

Practica 7 – Procesamiento del habla

Implementación de un sintetizador concatenativo de difonos

Antonio José Morano Moriña

Alberto Montes Ramos



Universidad
de Huelva

Índice del contenido

1.-Finalidad de la práctica	3
2.-Creación del inventario de sonidos	4
2.1.-Grabación y creación de los .wav de cada sonido	5
3.- Creación del programa capaz de concatenar los archivos generados en el inventario	8
3.1.- Cambio de la prosodia de las palabras.....	9

Finalidad de la práctica

El objetivo de este trabajo practico consiste en implementar un sintetizador concatenativo de dífonos para L, siendo este un lenguaje compuesto por 6 fonos:

[a], [e], [f], [k], [m], [r], [s] y [t].

Pudiendo obtener las siguientes sílabas, las cuales grabaré y añadiré al inventario del sistema:

V -> [a], [e].

CV -> [fa], [fe],[ka],[ke],[ma],[me],[ra],[re],[sa],[se],[ta],[te].

CCV -> [fra],[fre],[kra],[kre],[tra],[tre].

VC -> [as], [es].

CVC -> [fas], [fes],[kas],[kes],[mas], [mes],[sas],[ses],[ras], [res], [tas], [tes].

CCVC -> [fras],[fres], [kras],[kres],[tras], [tres].

El lenguaje L también presenta dos restricciones:

- 1.- La r no puede ser usado para iniciar una frase.
- 2.- La r no puede ir después de una s

El programa recibirá como entrada una cadena de caracteres ASCII (sin caracteres distintos a los utilizados para los sonidos a generar, y los espacios en blanco se interpretan como pausas) y el nombre del archivo .wav a generar. Pueden darse dos casos en los que se modifique la prosodia y esos son:

- 1.- La acentuación de una vocal, la cual se indica introduciéndola en mayúscula (A)
- 2.- La prosodia de pregunta, que se indica añadiendo el carácter '?' al final de la cadena de caracteres.

Creación del inventario de sonidos

El inventario de dífonos del sistema estará compuesto por los siguientes elementos:

- /ef/, /Ef/, /af/, /Af/
- /ek/, /Ek/, /ak/, /Ak/
- /em/, /Em/, /am/, /Am/,
- /er/, /Er/, /ar/, /Ar/
- /es/, /Es/, /as/, /As/
- /et/, /Et/, /at/, /At/
- /sf/, /sk/, /sm/, /ss/, /st/
- /-e/, /-E/, /e-/, /E-/
- /-a/, /-A/, /a-/, /A-/
- /fe/, /fE/, /fa/, /fA/
- /ke/, /kE/, /ka/, /kA/
- /me/, /mE/, /ma/, /mA/
- /re/, /rE/, /ra/, /rA/
- /se/, /sE/, /sa/, /sA/
- /te/, /tE/, /ta/, /tA/
- /-f/, /fr/
- /-k/, /kr/
- /-t/, /tr/.
- /-s/, /s-/
- /-m/
- /ee/, /EE/, /Ee/, /eE/
- /aa/, /AA/, /Aa/, /aA/
- /ae/, /AE/, /Ae/, /aE/
- /ea/, /EA/, /Ea/, /eA/
- /--/

Grabación y creación de los .wav de cada sonido

Para la grabación de dichos difonos, he utilizado la misma técnica utilizada en la memoria del compañero Manuel Moreno Benabat, del año pasado.

Por lo que he ido grabando palabras que contengan los difonos que vamos a utilizar.

Por ejemplo, grabamos las palabras “asma”, de las cuales podemos obtener los siguientes difonos:

-{asma} ---> /-a/, /as/, /sm//ma/ /a-/.

