75.24 Teoría de la Programación
75.31 Teoría de Lenguajes

Trabajo Práctico
Trenes: Playa de Maniobra

1º cuatrimestre 2011

1. Descripción del problema

Estamos encargados de maniobrar los vagones de un tren para armar la formación. Vamos a asumir que cada vagón se puede mover por si sólo sin necesidad de una locomotora que lo arrastre.

La descripción del trabajo que debemos realizar viene dada por dos secuencias de vagones, la formación actual y la formación deseada. Nuestro trabajo consiste en reorganizar los vagones de la formación actual, para obtener la formación deseada, con la ayuda de una playa o estación de maniobra, calculando una secuencia de movimientos.

La estación de maniobra está compuesta por una vía principal y dos vías de maniobras (vías uno y dos). La estación tiene un estado dado por los vagones que se encuentran en cada una de las vías y un movimiento describe como se mueve un vagón de una vía a otra (Ver Figura 1)

1.1. Objetivo

Nuestro objetivo es encontrar la secuencia más corta de movimimientos tal de reorganizar una formación actual en la vía principal en la formación deseada sobre la misma vía de la playa de maniobras.

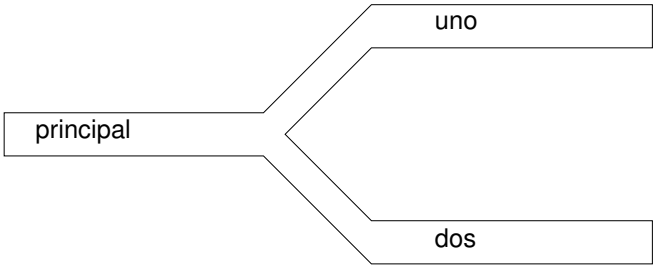

Figura 1: Estación de Maniobras

1.2. Propósitos del Trabajo Práctico

En este TP vamos a ejercitar como modelar un problema utilizando estructura de datos tales como listas y registros. Como se procesan las listas, usando desde patrones simples a patrones más complejos para controlar el procesamiento recursivo de listas.

2. Modelo

2.1. Trenes, vagones y estados:

Vamos a representar a los vagones como átomos y los trenes como una lista de átomos un tren no puede tener vagones duplicados ([*a b*]es un tren válido, mientras que [*a a*] es inválido).

Vamos a representar el estado de los trenes en la estación de maniobras con tres listas correspondientes a los trenes o formaciones en la vía principal, la vía uno y la vía dos. El estado de la estación quedará representado con un registro que tendrá como *features prinicipal*, *uno* y *dos* con las correspondientes listas como campos del registro. el *label* del registro será *estado*. En la figura 2 se puede ver el estado:

```
estado(principal:[a b] uno:[d] dos:[c])
```

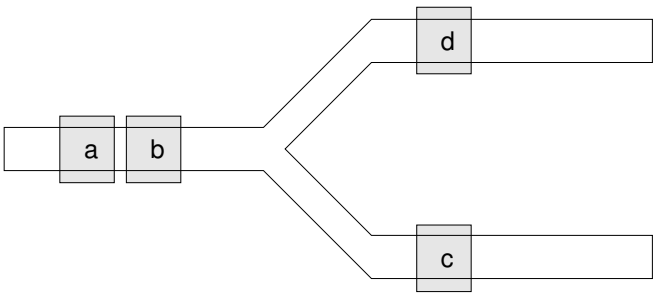

Figura 2: Estado de la estación

2.2. Movimientos:

Vamos a representar los movimientos con tuplas unarias, cuyos *labels* serán *uno* y *dos*. El único campo de la tupla será un entero (positivo o negativo) que representará la cantidad y la dirección de los vagones a mover de una vía a otra.

