UTN – Regional Buenos Aires

Sintaxis y Semántica de los Lenguajes
Curso K2102

Trabajo Práctico N°1

Generador de lenguajes

Grupo N°10

Apellido y Nombre	Legajo	Correo	GitHub
Baudo Sofia	213.498-6	sbaudo@frba.utn.edu.ar	SofiaBaudoUTN
Castro Planas Ignacio	213.579-6	icastroplanas@frba.utn.edu.ar	nacho-castro
Petrocelli Azul Martin	213.999-6	apetrocelli@frba.utn.edu.ar	AzulPetrocelliUTNBA
Bertella Tomas Emiliano	213.511-5	tbertella@frba.utn.edu.ar	berty-tb

Fecha de Presentación: 18/06/2024

Informe de Trabajo Práctico

Introducción

El presente trabajo consiste en realizar un programa en lenguaje C que permita generar aleatoriamente palabras de un lenguaje regular a partir de una gramática ingresada por el usuario.

El programa debe poder:

1. Ingresar una gramática: esto quiere decir que se debe poder ingresar la cuatrupla: símbolos terminales, símbolos no terminales, producciones y axioma.
2. Validar si la gramática es regular o no: El programa no debe permitir el ingreso de gramáticas que no sean regulares.
3. Generar aleatoriamente palabras del lenguaje, cada vez que genera una palabra nueva debe mostrar el proceso de derivación que se realiza. Puede mostrar una derivación horizontal.

Es importante aclarar que este informe tiene como objetivo servir como guía de usuario a la hora de utilizar el programa. Se incluyen nuestras conclusiones, además de la información relevante a la materia. En caso de querer conocer en extensión el funcionamiento interno, se recomienda ver el código disponible en GitHub: https://github.com/utn-frba-ssl/24-102-10

Condiciones iniciales

Toda Gramática Formal es una cuatrupla. Para ello, definimos un struct *'gramática'* el cual se compone de un arreglo de *terminales*, arreglo de *no-terminales*, un *axioma* y una matriz de *producciones*.

```
#define MAX_TERMINALES 3

#define MAX_NOTERMINALES 3

#define MAX_PRODUCCIONES 3

#define LONGITUD_PRODUCCION 6

//Todos estos valores se pueden modificar fácilmente a gusto antes
de ejecutar gracias a los #define.

typedef struct gram

{

    char producciones[MAX_PRODUCCIONES][LONGITUD_PRODUCCION];

    char noTerminales[MAX_NOTERMINALES];

    char terminales[MAX_TERMINALES];

    char axioma;

} gramatica;
```

Se decidió limitar a 3 el arreglo de terminales, los cuales deben ser caracteres ASCII de 'a' hasta 'z' (minúsculas). También se limitó a 3 el arreglo de no-terminales, los cuales deben ser caracteres ASCII de 'A' hasta 'Z' (mayúsculas).

Toda producción está formada por tres partes: el lado izquierdo, el lado derecho, y la flecha, que indica que el lado izquierdo de la producción "produce" (o es reemplazado por o equivale a) el lado derecho.

Ej: S -> aT

Las **gramáticas regulares** tienen como característica que sus producciones tienen las siguientes restricciones:

- El lado izquierdo debe tener un solo no-terminal.
- El lado derecho debe estar formado por un solo terminal o un terminal seguido por un no-terminal.

- A pesar de que algunos autores aceptan la inclusión de las producciones epsilon en las gramáticas regulares, decidimos no aceptar la inclusión de la cadena vacía, no se permite la producción S $\rightarrow \varepsilon$.

Por lo tanto, toda producción de una gramática regular posee una longitud máxima de **6** caracteres (se incluye el carácter nulo '\0'). El usuario puede ingresar hasta 3 producciones distintas. Esto conforma una *matriz producciones* de 3x6, donde cada fila corresponde a una producción diferente, y cada columna tiene sus caracteres correspondientes.

Ingreso de información

a. Terminales y No-Terminales

Mediante las funciones:

```
ingresarTerminales(g);

ingresarNoTerminales(g);
```

El usuario podrá ingresar 3 terminales diferentes, los cuales deben ser caracteres ASCII de 'a' hasta 'z' (minúsculas). Luego, la cantidad de 3 no-terminales diferentes, los cuales deben ser caracteres ASCII de 'A' hasta 'Z' (mayúsculas). No se permite la repetición de caracteres.

b. Axioma

```
while (verificarAxioma(g) == false);
```

Aquí se valida que el Axioma elegido por el usuario corresponda a alguno de los no-terminales ingresados anteriormente. No se permite ingresar Axiomas desconocidos.

c. Producciones

En caso de estar todo correcto, se le pide al usuario que ingrese 3 producciones. Recordemos que deben ser regulares, para ello deben ser de la forma 'X->xX', 'X->Xx' o 'X->x'.

Es importante conservar la estructura, comenzando siempre en un sólo no-terminal. Le sigue un meta-símbolo de dos caracteres *(->)* y, por último, el lado derecho que corresponda.

Romper la estructura recomendada puede perjudicar el funcionamiento del código, e incluso no cumplir con las características de una Gramática Regular.