Voy a poner un ejemplo de cómo he grabado todos los sonidos del inventario paso por paso:

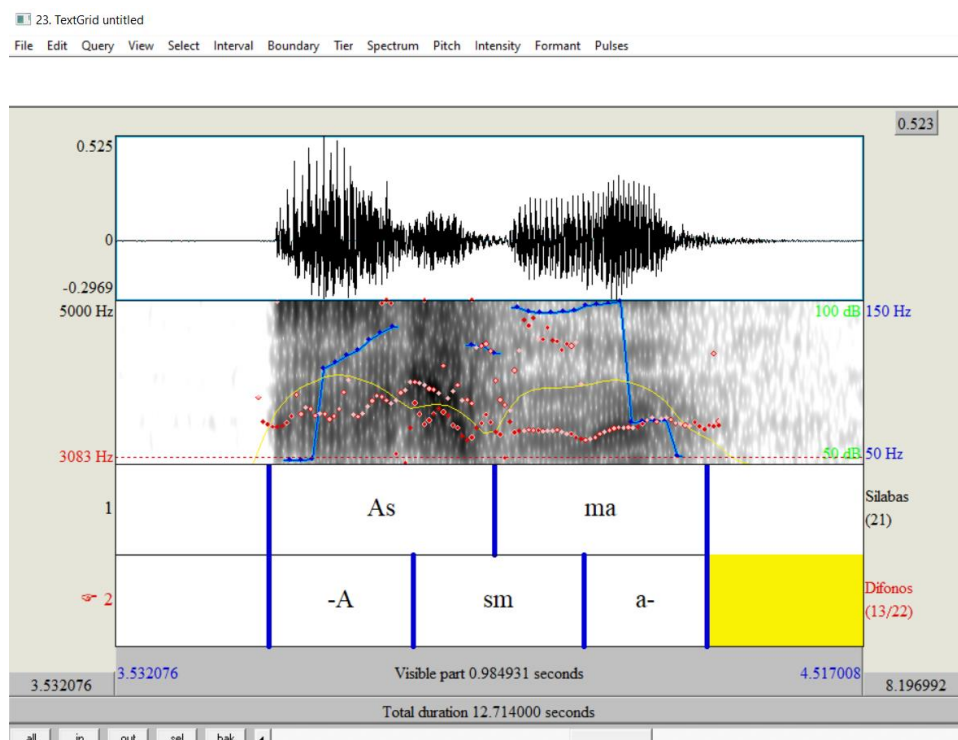
Grabamos el sonido con **New -> Record Mono Sound** , con Sampling frequency 16000Hz.

Luego le añadimos al sonido un TextGrid con **Annotate -> To TextGrid**, el cual lo creamos con los campos Silabas y Difonos.



