PRÁCTICA 3
MAQUINA DE MEALY

MODELOS DE COMPUTACIÓN

GRUPO B

Juan Carlos Ruiz Fernández B2

Contenido

Introducción a PairCodec

Para esta práctica se presentó la oportunidad de crear una herramienta para codificar, y decodificar, mensajes con una maquina de estados finitos (Mealy Machine).

PairCodec trabaja sobre cadenas binarias. Codifica, como su nombre indica, en pares de ceros o unos haciendo que cada uno de éstos se transforme en otro par dando así cuatro nuevas posibilidades. Pero no se queda ahí, un determinado par no se transformará siempre en el mismo par.

Componentes

Prerrequisitos

El único requisito es que la cadena sea par. Si se diera el caso que el mensaje fuera impar, se tiene que añadir un cero en la parte más significativa del mensaje. Da igual que sea a derechas o izquierdas, PairCodec no es sensible a ello. Solo los usuarios que lo usen deben ser conscientes en que orden prefieren codificarlo ya que debe mantenerse.

Pares

Como se trabaja con pares de caracteres de cadenas formadas por ceros y unos, solo se actuará sobre cuatro posibles combinaciones: *00, 01, 10* y *11.*

Reglas del códec

Las transformaciones de pares se dividen en cuatro *cuadros*, los cuales se llaman A, B, C y D.

Cuadro A

Este *cuadro* de permutaciones corresponde a un desplazamiento de una posición. Siendo entonces las trasformaciones:

00->01	01->10	10->11	11->00

Cuadro B

En este caso es, de nuevo, otro desplazamiento. El cual es de dos posiciones, quedando tal que así:

00->10	01->11	10->00	11->01

Cuadro C

En este caso se puede decir un desplazamiento en 3 posiciones o la resta del par menos *1*:

00->11	01->00	10->01	11->10

Cuadro D

Este ultimo *cuadro* es más especial, también se usa el desplazamiento de una sola posición pero además se añade una inversión de los caracteres al final, dando como resultado:

00->01->10	01->10->01	10->11->00	11->00->11

Cuadro resumen de las permutaciones

A	
00	01
01	10
10	11
11	00

B	
00	10
01	11
10	00
11	01

C	
00	11
01	00
10	01
11	10

D		
00	01	10
01	10	01
10	11	00
11	00	11

Comportamiento

Codificación

La regla de codificación se basa en todo lo anterior pero no se queda ahí. Para empezar toma los dos primeros valores como referencia de comienzo, es decir, dependiendo de como comience la cadena codificará con el *cuadro* de permutaciones correspondiente. Siendo la siguiente la asignación:

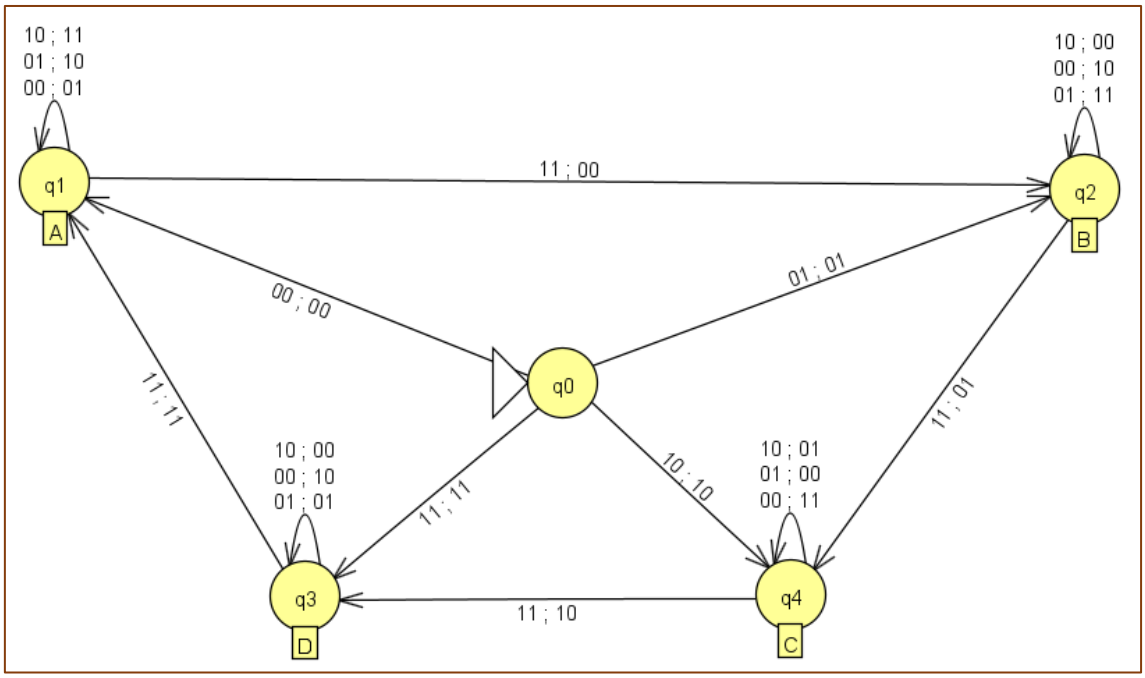
00->Cuadro A 01->Cuadro B 10->Cuadro C 11->Cuadro D

