LENGUAJES FORMALES

PRÁCTICA 2:

ANALIZADOR SINTÁCTICO CON BISON

Alumnes: Bové Gómara, Marc
López Mellina, Alejandro
Mege Barriola, Gwenaëlle

Grado: 3ro Ingeniería Informática

Curso: 2017/2018

Profesor: Riaño Ramos, David

Índice

Introducción

El objetivo de esta práctica es conseguir realizar un Analizador Sintáctico con Bison e integrarlo con el Analizador Léxico realizado en la práctica anterior. Los apartados a realizar son los detallados a continuación:

1. Dado un AF (determinista o indeterminista) construir una función transición en C que implemente la función de transición del AF.
2. Un alfabeto sin símbolos debe dar el error “[ERROR] El alfabeto debe contener uno o más símbolos”.
3. Un AF con ningún o más de un estado inicial debe dar el error “[ERROR] Los Autómatas Finitos solo deben tener un estado inicial”.
4. Un AF sin estados finales debe dar el error “[ERROR] Los Autómatas Finitos deben tener algún estado final”.
5. El uso de números de estado o símbolos inexistentes en una transición debe dar el error “[ERROR] El estado %i de la transición (%i , %c ; %i) es desconocido” o “[ERROR] El símbolo %c de la transición (%i , %c ; %i) es desconocido”, respectivamente.
6. Si en la declaración del alfabeto hay un símbolo repetido, no tenerlo en cuenta y lanzar un aviso de “[AVISO] El símbolo %c ya existe”.
7. En el momento en que se detecte que es un AF no determinista se debe avisar sólo una vez “[AVISO] Se ha detectado que el AF es no determinista”.
8. (Opcionalmente) Se pide acabar de implementar el programa C que, unido a la función transición construida por BISON, permita ejecutar completamente el AF. Este programa, que será siempre el mismo, estará en un fichero af.c que incluirá el fichero af.h (#include "af.h") generado por nuestro programa BISON y que contendrá la función transición.

Implementación

Modificaciones en el archivo AF.l:

- Se ha cambiado la declaración de los números de [0-9]* a 0|[1-9][0-9]+ lo que no permite que se identifiquen como distintos los números 001 y 1 que a la hora de pasarlos a enteros se tomarán como iguales.
- Se ha suprimido la declaración de las transiciones ya que se trata de un trabajo que se podrá realizar en el archivo bison.
- Se ha cambiado todas las funciones “printf()” del código por “return()” para pasarle al archivo bison lo que tomará como “tokens”.
- Se ha añadido que al identificar un elemento de tipo NUM (número) se pasa el elemento leído a la estructura car (en el fichero bison) y se retorna el token SIMB para identificarlo como símbolo. Esto permite pasar todos los caracteres de yytext a la estructura creada en bison. Este cambio se ha realizado de la misma manera con los elementos identificados como símbolos.
- Se ha realizado algunas pruebas de errores para no permitir al usuario introducir símbolos en vez de números a la hora de poner los estados, los finales e inicial. En caso de equivocación del usuario se mostrará:

  “[ERROR]: Inicial no puede tener símbolos”, "[ERROR]: Estados no puede tener símbolos" o "[ERROR]: Finales no puede tener símbolos” según el origen del error.

- Se ha añadido un contador del número de líneas que recibe el fichero.
- Por último, se ha suprimido el main de este fichero.

Fichero AF.y:

- Se ha creado una estructura en la que se guarda una
- Se han definido los tokens `ALFABETO`, `ESTADOS`, `TRANSICIONES`, `INICIAL`, `FINALES`, `<car>SIMB`, `ERROR`, `ABRIR`, `CERRAR`, `COMA`, `PAR_A`, `PUNTOCOMA`, `PAR_C`.

  Cada uno de estos tokens será el utilizado por el archivo flex para retornar al fichero bison una vez identificados cada uno de estos tipos de elemento.

- Se ha definido un *autómata finito* de la manera siguiente:

```
af:          alfabeto estados transiciones inicial finales ;
```

Lo que indica que un autómata finito estará compuesto por una estructura de tipo alfabeto, estados, transiciones, inicial y finales.

- Se ha definido un *alfabeto* de la manera siguiente: :

```
alfabeto:    ALFABETO ABRIR lsimbolos CERRAR;
```

Lo que indica que un alfabeto está compuesto por el token ALFABETO seguido por el token ABRIR, el símbolo no terminal lsimbolos y el token CERRAR.

- Se ha definido *lsimbolos* de la manera siguiente:

```
lsimbolos :  { printf("[ERROR]: El alfabeto debe contener uno o
             más símbolos\n"); } /* (2) */
             | SIMB COMA lsimbolos      { simbRepeated($1); }
             | SIMB                     { simbRepeated($1); };
```

Lo que indica que lsimbolos estará compuesto por un token SIMB seguido por COMA y lsimbolos o simplemente el token SIMB. En caso de que esté vacío, se mostrará el error perteneciente al del apartado **(2)** del enunciado.