Por ejemplo $uno(2)$, $dos(2)$ y $uno(\sim 2)$ son movimientos válidos

2.3. Cambios de estados:

Cuando se aplica un movimiento la estación cambiará de estado, según las siguientes reglas:

- Si el movimiento es $uno(N)$ y $N > 0$, entonces los N vagones más a la derecha de la vía *principal* se trasladan a la vía *uno* (a la cabeza de la formación)
- Si el movimiento es $uno(N)$ y $N < 0$, entonces los N vagones más a la izquierda de la vía *uno* se trasladan a la vía *principal* (a la cola de la formación)
- El movimiento $uno(0)$ no tendrá efecto
- Los mismos conceptos se aplican a los movimientos concernientes a la vía *dos*

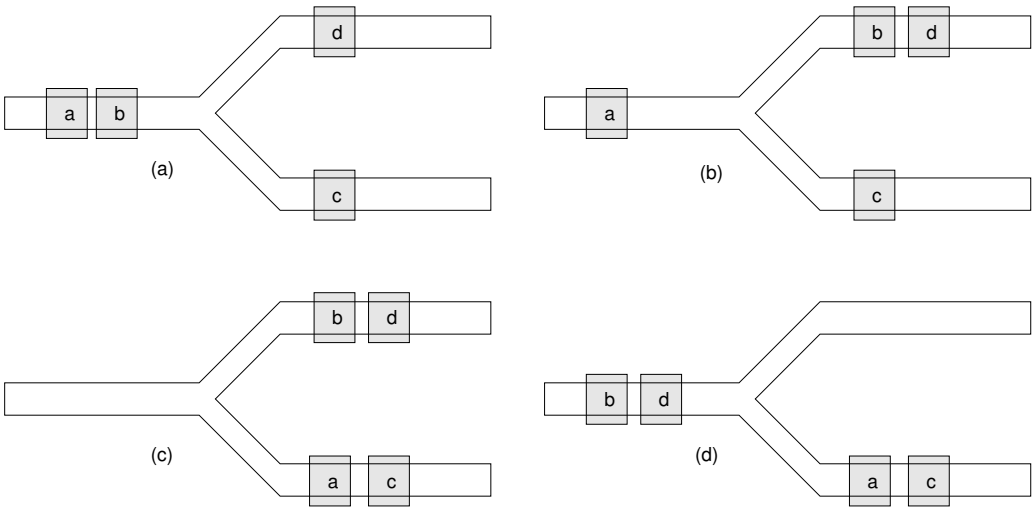

Figura 3: Estados y movimientos

En la figura 3 se ven los cambios de estados donde (a) es el estado inicial, (b) el siguiente estado al que se llega luego de ejecutar el movimiento $uno(1)$, (c) es el estado al que se llega desde (b) luego del movimiento $dos(1)$ y finalmente al estado (d) se llega desde (c) luego del movimiento $uno(\sim 2)$.

3. Procesamiento de listas

En primer lugar deberán desarrollar algunas rutinas básicas para procesar listas que serán útiles para el TP:

- $\{Longitud\ X_S\}$ devuelve la longitud de la lista X_S.
- $\{Tomar\ X_S\ N\}$ devuelve una lista con los primeros N elementos de la lista X_S. Si la longitud de X_S es menor que N deberá devolver la lista X_S.
- $\{Eliminar\ X_S\ N\}$ modifica la lista X_S eliminando los primeros N elementos. Si la longitud de X_S es menor que N deberá dejar *nil* en X_S.
- $\{Agregar\ X_S\ Y_S\}$ modifica la lista X_S agregando los elementos de la lista Y_S a la cola de X_S.

- $\{Miembro\ X_S\ Y\}$ verifica si el elemento Y se encuentra en la lista X_S.
- $\{Posicion\ X_S\ Y\}$ devuelve la posición de la primera aparición de Y en la lista X_S. Se puede asumir que Y se encuentra en la lista.

Crear el módulo *lista.oz* con las rutinas para procesar listas. Para utilizar el módulo en un archivo fuente se puede incluir la línea:
\insert lista.oz

4. Aplicar movimientos

Se debe escribir el procedimiento *AplicarMovimiento* que toma un estado inicial y una lista de movimientos y devuelve una lista de estados. Si la lista de movimientos que recibe tiene n elementos, entonces la lista de estado que devuelve tendrá n+1 elementos, siendo el primer estado de la lista el estado inicial que recibió como parámetro y el último el estado final al que se llega luego de aplicar todos los movimientos en el orden dado por la lista de movimientos que recibió.

