Grupo Miércoles 10:00-12:00 semanas B

—Práctica 4—

Autor: Athanasios Usero

NIP: 839543

Ejercicio 1:

Para especificar el problema de las Torres de Hanoi con autómatas finitos, se parte de la idea de que cada estado va a ser una configuración válida de los discos en los tres postes verticales. Así, si queremos construir el autómata finito no determinista correspondiente $M = (\Sigma,\ Q, q_o, F, \Delta)$, definiremos su conjunto de estados Q como

$$Q = \{s \mid s\ es\ un\ estado\ regular\ de\ TH\}$$

Si Codificamos cada poste con el conjunto $T = \{0, 1, 2\}$, podemos definir una configuración como $s = s_3 s_2 s_1 \in T^3$, donde $s_i \in T$ es el poste donde se encuentra el disco i, siendo 1 el disco más pequeño y 3 el más grande. Considerando además que si $s_i = s_j, con\ i \neq j$, la configuración corresponderá a un estado regular, y que se parte del poste 0, se tiene:

$$Q = \{s_3 s_2 s_1 \in T^3|\},\ q_0 = 000 \in Q$$

Por otra parte, considerando que buscamos la secuencia de movimientos mínima entre dos estados perfectos (todos los discos en un mismo poste), parece conveniente que el lenguaje aceptado por M sea aquel de toda secuencia que lleve a un estado perfecto diferente del de partida, es decir $F = \{111, 222\} \subset Q$.

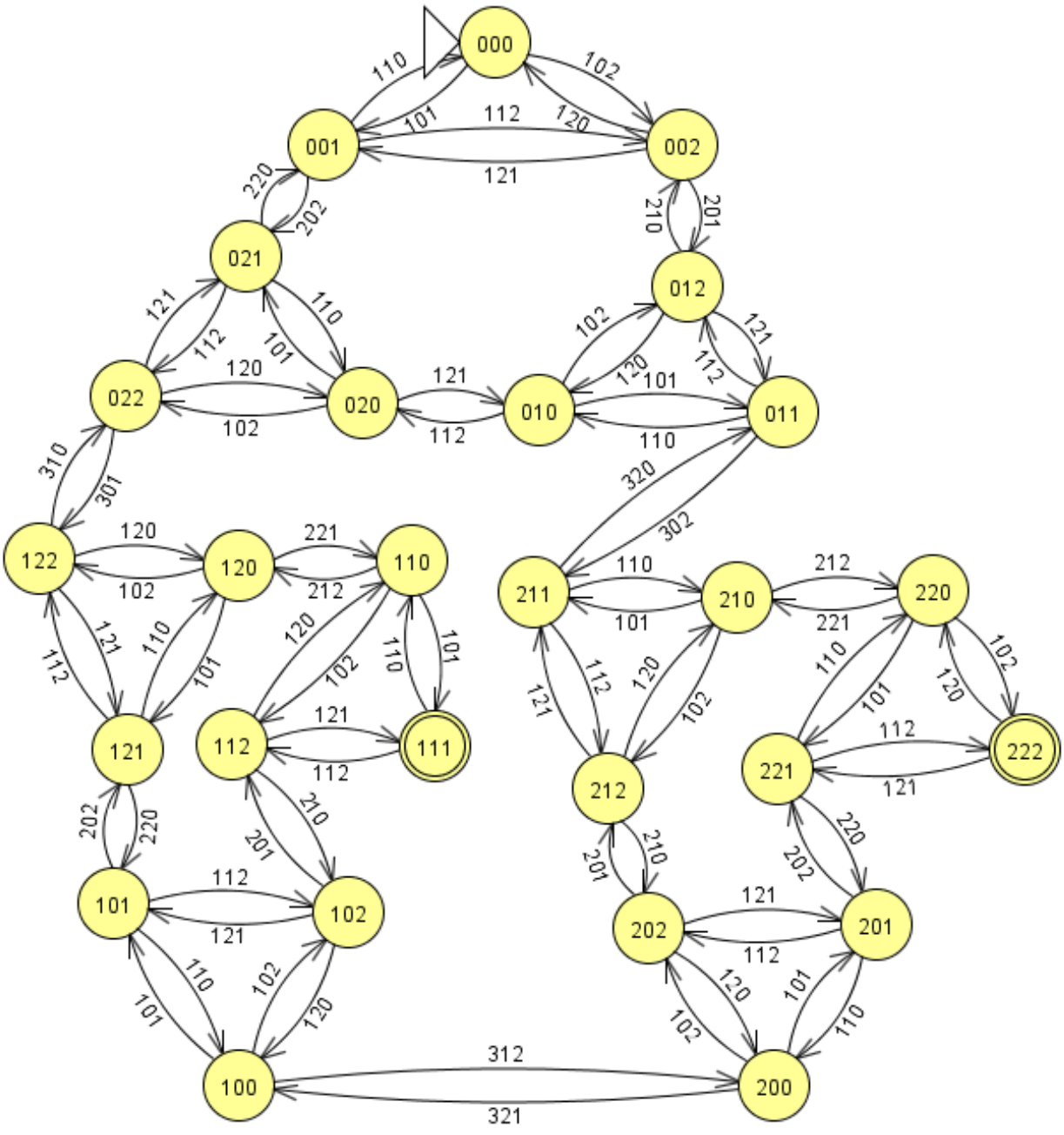
Cada transición corresponderá a un movimiento válido de un disco, el cual se codifica con la terna $(disco, origen, destino)$, donde se tiene que $disco \in \{1, 2, 3\}\ y\ origen, destino \in T$. Esto permite definir el alfabeto de entrada como