Para cada símbolo leído se llamará a la función simbRepeated, ésta recibirá por parámetro el símbolo leído y lo comparará con todos los leídos anteriormente para, en caso de detectarse que el símbolo está repetido, mostrar un aviso por pantalla tal y como se indica en el apartado **(6)** del enunciado. El código de esta función se muestra a continuación:

```
void simbRepeated(char *simb){
       int i = 0;
       bool found = false;
       while (i < num_simbols && !found) {
              if (!strcmp(alf[i], simb)){
                     found = true;
                     printf("[AVISO]: EL símbolo %s ya existe\n",
                     simb); /* (6) */}
              i++;
       }
       if (!found)
              alf[num_simbols++] = simb;
}
```

- Se ha definido *estados* de la manera siguiente:

```
estados :     ESTADOS ABRIR SIMB CERRAR;
```

Lo que indica que estados está compuesto por el token ESTADOS seguido por el ABRIR, SIMB y CERRAR.

- Se ha definido *transiciones* de la manera siguiente:

```
transiciones: TRANSICIONES ABRIR ltransicion CERRAR;
```

Lo que indica que transiciones está compuesto por el token TRANSICIONES seguido por el token ABRIR, el símbolo no terminal ltransicion y el token CERRAR.

- Se ha definido *ltransicion* de la manera siguiente:

```
ltransicion : trans COMA ltransicion
            | trans;
```

Lo que indica que ltransicion está compuesto por el símbolo no terminal trans, el token COMA y el símbolo no terminal ltransicion o simplemente por el símbolo no terminal trans.

- Se ha definido *trans* de la manera siguiente:

```
trans: PAR_A SIMB COMA SIMB PUNTOCOMA SIMB PAR_C
       {comprTransicion($2, $4, $6);};
```

Lo que indica que trans estará compuesto por un token PAR_A seguido por un SIMB, COMA, otro SIMB, PUNTOCOMA, otro SIMB i PAR_C.

Los tres símbolos de la transición se pasaran por parámetro a la función comprTransicion(). Ésta se encargará de comprobar que los símbolos corresponden a estados y a símbolos del alfabeto válidos En caso afirmativo guardará la transición como una nueva y en caso negativo mostrar un mensaje de error tal y como se indica en el apartado **(5)** del enunciado.

```
void comprTransicion(char* estatI, char* simb, char* estatF){
      int i=0;
      bool trobat=false;
      int estatInici = atoi(estatI);
      int estatFinal = atoi(estatF);
      while (i<num_simbols && !trobat) {
            if (strcmp(alf[i], simb) == 0)
                  trobat = true;
            else
                  i++;
```

```
        }
        if(!trobat||(estatInici>num_states)||(estatFinal>num_states))
        printf ("[ERROR]: El simbolo %s de la transición (%d , %s ; %d)
        es desconocido\n", simb, estatInici, simb, estatFinal); /*(5)*/

        if(trobat && estatInici<=num_states && estatFinal<=num_states){
              i=0;
              bool trobat2=false;
              while (i<num_trans && !trobat2) {
                    if     ((strcmp(transi[i][0], estatI)==0) &&
                           (strcmp(transi[i][1],      simb)==0)      &&
                           (strcmp(transi[i][2], estatF)==0))
                           trobat2=true;
                    else
                           i++;
              }
              if (!trobat2) {
                    transi[num_trans][0]=estatI;
                    transi[num_trans][1]=simb;
                    transi[num_trans][2]=estatF;
                    num_trans++;
              }
        }
}
```

- Se ha definido *inicial* de la manera siguiente:

```
inicial : INICIAL ABRIR linicial CERRAR;
```

Lo que indica que inicial está compuesto por el token INICIAL seguido por el token ABRIR, el símbolo no terminal linicial y el token CERRAR.

- Se ha definido *linicial* de la manera siguiente:

```
linicial :    {
                    printf("[ERROR]: Los Autómatas Finitos solo deben
                    tener un estado final\n");
              } /* (3) */
              | SIMB COMA linicial
              {
                    printf("[ERROR]: Los Autómatas Finitos solo deben
                    tener un estado final\n");
              } /* (3) */
              | SIMB;
```

Lo que indica que linicial está compuesto por el token SIMB. En el caso que no haya ningún símbolo en este o de que linicial esté compuesto del token SIMB, COMA y el

símbolo no terminal linicial, se mostrará de la misma forma el error del apartado **(3)** del enunciado.

- Se ha definido *finales* de la manera siguiente:

```
finales : FINALES ABRIR lfinales CERRAR;
```

Lo que indica que finales está compuesto por el token FINALES seguido por ABRIR, el símbolo no terminal lfinales y el token CERRAR.

- Se ha definido *lfinales*de la manera siguiente:

```
lfinales :    {
                      printf("[ERROR]: Los Autómatas Finitos deben tener
                      algún estado final\n");
              } /* (4) */
              | SIMB COMA lfinales
              | SIMB ;
```

Lo que indica que lfinales está compuesto por el token SIMB seguido por COMA y lfinales o simplemente por SIMB. En caso de que no tenga ningún elemento, se mostrará el error del apartado **(4)** del enunciado.