Por ejemplo:

```
{AplicarMovimiento estado(principal:[a b] uno:nil dos:nil)
                                [uno(1) dos(1) uno(~1)]}
```

devuelve la lista de estados:

```
[estado(principal:[a b] uno:nil dos:nil)
 estado(principal:[a]   uno:[b] dos:nil)
 estado(principal:nil   uno:[b] dos:[a])
 estado(principal:[b]   uno:nil dos:[a])]
```

4.1. Diseño del procedimiento

El procedimiento *AplicarMovimiento* deberá realizar las siguientes acciones:

- Aplicar cada movimiento en forma recursiva
- Cuando no hay más movimientos para aplicar debe devolver el estado inicial (caso base)
- Para cada movimiento se debe decidir que vía está involucrada, para ello se deberá usar *pattern-matching.*
- Para cada vía donde involucrada en un movimiento se debe decidir si el vagón entra o sale de la vía (analizando si N es positivo o negativo)
- Para procesar las listas que representan a los trenes en cada vía, se usará las rutinas para procesar listas desarrolladas anteriormente.

- Si E es un estado, entonces el tren en la vía principal se accede como *E.principal* y de forma análoga se accede a los trenes en las vías uno y dos como *E.uno* y *E.dos*

```
fun {AplicarMovimiento E Ms}
   %% E es el estado inicial, Ms la lista de movimientos
   %% Devuelve la lista de estados
   case Ms of nil then ....
   [] M|Mr then
      %% Calcular E1 como un nuevo estado
      E1 = case M
           of uno(N) then
              if ... then ... else .... end
           [] dos(N) then
              if ... then ... else ... end
           end
   in
      ...
   end
end
```

5. Calcular movimientos

Desarrollar un procedimiento *CalcularMovimientos* que tome dos trenes (dos listas) X_S e Y_S y devuelva la lista de movimientos tal que transformen el estado:

```
estado(principal:Xs uno:nil dos:nil)
```

en el estado:

```
estado(principal:Ys uno:nil dos:nil)
```

X_S e Y_S deben contener los mismos vagones, es decir las listas son permutaciones de los mismos elementos.

Para diseñar este procedimiento vamos a utilizar la técnica recursiva:

- Si no hay vagones, entonces no hace falta agregar ningún movimiento a lista (caso base).
- Localizar el primer vagón de Y_S en el tren X_S. Sea este vagón Y.
- Mover todos los vagones dese Y inclusive y hasta el último vagón del tren $X_S$desde la vía *principal* hasta la vía *uno*.
- Mover todos los vagones restantes desde la vía *principal* a la vía *dos* (los vagones que originalmente estaban antes de Y en el tren X_S).
- Mover todos los vagones desde la vía *uno* a la vía *principal* (tal que ahora el vagón Y, es el primer vagón del tren).

- Mover todos los vagones desde la vía *dos* a la vía *principal*.

Por ejemplo, dados los trenes:

```
Xs=[a b]
Ys=[b a]
```

La lista de movimientos que devuelve $CalcularMovimientos$ será:

```
[uno(1) dos(1) uno(~1) dos(~1) uno(1) dos(0) uno(~1) dos(0)]
```

5.1. Partir tren

Para poder calcular el movimiento con el algoritmo descripto en el punto anterior, es necesario poder contar cuantos vagones hay que mover, por lo que será necesario escribir la rutina $PartirTren$ que tome un tren y un vagón y lo parta en dos listas H_S y T_S donde H_S será la lista con todos los trenes que se encontraban antes del vagón dado y T_S será la lista con todos los vagones ubicados después del vagón dado.