$\Sigma = \{xyv\ |x \in \{1,2,3\},\ y, v \in T\}$. Finalmente, se puede determinar todo estado alcanzable desde un estado regular dado: cada transición implica el movimiento de un solo disco (por lo que la codificación del estado alcanzable solo variará en el valor de un disco), este disco tendrá que ocupar la posición superior de un poste, y solo podrá moverse a un poste vacío o con discos más pequeños. Empleando la notación empleada:

- $\Delta(q_i, \sigma) = q_j,\ con\ q_i = s_{3i} s_{2i} s_{1i}, q_j = s_{3j} s_{2j} s_{1j} \in Q,\ \sigma = kxy \in \Sigma;\ si\ s_{ki} \neq s_{kj}\ y\ \forall t \neq k\ s_{ti} = s_{tj}, s_{ki} \neq s_{pi}\ si\ k > p,\ y\ s_{kj} \neq s_{pi}\ si\ k > p$

- $\Delta(q_i, \sigma) = \emptyset$ en caso contrario.

Gráficamente, el diagrama de estados es el siguiente:



Ejercicios 2 y 3:

El lenguaje de descripción de grafos basado en el autómata construido toma inspiración en el lenguaje DOT. Las reglas que describen esta gramática son las siguientes (Notación Backus-Nour):

<graph> ::= “graph” <nombre> “{ ” {<node>} “}”

<name> ::= {literal-caracter | “_” | literal-entero} (literal-caracter) {literal-caracter | “_” | literal-entero}

<node> ::= <ID> “ “ “->” [{<edge>”,”} <edge>]”;”

<edge> ::= <ID> “(” <ID> “)”

<ID> ::= {literal-caracter | literal-entero}(literal-caracter | literal-entero)

Arbitrariamente se impone que el nombre de un grafo contenga al menos un carácter y, entre llaves, aparece en cada línea un nodo del grafo junto con los arcos a otros nodos (pares de nodos destino junto con el símbolo de entrada (arco) entre paréntesis), si los hay. Así, el diagrama de estados de M se puede presentar textualmente como:

```
graph automata_Hanoi{
000 ->001(101),002(102);
001 ->002(112),021(202),000(110);
002 ->000(120),001(121),012(201);
021 ->001(220),022(112),020(110);
012 ->002(210),011(121),010(120);
022 ->021(121),020(120),122(301);
020 ->021(101),022(102),010(121);
011 ->012(112),010(110),211(302);
010 ->012(102),011(101),020(112);
122 ->022(310),120(120),121(121);
120 ->122(102),121(101),110(221);
211 ->011(320),210(110),212(112);
210 ->211(101),212(102),220(212);
121 ->122(112),120(110),101(220);
110 ->120(212),112(102),111(101);
212 ->211(121),210(120),202(210);
220 ->210(221),221(101),222(102);
101 ->121(202),102(112),100(110);
112 ->110(120),111(121),102(210);
111 ->110(110),112(112);
202 ->212(201),201(121),200(120);
221 ->220(110),222(112),201(220);
222 ->220(120),221(121);
102 ->101(121),112(201),100(120);
100 ->101(101),102(102),200(312);
201 ->202(112),221(202),200(110);
200 ->202(102),201(101),100(321);
}
```

Dada la descripción, se ha construido la gramática con las instrucciones pertinentes en Bison para traducir el grafo del autómata de Hanoi a una matriz de adyacencia de dimensión 27 (igual al número de nodos), donde cada fila y columna corresponden a un nodo diferente del autómata y el valor de la componente de la matriz dada por dichos índices corresponde a la etiqueta del arco desde el nodo de la fila hasta el nodo de la columna. Los nodos son cadenas de tres caracteres donde cada caracter es 0, 1 o 2. Es decir, son números de tres dígitos en base 3, que toma valores decimales en el rango [0, 26]. Por ello, se ha declarado una función en Bison que dada una cadena correspondiente a un estado devuelve el entero correspondiente a la codificación en decimal.