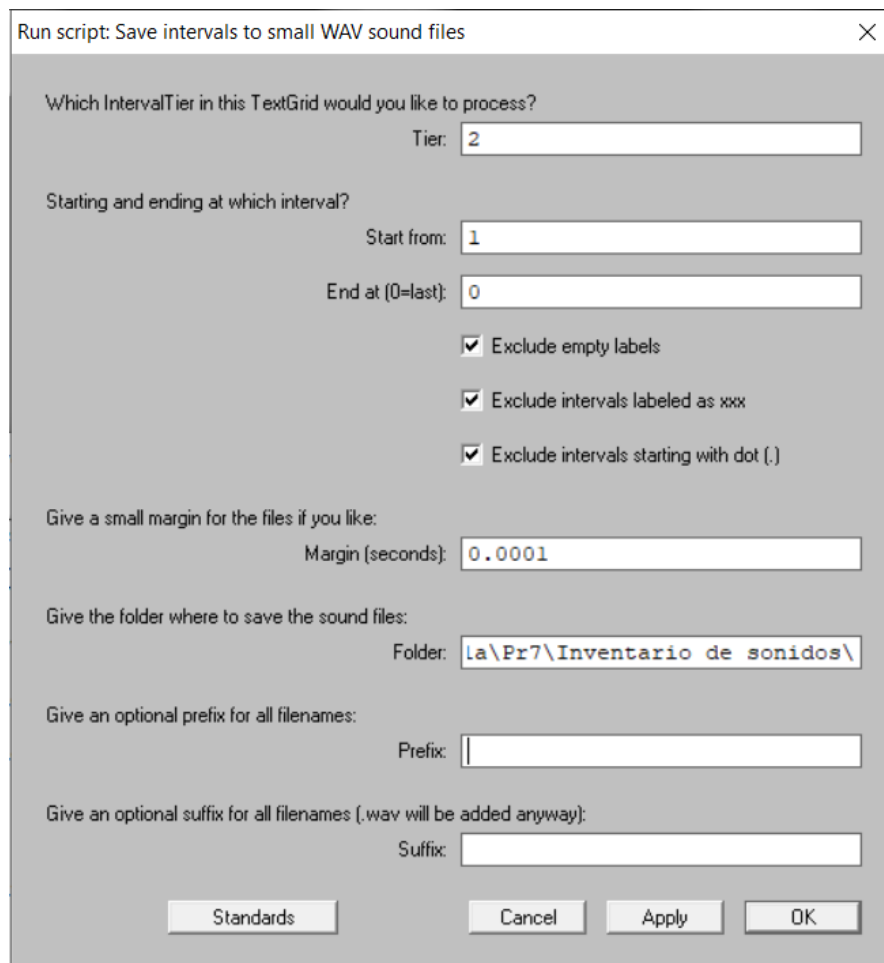
Una vez tenemos ambos elementos creados, los seleccionamos y nos vamos a **View & Edit** lo que nos abrirá la ventana que usaremos para separar y sacar cada difono necesario para la realización de la práctica.

En este caso concreto, he grabado la palabra asma, de la cual, podemos coger los difonos de la que está compuesta, como se puede ver en la imagen. La palabra asma se podría descomponer en más difonos, como he hecho arriba de forma teórica, lo que pasa que, si intentaba sacar todos los difonos posibles, estos resultaban de una calidad mucho menor que grabando solo los mostrados en la imagen, y lo importante es la calidad del sonido, por lo tanto, casi todas las palabras usadas para sacar los difonos podrían haber sido descompuestas en más difonos, pero esto hubiera bajado más la calidad de estos.



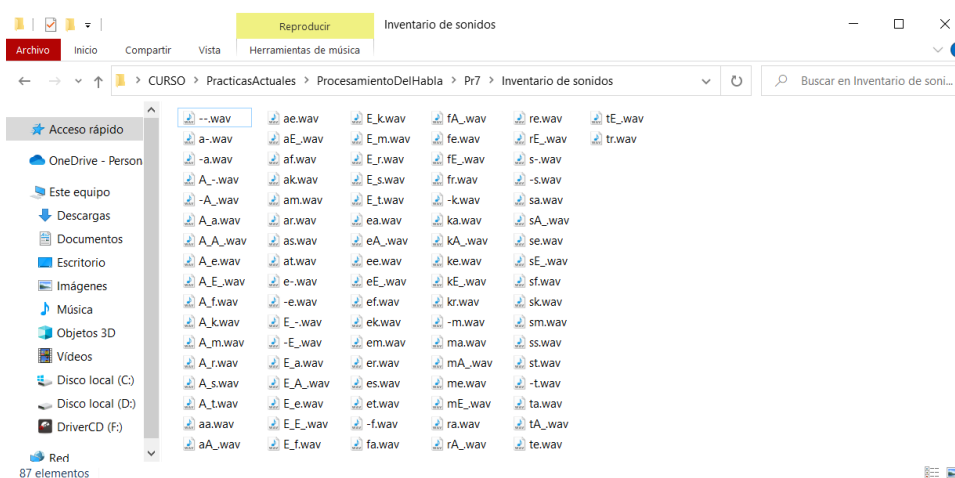
Una vez ya tenemos los difonos separados y comprobamos que en el tramo seleccionado se escuche bien el difono señalado, después, cerramos la ventana y ejecutamos el script facilitado con la practica (*save_labeled_intervals_to_wav_sound_files.praat*).