Al principio se comentó que las permutaciones no serían iguales en toda la cadena, eso se produce a que cuando se lee el par *11* se cambiará al siguiente cuadro de permutaciones siguiendo este orden: A->B->C->D. Siendo cíclico.

Por lo que la codificación vendrá determinada por el comienzo de la cadena y el numero de apariciones de la subcadena *11*.

Mealy Machine

La maquina de estados usada para la codificación se encuentra en el archivo *PairCodecC.jff* para su ejecución en *JFlap:*



En él se encuentran cinco estados, siendo *qo* el inicial que determinará la lectura de los dos primeros caracteres y ejecutará el cuadro correspondiente. El resto de los estados tienen etiquetas a los cuadros a los que están asociados.

Las transiciones entre esos estados son los cambios de cuadros cuando se lee la cadena *11*, aunque antes de pasar al siguiente cuadro trasforma esa cadena según su permutación.

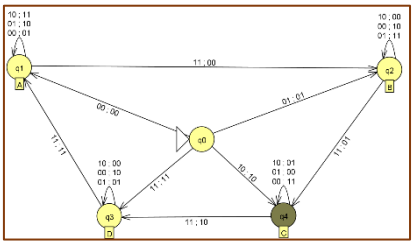
No tiene estados finales por lo que aceptará cualquier cadena que sea par, dando error para la cadena que no lo sea.

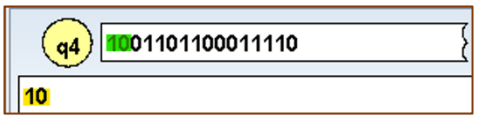
Ejemplo

Para un ejemplo rápido y corto se usará la cadena 1001101100011110. Se puede hacer un desglose previo donde con 10 se seleccionará el cuadro C y se generará las dos trasformaciones siguientes acorde al cuadro C. Al leer 11 se transformará a 10 pero cambiará de estado al D, en el cual las trasformaciones son diferentes a los anteriores. Hasta leer de nuevo 11, devolviendo su correspondiente y acabando de codificar en el cuadro A.

Paso 1 – Selección de cuadro

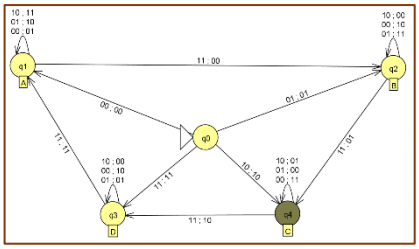
Como los dos primeros valores corresponden a 10, cambia de estado a *q4* que es el cuadro C. Se devuelve dichos primeros valores sin cambiar para luego la decodificación.

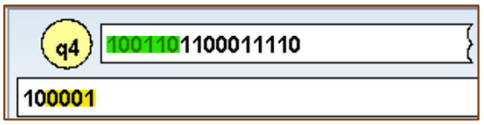




Paso 2 – Codificación con cuadro C

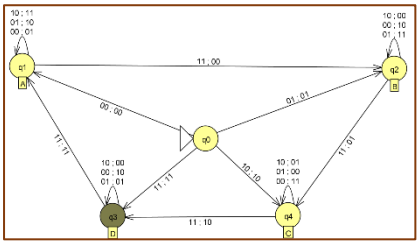
Después de este primer paso se encuentra dos pares distintos de 11, los cuales los trasforma de acuerdo con la tabla de C, permaneciendo en *q4:*