Asimismo, se han declarado varios tokens para construir la gramática: ID (nodos o etiquetas de arcos), GRAPH (tipo graph), NOMBRE (nombre del grafo), FLECHA ("->" para indicar transiciones de un nodo), COMA, PYC, LLAVEIZQ, LLAVEDER, PARENTIZQ, PARENTDER (coma, punto y coma, llaves y paréntesis) y FL (fin de línea). El programa en el fichero Flex se encarga de reconocer patrones y devolver los tokens correspondientes. Cabe mencionar el patrón

$$r = [a - zA - Z\_0 - 9]^*[a - zA - Z][a - zA - Z\_0 - 9]^*$$

que reconoce nombres de un grafo, pues estos deben contener al menos un carácter, y solo pueden además contener '_' o dígitos numéricos; y que se ha definido un nuevo tipo de dato cadena *nombre* para yylval que afecta al token ID, por lo que el patrón [0-9]+ se devuelve en flex como una cadena y no como un entero.

La gramática consta de cuatro símbolos no terminales, que se muestran junto con las reglas de producción correspondientes. El símbolo inicial es

$$GRAFO \rightarrow GRAFO\ graph\ nombre\ llaveizq\ fl\ TRANSCIONES\ llaveder\ fl \mid \epsilon$$

Correspondiente a la generación de un grafo, con su línea de declaración, conjunto de líneas de nodos y transiciones y línea de cierre. El símbolo no terminal *transiciones* viene dado por

$$TRANSICIONES \rightarrow TRANSICIONES\ id\ flecha\ DESTINOS\ pyc\ fl \mid \epsilon$$

Y genera el conjunto de líneas con un nodo origen y una flecha que lo separa por un conjunto de nodos destino y transiciones que finalizan necesariamente en punto y coma (nótese que se permite definir un grafo vacío, sin nodos). A su vez, destinos viene dado por

$$DESTINOS \rightarrow ARCOESTADO\ coma\ DESTINOS \mid ARCOESTADO$$

que genera un conjunto de estados y arcos separados por comas (nótese que no se permite definir un nodo sin transiciones, pues para el caso de las torres de Hanoi cualquier movimiento es reversible y no tendría sentido tener un nodo alcanzable sin arcos de salida). Por último, arcoestado se declara como

$$ARCOESTADO \rightarrow \ \ id\ parentizq\ id\ parentder$$

Generando un identificador de nodo junto con otro de arco entre paréntesis.

Para facilitar las acciones de traducción asociadas a algunas reglas, viene dado un registro *ListaTransiciones* que agrupa una cadena con un nodo origen, dos vectores de cadenas con nodos destino y las etiquetas de los arcos a estos, y un entero con el número total de transiciones existentes para el nodo origen (que se inicializa a 0). Así, cada vez que se genere un arco-estado,

se añade el primer identificador a la siguiente componente no utilizada del vector de nodos destino, análogamente con el segundo identificador y el vector de etiquetas, y se incrementa en uno el total de transiciones. Así, una vez se genera una línea de nodos y transiciones completa, la componente nodo origen del registro se inicializa con el identificador y su codificación a decimal se almacena en una variable cadi. Además, se recorren el total de componentes de los vectores, almacenando en cadj la codificación a decimal de los destinos, en el índice correspondiente de la matriz de adyacencia su etiqueta y finalmente reseteando el campo *total* para la lectura del siguiente nodo. De este modo, al final de la llamada al analizador gramatical la matriz correspondiente contendrá el conjunto de relaciones de transición del grafo (siendo la cadena vacía en caso de no existir).

Una vez diseñada la gramática, nuestra función inicializa el registro auxiliar y matriz de adyacencia, establece como estado regular origen “000” y cualquier otro estado perfecto como fin, y llama al parser. Una vez finaliza la llamada y se obtiene la matriz de adyacencia, se realiza sucesivamente el producto de esta sobre sí misma almacenado en una matriz potencia (a través de otra auxiliar resultado) hasta que la componente en el índice dado por el estado origen y fin no sea la cadena vacía. El valor en este caso es el conjunto mínimo de transiciones que llevan desde el estado inicial al otro estado perfecto.

A continuación, se muestran los resultados obtenidos para cada estado perfecto:

```
Nodo inicial  : 000
Movimientos   : 101-202-112-301-120-221-101
Nodo final    : 111
```

```
Nodo inicial  : 000
Movimientos   : 102-201-121-302-110-212-102
Nodo final    : 222
```