Usamos **Open -> Read From File** para abrir el script y luego lo ejecutamos con **Run -> Run**, lo que nos abrirá una ventana para que cambiemos la configuración.



Procesamos el campo 2 que en este caso es el campo Difonos, los cuales queremos procesar, ponemos el Margin a 0.0001 como se nos indica en el enunciado de la practica e indicamos la ruta en la cual queremos que se nos guarden los archivos procesados.

Una vez hecho esto y comprobado que el .wav resultante se escucha bien, lo añadimos al inventario de sonidos.



Creación del programa capaz de concatenar los archivos generados en el inventario

Nuestro programa está formada por diferentes clases, entre ellas la clase gramática que se encarga de comprobar las reglas del lenguaje.

```
private static boolean cumpleAlfabeto(String palabra) {
    boolean cumple = true;
    int i = 0;
    while(i <= palabra.length()-1 && cumple == true) {
        if(palabra.charAt(i) == 'e' || palabra.charAt(i) == 'E' || palabra.charAt(i) == 'a' || palabra.charAt(i) == 'A' || palabra.charAt(i) == 'm' ||
           palabra.charAt(i) == 'r' || palabra.charAt(i) == 's' || palabra.charAt(i) == 'k' ||
           palabra.charAt(i) == 't' || palabra.charAt(i) == 'f' || palabra.charAt(i) == 'l' ||
           palabra.charAt(i) == 'y' || palabra.charAt(i) == ' ' )
            i++;
        else
        {
            cumple = false;
            System.out.println("No cumple el alfabeto de la gramática");
        }
    }
    return cumple;
}
```

Comprueba que todas las letras están comprendidas entre los fonos utilizados para la práctica.

```
private static boolean cumpleRestriccionesKys(String palabra) {
    //no puede empezar por r |
    // una r no puede suceder a una s
    boolean cumple = true;
    int i = 0;

    if(palabra.charAt(0) == 'r')
    {
        cumple = false;
        System.out.println("No cumple, empieza por r..");
    }

    while(i < palabra.length()-1 && cumple == true) {

        if(palabra.charAt(i) == 'r' && palabra.charAt(i+1) == 's')
        {
            cumple = false;
            System.out.println("No cumple las restricciones, r no puede suceder a una s.");
        }
        i++;
    }
    return cumple;
}
```

Se encarga de comprobar que no se den los casos de empezar por r, ni que una r vaya después de una s.

```
private static boolean cumpleRestriccionesSilabas(String palabra) {
    boolean cumple = false;
    String r = palabra.replaceAll("([aeAE]|([mrs][aeAE](s?))|([kft](r?)[aeAE](s?)))+", "");
    if (r == "") {
        cumple = true;
    }
    else
    {
        cumple = false;
        System.out.println("No cumple las restricciones de las silabas");
    }
    return cumple;
}
```

Se encarga de componer bien las silabas en el lenguaje.


```

public static boolean cumpleReglas(String frase){
    StringTokenizer st = new StringTokenizer(frase, "( )+");
    boolean cumple = true;
    while(st.hasMoreElements() && cumple==true)
    {
        String palabra = st.nextToken();
        if(!cumpleRestriccionesRyS(palabra) || !cumpleAlfabeto(palabra) || !cumpleRestriccionesSilabas(palabra))
            cumple = false;
    }
    return cumple;
}

