Facultad de Informática

UNIVERSIDAD DE MURCIA

PRÁCTICA 1

Autómatas y Lenguajes Formales

Última actualización: 14 de octubre de 2019

Eduardo Martínez Graciá, Mercedes Valdés Vela,
Santiago Paredes Moreno, José Manuel Juárez Herrero

Dpto. de Ingeniería de la Información y las Comunicaciones
Facultad de Informática de la **Universidad de Murcia**

INDICE

INTRODUCCIÓN

INTRODUCCIÓN (1)

- En las transmisiones digitales a través de canales con interferencias, es necesario **proteger la señal**.
- La protección de la señal consiste en añadir **información redundante** para que, en ciertas condiciones, se pueda detectar el error y la señal pueda ser recuperada.

INTRODUCCIÓN (2)

- Cuanta más información redundante se añade, mayor protección se obtiene, pero también es mayor el coste de la transmisión de un bit de la señal.
- En ingeniería nos encontramos con infinidad de situaciones en las que hay que llegar a un **compromiso entre bueno y barato**.

INTRODUCCIÓN (3)

¿Por qué se producen errores de transmisión?...

INTRODUCCIÓN (3)

¿Por qué se producen errores de transmisión?...

Mejor, evitamos subir a arreglarlo...

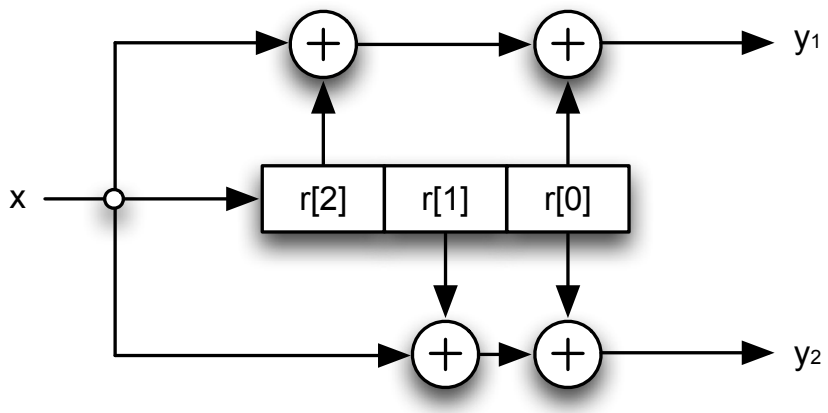
INTRODUCCIÓN (3)

¿Por qué se producen errores de transmisión?...

Mejor, evitamos subir a arreglarlo...

INTRODUCCIÓN (4)

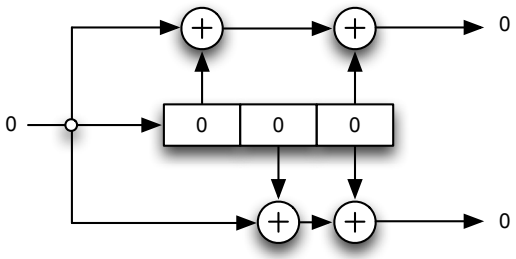
- Los sistemas actuales de transmisión digital inalámbrica (como las transmisiones de teléfonos móviles o la televisión digital terrestre) utilizan los denominados *códigos convolucionales*.
- Se basan en el uso de registros de desplazamiento a nivel de bits y de operaciones XOR. Usaremos uno sencillo:



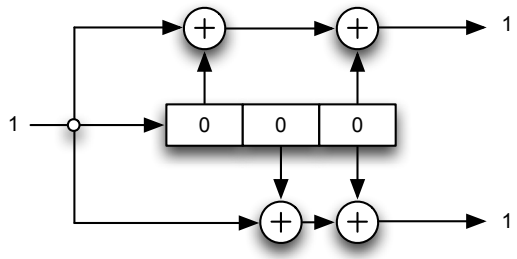
$$y_1 = x + r[2] + r[0]$$
$$y_2 = x + r[1] + r[0]$$
$$r[0] = r[1]$$
$$r[1] = r[2]$$
$$r[2] = x$$

INTRODUCCIÓN (5)

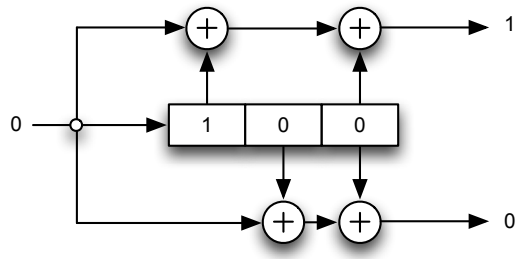
Secuencia de estados del codificador con la entrada 010111:



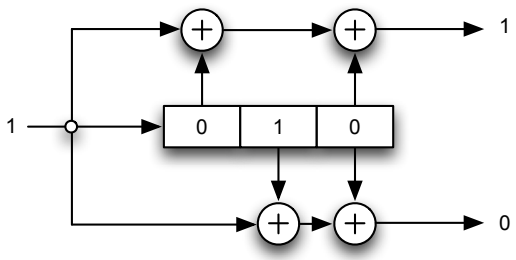
(a) $x = 0$



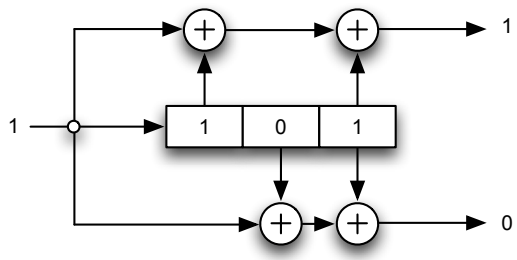
(b) $x = 1$



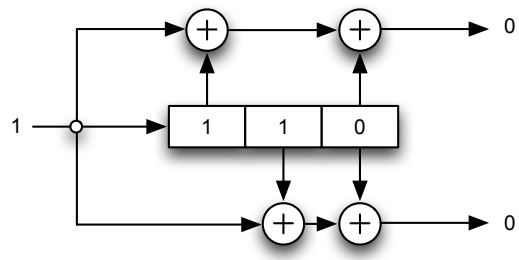
(c) $x = 0$



(d) $x = 1$



(e) $x = 1$



(f) $x = 1$

PRÁCTICA 1

LA PRÁCTICA

Esta práctica tiene **dos partes**:

1. Implementar con JFLAP un AFD que permita validar la salida de un codificador convolucional como el indicado en la sección previa.
2. Implementar en Python un programa que emplee el AFD anterior para verificar cadenas del alfabeto $V = \{0, 1\}$ almacenadas en un fichero. La validación se debe implementar mediante el algoritmo general de simulación de un AFD (página 19 del tema 2).

EL AUTÓMATA

El autómata debe tener en cuenta las siguientes reglas:

1. Debe ser un **autómata finito determinista**.
2. El estado inicial del autómata representa la configuración del codificador con el registro de bits a 0.
3. El autómata puede tener un alfabeto de entrada distinto de $V = \{0, 1\}$ siempre y cuando la conversión de una cadena de V^* al nuevo alfabeto se realice en orden lineal.

EL PROGRAMA (1)

El programa debe implementarse en Python haciendo uso del paquete `jflap` que aparece descrito en el capítulo 8 de los *Apuntes de Python*. El programa debe realizar los siguientes pasos:

1. Solicitar un nombre de fichero al usuario.
2. Si el nombre de fichero es una cadena vacía, el programa debe terminar.
3. Si el fichero no existe, el programa debe indicarlo y volver a solicitar un nombre nuevo.
4. Si el nombre es correcto, el programa debe procesar cada línea del fichero como una secuencia de bits que sale del codificador convolucional.
5. El programa debe imprimir un mensaje con el resultado de la validación de cada línea.

EL PROGRAMA (2)

El formato de salida de cada línea debe ser el siguiente:

`Línea X CADENA: RESULTADO`

donde:

- `X`: es el número de línea
- `CADENA`: es la secuencia de bits de la línea
- `RESULTADO`: es o bien `válida` o bien `no válida en N`, donde `N` indica la posición (empezando en 1) del primer bit en el que la validación de la cadena falla.

EL PROGRAMA (3)

Por ejemplo, un fichero **`prueba.txt`** con el siguiente contenido:

```
001110101000
11011101011010
1101001111
11011010
00001101111110
```

genera la siguiente salida:

```
Línea 1 001110101000: válida
Línea 2 11011101011010: válida
Línea 3 1101001111: válida
Línea 4 11011010: no válida en 6
Línea 5 00001101111110: no válida en 12
```

EVALUACIÓN

LA EVALUACIÓN

Este ejercicio puntúa un máximo de **2 puntos**: 1 punto por el autómata y 1 punto por el programa Python. La entrega de la práctica consistirá en dos ficheros:

- `practica1.jff`: autómata desarrollado con JFLAP.
- `practica1.py`: programa Python de validación.

La práctica se entrega al final del cuatrimestre, junto con las restantes.