La siguiente función hace la verificación:

```
bool verificarProduccion(gramatica *g, int fila)
```

Respecto al lado derecho, una gramática regular no permite la repetición de terminales o no-terminales seguidos. *Por ejemplo, 'S->aa' NO es regular.*

Tampoco es regular una producción que posee un sólo terminal del lado derecho. *Ejemplo, 'S->T'.*

Por último, la función verifica si la producción se realizará a izquierda o derecha, debiendo ser todas las producciones por el mismo lado. Esto mediante:

```
bool verificarLinealidad (gramatica *g, int fila)
```

d. Validación

En caso de ingresar información errónea en alguno de los anteriores pasos, el programa solicitará al usuario que ingrese los datos correctos hasta que la condición se cumpla.

Esto es gracias a las funciones de

```
bool esNoTerminal(char posibleNoTerminal)

bool esTerminal(char posibleTerminal)

bool verificarNoTerminal(gramatica *g, char noTerminal)

bool verificarTerminal(gramatica *g, char terminal)

bool esRegularPorDerecha(gramatica *g, int fila, int columna)

bool esRegularPorIzquierda(gramatica *g, int fila, int columna)
```

Las dos primeras verifican que el carácter ingresado se encuentre en el rango ASCII correspondiente.

Las dos siguientes comprueban si ya existe el carácter en la gramática, permitiendo evitar repeticiones o el ingreso de caracteres desconocidos para la cuatrupla.

Las dos últimas verifican que se cumpla la linealidad de la producción. Deben ser todas siempre por derecha o izquierda. Si la primera producción que se ingresa posee únicamente un terminal, por defecto, se la considera regular por derecha.

Generación de palabras

Una vez definida correctamente la gramática, el programa comienza a generar palabras aleatorias del lenguaje. Esto es mediante la función:

```
void derivacion(gramatica *g)
```

El proceso de derivación se realiza sobre una cadena con un máximo ya definido (en este caso de 2048 caracteres), la cual se inicializa con el Axioma seleccionado anteriormente.

```
char cadenaDerivacion[MAX_CADENA];

    int longitudCadena = 1;

    //Comenzamos siempre por el Axioma

    cadenaDerivacion[0] = g->axioma;

    cadenaDerivacion[1] = '\0';

    printf("Cadena inicial: %c\n", cadenaDerivacion[0]);
```

La función debe buscar el Axioma en la matriz de producciones, gracias a la función verificarNoTerminal() nombrada anteriormente. Para ello recorremos las posiciones [i][0] de la matriz, siendo ‘i’ la fila de la producción correspondiente al no-terminal buscado. Se toma el ‘0’ ya que toda producción (del lado izquierdo) tiene como primer elemento al no-terminal.

Una vez encontrado el Axioma y, en consecuencia, su correspondiente producción, damos por hecho que el lado derecho de ésta comienza en la posición ‘3’ del arreglo. Reemplazamos el Axioma por la cadena del lado derecho. Se aplicará el mismo procedimiento para las producciones siguientes.

Ejemplo:

S->aT (5 caracteres).

**Véase que el lado derecho comienza en el 4to carácter (posterior a 'S->')*

T->b (4 caracteres).

Axioma = S.

Derivación:

1. *S*
2. *aT*
3. *ab*

e. Selección Aleatoria

Existe la posibilidad de que haya más de una producción que comience por el mismo no-terminal. En ese caso es tarea del programa elegir de forma aleatoria alguna de las diferentes opciones.

Ejemplo:

1. *S->a*
2. *S->b*

**El programa elige de forma aleatoria la posición 2: 'S->b', formando la palabra 'b'.*

Para ello, se dispone de la función:

```
int seleccionarAleatorio(gramatica *g, char noterminal)
```

La cual selecciona una fila aleatoria de la matriz producciones que contenga una producción que inicie en el no-terminal esperado. Gracias a *rand* es posible elegir un número aleatorio dentro de un rango determinado por la cantidad de producciones que inicien con el noTerminal que se le solicite.

```
 srand(time(NULL)); //esto en el main, para evitar errores

    int randomIndice = rand() % cantidad; //y esto dentro de la
funcion mencionada
```

f. Sustituir caracteres

La función *derivación* trabaja en paralelo sobre *nuevaCadena*, la cual luego se sustituye en la *cadenaDerivacion* original. Todo esto mediante *strcpy*.

```
strcpy(cadenaDerivacion, nuevaCadena);

  longitudCadena = strlen(cadenaDerivacion);

  printf("Cadena derivada: %s\n", cadenaDerivacion);
```

g. Repetición

Finalmente, el usuario puede ejecutar tantas derivaciones como quiera, para comprobar que todas se generan de forma aleatoria bajo la cuatrupla formulada anteriormente.

```
int flag = 1;
    do
    {
        derivacion(g);
        printf("\nPresione 1. para derivar otra vez\n0. Salir\n");
        scanf("%d", &flag);
    } while (flag == 1);
```

Conclusión

El programa desarrollado permite la creación y validación de una gramática regular y la generación de palabras aleatorias del lenguaje definido por dicha gramática. Esto se logró mediante funciones que aseguran la regularidad de la gramática y la correcta derivación. Se buscó que la interfaz de usuario sea simple, con un correcto manejo de errores e indicaciones claras de lo que va ocurriendo en ejecución.