```

Se encarga de comprobar que todas las restricciones antes mencionadas, funcionan bien en conjunto.

```

public static void main(String[] args) {
    String entrada = " kasa se kae ";
    String salida = "fra.wav";

    // elimino espacios delante y detras, pero no los espacios interiores, de la entrada y del archivo salida
    entrada = entrada.trim();
    salida = salida.trim();

    //si la entrada cumple las reglas de la gramatica entonces
    if(Gramatica.cumpleReglas(entrada) == true) {

        // elimino el punto de fin o la interrogacion
        for (int i = 0; i < entrada.length(); i++) {
            if(entrada.charAt(i) == '?' || entrada.charAt(i) == '.')
            {
                entrada = entrada.substring(0, i);
            }
        }

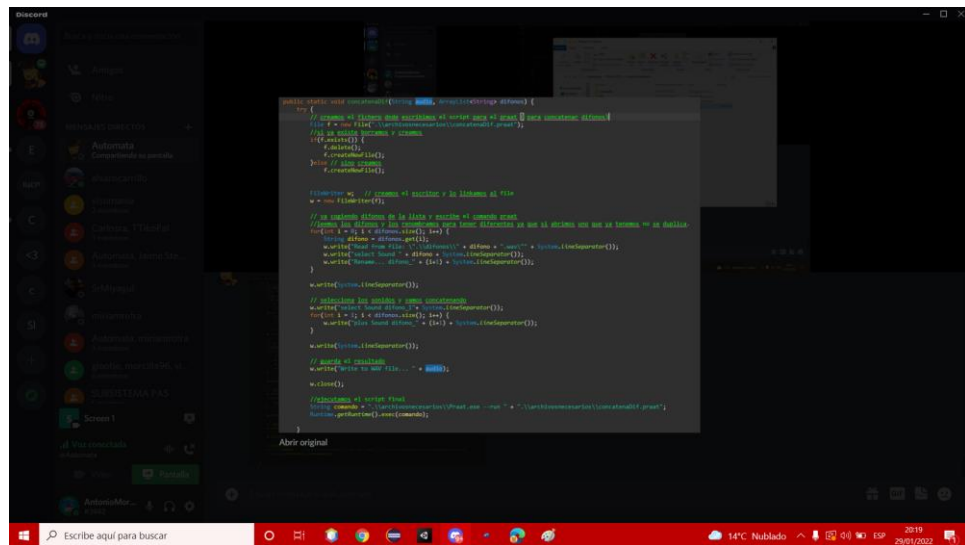
        //vuelvo a quitar los espacios delante y detras despues de quitar el . o ?
        entrada = entrada.trim();
        // en los espacios intermedios los sustituyo por --,
        //asi podemos identificar su sonido correspondiente para cuando concatenemos los difonos
        entrada = entrada.replaceAll("( )+", "--");

        //Pongo guiones delante y detras de la entrada para ayudarme luego a seleccionar
        //los difonos que no tienen nada delante ni detras.
        entrada = "--" + entrada + "--";

        //Voy de dos en dos sacando difonos y añadiendolos a una lista de difonos.
        //al mismo tiempo convierto las mayusculas en mayusculas_ para referirme a las vocales acentuadas
        ArrayList<String> lDifonos = new ArrayList<String>();
        for(int i = 0; i < entrada.length() - 1; i++) {
            String s = entrada.substring(i,i+2);
            s = s.replaceAll("E", "E_");
            s = s.replaceAll("A", "A_");
            lDifonos.add(s);
        }

        //concateno los difonos de la lista y genero el script
        Scripts.concatenaDif(salida, lDifonos);
    }
}

```



The screenshot shows a Discord chat window on the left with a list of servers and a search bar. Overlaid on the chat is a code editor window displaying C++ code. The code defines a function `fibonacci` that calculates the nth Fibonacci number using a loop and a vector to store the sequence. The code is as follows:

```
int fibonacci(int n) {  
    if (n < 0) return -1;  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
    vector<int> fib;  
    fib.push_back(0);  
    fib.push_back(1);  
    for (int i = 2; i <= n; i++) {  
        fib.push_back(fib[i-1] + fib[i-2]);  
    }  
    return fib[n];  
}
```

The code editor also shows a file explorer on the left with a file named `fibonacci.cpp`. The bottom of the screenshot shows a Windows taskbar with the date and time 20/01/2022 14:00.