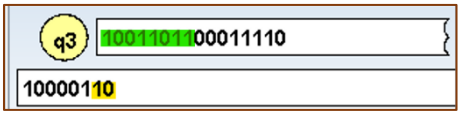




Paso 3 – Cambio de cuadro

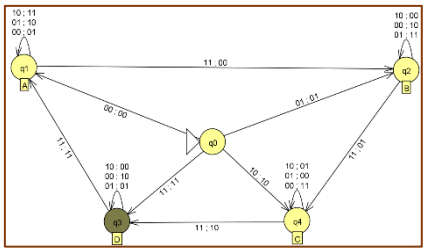
Como el siguiente par leído es 11, se devuelve esos caracteres respecto el cuadro C pero se cambia de estado a *q3:*

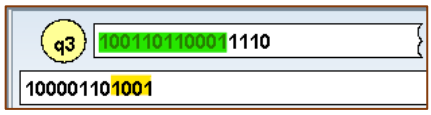




Paso 4 – Codificación con cuadro D

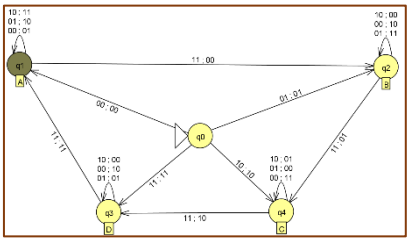
De igual manera que con C, leerá dos pares distintos a 11 y los trasformará según dicta D. Permaneciendo en *q3*:

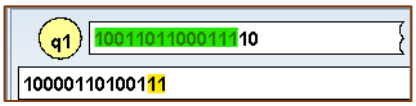




Paso 5 – Cambio de cuadro

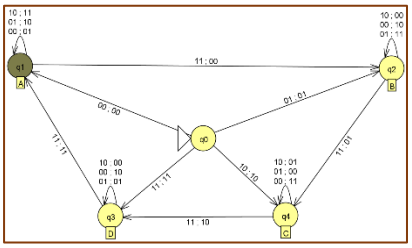
El siguiente par es 11, por lo que lo devolverá y pasará de nuevo de estado. Esta vez al A:

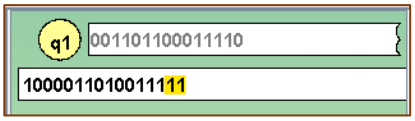




Paso 6 – Ultima codificación

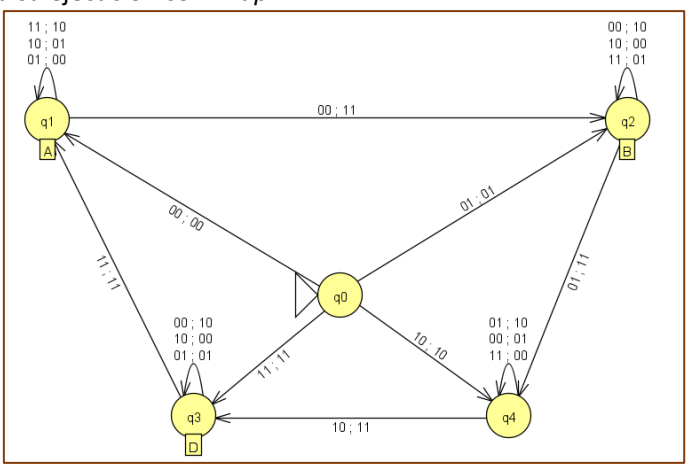
Solo queda por leer el ultimo par de caracteres, por lo que lo codificará respecto A y terminará:





Decodificación

En este caso se hará exactamente el proceso inverso, el autómata tendrá los mismos estados y mismo número de transiciones. Con la salvedad de que la entradas y salidas de las transiciones estarán invertidas, es decir, las salidas del autómata codificador serán las entradas del decodificador y las entradas del codificador las salidas del decodificador. El archivo que lo contiene se llama *DePairCodec.jff* para su ejecución con *JFlap*.

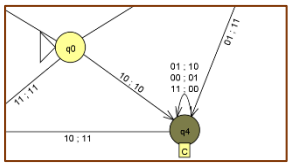


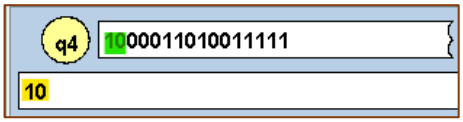
Ejemplo

Para demostrar su funcionamiento se va a tomar la salida del codificador y comprobar que coincide con su entrada. Para recordar, la secuencia inicial era 1001101100011110 y su salida codificada es 1000011010011111. Esta ultima es la que se ingresará para tomar el ejemplo.