- En cuanto al código de usuario, se ha actualizado el main:

```
int main(int argc, char **argv)
{
       if (argc > 1) /* if there is at least 1 argument apart from
       program name */
       {
              yyin = fopen(argv[1], "rt");

              if (yyin == NULL) /* if the file cannot be opened */
              {
                     printf("[ERROR]: Ha habido algún error en el
                     fichero\n");
              return 1; /* Return error code */
              }
       }
       else
              yyin = stdin;
       yyparse();
       transicion(); /* imprime la función creada por pantalla */
       //provaTransicion(); /* Redirige la salida a un fichero .c */
       printf("\nEl programa ha finalizado correctamente! Consulta los
       ficheros transicion.c y transicion.h para ver la función
       resultante.\n");
       fclose(yyin);
       return 0; /* Return succeed code */
}
```

En esta parte de código se llama a la función transicion(). Que para todas las transiciones al final de su ejecución se habrá mostrado por pantalla la parte equivalente al código de la tercera columna de la tabla del enunciado según si se trata de un Autómata Finito Determinista o un Autómata Finito No Determinista. Además, en caso de que se trate de un AFND se mostrará para antes del código un aviso indicando que se trata de un AF no determinista, equivalente al apartado **(7)** del enunciado.

```
void transicion() /* (1) */
{
	for (int i = 0; i < num_trans; i++){
		if (esDeterminista()) {
			if(i == 0)
				printf("\nint transicion(int estado, char
				simbolo) {\n\tint sig;");

			printf("\n\tif((estado==%d)&&(simbolo=='%s'))
			sig = %d;", atoi(transi[i][0]), transi[i][1],
			atoi(transi[i][2]));

			if(i == num_trans-1)
				printf("\n\treturn(sig);\n}\n");
		}
		else {
			if(i == 0){
				printf("[AVISO]: Se ha detectado que el AF
				es no determinista\n"); /* (7) */

				printf("\nint * transicion(int estado, char
				simbolo)		{\n\tstatic		int
				sig[num-estados+1], n=0;");
			}
			printf("\n\tif((estado==%d)&&(simbolo=='%s'))
			sig[n++] = %d;", atoi(transi[i][0]), transi[i][1],
			atoi(transi[i][2]));

			if(i == num_trans-1)
				printf("\n\tsig[n]=-1; /* centinella
				*/\n\treturn(sig);\n}\n");
		}
	}
}
```

Se ha reescrito la función yyerror de forma que nos indique el número de línea donde ha saltado el error. Además se ha añadido la opción `%define parse.error verbose` para que nos de más información del error.

```
void yyerror(char *s) {
	fprintf(stderr, "%d: Error: %s\n", num_lines, s);
	exit(1);
}
```

Finalmente, para la parte opcional **(8)** se ha reescrito la función transicion() como provaTransicion()de manera que dirija el texto a un fichero transicion.c y transicion.h en lugar de imprimir el texto por pantalla, creando así la función transicion() en C en función de si se ha detectado que el autómata finito era determinista o indeterminista. La función se muestra a continuación:

```
void provaTransicion() /* (1 y 8) */{
	FILE *fileH = fopen("transicion.h", "a");
	FILE *fileC = fopen("transicion.c", "a");
	fprintf(fileC, "#include ""transicion.h""\n");

	fprintf(fileH, "int transiciondet(int estado, char
	simbolo);\n");
	fprintf(fileH, "int * transicion(int estado, char
	simbolo);\n");
	fclose(fileH);

	for (int i = 0; i < num_trans; i++){
		if (esDeterminista()) {
			if(i == 0)
				fprintf(fileC, "\nint transicion(int
				estado, char simbolo) {\n\tint sig;");

			fprintf(fileC,"\n\tif((estado==%d)&&(simbolo=='%s'
			))sig  =%d;",  atoi(transi[i][0]),  transi[i][1],
			atoi(transi[i][2]));

			if(i == num_trans-1)
				fprintf(fileC, "\n\treturn(sig);\n}\n");
		}
		else {
			if(i == 0){
				printf("[AVISO]: Se ha detectado que el AF
				es no determinista\n"); /* (7) */

				fprintf(fileC,  "\nint  *  transicion(int
				estado,  char  simbolo)  {\n\tstatic  int
				sig[num-estados+1], n=0;");
			}
			fprintf(fileC,    "\n\tif((estado==%d)&&(simbolo==
			'%s'))    sig[n++]  =  %d;",  atoi(transi[i][0]),
			transi[i][1], atoi(transi[i][2]));

			if(i == num_trans-1)
				fprintf(fileC, "\n\tsig[n]=-1; /*
				centinella*/\n\treturn(sig);\n}\n");
		}
	}
	fclose(fileC);
}
```