OPCIÓN ALTERNATIVA

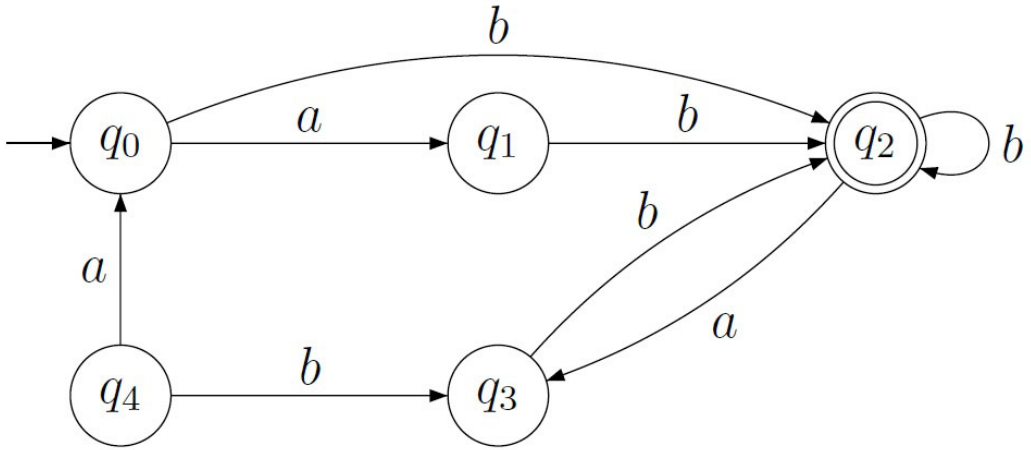
MINIMIZACIÓN DE AFD (1)

En lugar de diseñar el autómata y el programa de validación, puedes **obtener 3 puntos** si implementas un programa en Python que minimice un autómata finito determinista. El programa debe realizar los siguientes pasos:

1. Solicitar un nombre de fichero JFLAP al usuario.
2. Si el nombre de fichero es una cadena vacía, el programa debe terminar.
3. Si el fichero no existe, el programa debe indicarlo y volver a solicitar un nombre nuevo.
4. Si el nombre es correcto, el programa debe leer el fichero JFLAP y generar una salida con la tabla de transiciones del autómata mínimo equivalente.

MINIMIZACIÓN DE AFD (2)

Supongamos que la entrada al programa es el siguiente autómata:



La salida debe tener un formato similar a éste:

```
1     Estados   | a           | b
2  ------------------------------------
3   ->{q0}      | {q1,q3}     | {q2}
4     {q1,q3}   | {qerror}    | {q2}
5    #{q2}      | {q1,q3}     | {q2}
6     {qerror}  | {qerror}    | {qerror}
```

PAQUETE JFLAP

PAQUETE JFLAP (1)

En los recursos del Aula Virtual puedes encontrar la carpeta `Python`. Dentro de ella hay un fichero `jflap.zip`. Al descomprimirlo se genera una carpeta `jflap` dentro de la cual encontrarás los siguientes tres ficheros:

- `__init__.py`: fichero de inicialización del paquete.
- `Afd.py`: es el módulo principal que contiene la clase `Afd` que manejaremos para manipular el autómata desde Python.
- `Transiciones.py`: es un módulo auxiliar de `Afd.py`. No es necesario importarlo.

Mueve la carpeta `jflap` al directorio `src` del proyecto de Eclipse en el que estés trabajando.

PAQUETE JFLAP (2)

Para crear un autómata desde Python:

```
1 from jflap.Afd import Afd
2 from sys import stderr
3 ruta = r'C:\prueba.jff'
4 try:
5     autómata = Afd(ruta)
6 except FileNotFoundError:
7     print('La ruta está mal',file=stderr)
8 except Exception as error:
9     print('Problemas:',error,file=stderr)
```

Si se usa JFLAP versión 8beta, debes crear el autómata así:

```
1 autómata = Afd(ruta,8)
```

PAQUETE JFLAP (2)

Para mostrar el contenido del autómata:

```
1 autómata = Afd(ruta)
2 autómata.mostrarAfd()
```

Para obtener el estado inicial:

```
1 autómata = Afd(ruta)
2 estadoInicial = autómata.getEstadoInicial()
```

Para consultar una transición:

```
1 estadoSig = autómata.estadoSiguiente(
      estadoActual,simboloActual)
```

Si la transición no está definida, el método devuelve `None`.

PAQUETE JFLAP (3)

Para comprobar si un estado es final:

```
1 estadoSig = autómata.estadoSiguiente(
      estadoActual,simboloActual)
2 print(autómata.esFinal(estadoSig))
```

Devuelve `True` en caso de que sí lo sea, `False` en caso contrario.

Se puede consultar el alfabeto que emplea el autómata con el método `getAlfabeto()`:

```
1 # Conjunto de símbolos
2 alfabeto = autómata.getAlfabeto()
```