Paso 1 – Selección de cuadro

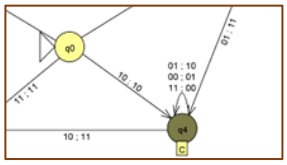
Como la otra vez, se sitúa en el estado *q4* que corresponde al cuadro C. Devuelve tal cual los dos primeros caracteres.

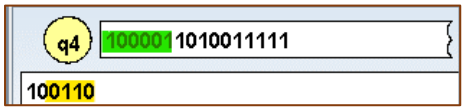




Paso 2 – Decodificación en C

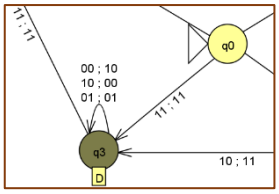
Una vez devueltos los dos primeros caracteres sin alterar se decodificará los pares que se encuentren hasta que se encuentre el par que corresponde a 11.

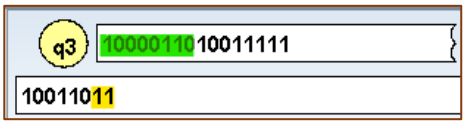




Paso 4 – Cambio de cuadro

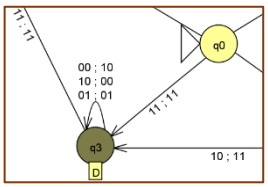
Estando en el cuadro C, se sabe cual seria el par generado en la codificación para cambiar de estado. Es el caso de 10. Entonces se devolvería su homologo 11 y se cambia al estado D.

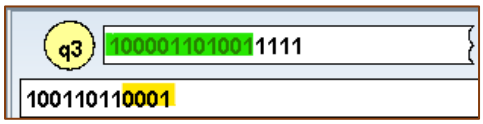




Paso 5 – Decodificación en D

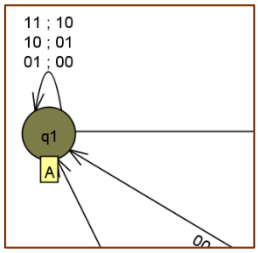
En este caso se producirá el cambio de estado cuando se lea 11, ya que coincide en este cuadro. Hasta entonces se decodifican los dos pares siguientes.

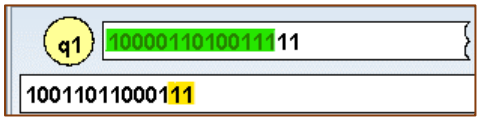




Paso 6 – Cambio de cuadro

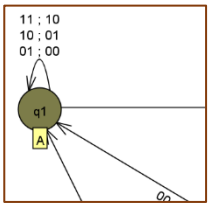
El siguiente par corresponde al cambio de estado, entonces lo devolverá y pasara al cuadro A.

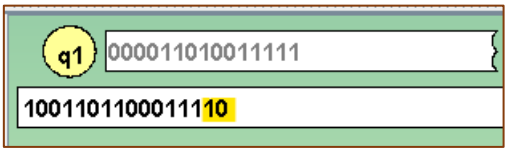




Paso 7 – Decodificación final en A

Solo queda un par que corresponde a 11, en el cuadro A corresponde a 10, lo devuelve y termina la ejecución dando asi el producto final.





Rescatando de nuevo la entrada inicial sin decodificar 1001101100011110, se comprueba fácilmente que coincide con la salida del decodificador demostrando asi su funcionamiento.

Conclusiones

Nada más lejos de ser una buena maquina de codificación, lo optimo hubiera sido en asemejarse a *Enigma* en la parte de que el usuario puede seleccionar ciertas permutaciones. Una forma habría sido crear un preámbulo o cabecera que seleccionase más modos, pero sería lo mismo que ahora pero con mas combinaciones. De *Enigma* se ha intentado asemejar en las permutaciones y en los cambios en las permutaciones, donde esta hacia uso de las ruedas que por cada letra pulsaba iban girando aquí se ha creado de forma que cada vez que se lea un 11 cambie, *o gire,* el cuadro de permutaciones, *la rueda*.