Por ejemplo:

```
{PartirTren [a b c] a}=nil#[b c]
{PartirTren [a b c] b}=[a]#[c]
```

Usar las rutinas para procesar listas

6. Minimizar los movimientos

La primera observación que se puede realizar para minimizar los movimientos es que si un vagón se encuentra en su posición no es necesario moverlo. Por ejemplo, dados los trenes:

```
Xs=[c a b]
Ys=[c b a]
```

La lista de movimientos optimizados será:

```
[uno(1) dos(1) uno(~1) dos(~1)]
```

7. Compresión de movimientos

Siguiendo unas cuantas reglas simples se puede minimizar aún más la cantidad de movimientos a realizar, comprimiendo la lista de movimientos:

1. Reemplazar el movimiento $uno(N)$ seguido del movimiento $uno(M)$ por el movimiento $uno(N + M)$.

2. Reemplzar el movimiento $dos(N)$ seguido del movimiento $dos(M)$ por el movimiento $dos(N + M)$.

3. Eliminar el movimiento *uno*(0).

4. Eliminar el movimiento *dos*(0).

Podría ser necesario aplicar varias veces las reglas hasta que no se puedan aplicar más para lograr la compresión deseada. Por ejemplo si la lista de movimientos es:

```
[dos(~1) uno(1) uno(~1) dos(1)]
```

Aplicando primero la regla 1 y luego la regla 2, nos quedamos con la lista vacía.

Se deberá desarrollar el procedimiento *ComprimirMovimientos* que aplique las reglas hasta que ninguna regla más sea aplicable. Por ejemplo:

```
fun {ComprimirMovimientos Ms}
   Ns={AplicarReglas Ms}
in
   if Ns==Ms then Ms else {ComprimirMovimientos Ns} end
end
```

8. Mejoras para minimizar los movimientos

Una última observación nos pude llevar a disminuir la cantidad de movimientos necesarios para lograr el tren deseado.

Hasta el momento y tal como fue descripto, el procedimiento *CalcularMovimientos* por cada vagón del tren que ubica correctamente pasa todos los vagones de las vías *uno* y *dos* de nuevo a la vía *principal*, cuando podría existir la posibilidad de usar la vía *dos* como auxiliar de la vía *uno* y seguir acomodando vagones de la vía *uno* en la vía *principal* y viceversa.

Para implementar estas mejoras se sugiere escribir un procedimiento auxiliar que se usará dentro de *CalcularMovimientos* que tenga cuatro parámetros, P_S: tren en la vía principal, U_S: tren en la vía uno D_S: tren en la vía dos, e Y_S: el tren deseado y que en cada paso recursivo ubique un vagón en la vía principal, siguiendo estos pasos:

- Sea Y el primer vagón de Y_S
- Si Y se encuentra en la vía principal, ubicar Y en el primer lugar sobre la vía principal siguiendo los pasos descriptos en el punto 5, pero dejando el resto de los vagones en las vías uno y dos.
- Si Y se encuentra en la vía uno, entonces pasar todos los vagones que estén delante de Y a la vía principal y luego a la vía dos, y finalmente pasar Y de la vía uno a la vía principal, tal que quede ubicado en su posición final.
- En forma análoga ubicar el vagón Y si se encuentra en la vía dos.

9. Presentación

El informe deberá contener el diseño detallado de todas las funciones y procedimientos desarrollados, respetando las estructuras de datos indicadas y considerando todos los detalles presentados en este trabajo para asegurar que la cantidad de movimientos que se realiza es la mínima.

El procedimiento principal se denominará $Maniobrar$ y tendrá dos parámetros X_S: el tren actual e Y_S: el tren deseado y deberá mostrar el listado de estados por los que pasan los vagones en las distintas vías de la playa de maniobra.

Se deberá proveer el código fuente impreso, y enviar por e-mail la versión digital tanto del informe como del código fuente a la dirección mfranzo@gmail.com

Se recomienda utilizar el visualizador desarrollado por Peter Van Roy y sus colaboradores, tanto durante la etapa de desarrollo como en la presentación final (`http://www.info.ucl.ac.be/people/PVR/ds/CS2104/Assignments/Visualizer.zip`), adaptando si hiciera falta, el código para que funcione con las estructuras de datos incluidas en este trabajo práctico.

10. Reconocimientos

La idea y formulación del presente trabajo práctico se tomó de un trabajo práctico que Peter Van Roy dió a sus alumnos.