 ↔ N6 ⋄ A6
                 |______|
                 Canonizacion1
           |______|
           Canonizacion1
           |_____________________|
           Operador3

--> A6 ⋄ N6 ↔ Z4 ⊙d Z6 ↔ N6 ⋄ A6
```

3. Tareas pendientes

- Puesto que ya funcionan la búsqueda de rutas y la aplicación de reglas, queda pendiente la tarea de integrar los dos procesos; es decir, generar los escenarios a partir de las rutas y aplicar las reglas estos escenarios. Esto no debería llevar mucho tiempo, y permitiría tener la parte más importante de la aplicación funcionando.
- Como se puede ver en el ejemplo anterior, el actual modelo de aplicación de reglas introduce muchas reduncancias, ya que en un escenario en el que se pueda aplicar una regla parcial en múltiples puntos, se terminarán probando todas las posibles permutaciones. Si en un ejemplo tan sencillo surgen 5 soluciones iguales, en un escenario complejo puede ser mucho peor. Por

ello habría que establecer algún criterio que eliminase estas redundancias. Algunas ideas que se me ocurren son:

  - Elegir únicamente la primera aplicación exitosa de una regla parcial. Si hay otras opciones, se probarán en niveles inferiores del árbol.
  - Dada una red $R_1 op_1 R_2 op_2 \ldots op_{n-1} R_n$, si se puede aplicar una regla a los nodos $R_i \ldots R_j$, se descarta cualquier posible aplicación que empiece en un nodo situado entre estos dos. Es decir, se eliminan las aplicaciones que sean subconjuntos de otra mayor, eligiendo sólo las más largas.
  - Las reglas se clasifican por niveles, de menor a mayor complejidad. Se podría forzar a que las reglas se aplicasen de manera escalonada, subiendo de nivel sólo en caso de que no se pueda aplicar ninguna regla de nivel inferior.

- He probado las tres bibliotecas para crear interfaces de usuario más importantes que hay: GTK2HS, WXHASKELL, y QTHASKELL. De momento no he conseguido que funcione ninguna; GTK2HS y WXHASKELL no compilan, mientras que QTHASKELL si lo hace, pero no hay documentación y no consigo probar los ejemplos. Intentaré hacer funcionar alguna de ellas, y si no lo consigo, buscaré otra opción.