TP Autómatas

Sintaxis y Semántica de los Lenguajes
K2003

Dr. [REDACTED] - Ing. [REDACTED]

Integrantes:

De Sousa, Agustín - 000.000-0
Dipietro, Guido - 000.000-0
Pellegrini, Pablo - 000.000-0
Verdun, Juan Cruz - 000.000-0

Universidad Tecnológica Nacional

Facultad Regional Buenos Aires
julio de 2020

Índice

1. Consigna

En cada caso, escriba un programa en ANSI C que pruebe la implementación realizada.

1. Elabore un AFD que reconozca el lenguaje de constantes enteras con signo para:
   - Constantes octales
   - Constantes hexadecimales

   (Agregue, además, el reconocimiento de sufijos)

2. Implemente una única solución que reconozca las constantes enteras en C descriptas en la gramática del lenguaje:
   - Decimales
   - Octales
   - Hexadecimales

3. Desarrolle un programa comando que reciba una cadena que puede representar una constante entera decimal con signo y, si lo representa, retorne el valor decimal de la misma

4. Desarrolle un programa comando que reciba una expresión aritmética simple y retorne su valor

Procederemos primero a detallar la teoría usada, y posteriormente a mostrar los programas internamente y su correcto funcionamiento en consola.

2. Teoría

Utilizaremos las siguientes definiciones de las constantes y sufijos, de la gramática del lenguaje ANSI C:

```
<constante entera> ->
    <constante decimal> <sufijo entero>? |
    <constante octal> <sufijo entero>? |
    <constante hexadecimal> <sufijo entero>?
<constante decimal> ->
    <dígito no cero> |
    <constante decimal> <dígito>
<dígito no cero> -> uno de
    1 2 3 4 5 6 7 8 9
<dígito> -> uno de
    0 1 2 3 4 5 6 7 8 9
<constante octal> ->
    0 |  <constante octal> <dígito octal>
<dígito octal> -> uno de
    0 1 2 3 4 5 6 7
<constante hexadecimal> ->
    0x <dígito hexadecimal> |
    0X <dígito hexadecimal> |
    <constante hexadecimal> <dígito hexadecimal>
<dígito hexadecimal> -> uno de
    0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F
<sufijo entero> ->
    <sufijo "unsigned"> <sufijo "long">? |
    <sufijo "long"> <sufijo "unsigned">?
<sufijo "unsigned"> -> uno de
    u U
<sufijo "long"> -> uno de
    l L
```

Todas las constantes tienen la forma de un lenguaje regular, por lo que pueden ser reconocidas por un autómata finito determinístico (AFD).

La expresión regular (ER) asociada a cada una de ellas es:[1]

Tipo	ER
Octal	`0.[0-7]`*`.(ε+ sufijo)`
Hexa	`0.(x+X).([0-9]+[a-f]+[A-F])`$^+$`.(ε+sufijo)`
Decimal	`[1-9].[0-9]`*`.(ε+sufijo)`

Siendo "sufijo" `= ((u+U+l+L)+(u+U+l+L).(u+U+l+L))` en todos los casos.

De igual forma, se puede definir un lenguaje regular que reconozca las operaciones aritméticas simples, que definimos como:

[1] Utilizamos la notación de 'rango numérico' de RegEx para simplificar la lectura: `[a-b]` significa todos los dígitos o letras entre a y b, incluyendo a y b, con $a < b$.

`[0-9]`$^+$ `. (operador) .` `[0-9]`$^+$

Con "operador"siendo uno de {+, -, *, /}, o bien modificarlo para no permitir la división por cero:

`([0-9]`$^+$`/[1-9][0-9]`*`)` + `([0-9]`$^+$`(operador-{/})[0-9]`$^+$`)`

Usaremos este segundo lenguaje para nuestros programas.

Es posible crear una implementación en C que lea la tabla de transiciones asociada a los autómatas que reconocen estos Lenguajes Regulares (LR).

Hemos modificado ligeramente la gramática para que se pueda permitir comenzar la palabra con un signo `-`, para indicar constantes negativas.
Si bien el símbolo `-` no pertenece a las constantes, sino que es un operador que posteriormente realiza una operación del tipo `0 - num` para asignar el valor negativo, nosotros lo incluimos como si fuera parte de la gramática de C porque así se pidió en el enunciado.

Se explicarán en las secciones posteriores los programas principales que resuelven las consignas planteadas. Igualmente, todos los archivos relacionados a este Trabajo Práctico están disponibles en el repositorio de GitHub que utilizamos (link en página 18, o click en "GitHub").

3. Autómatas

Procederemos a resolver los problemas de cada consigna, primero diseñando el autómata finito para cada lenguaje descrito en la sección 2.

3.1. Reconocedor de constantes tipo octal, hexadecimal y decimal

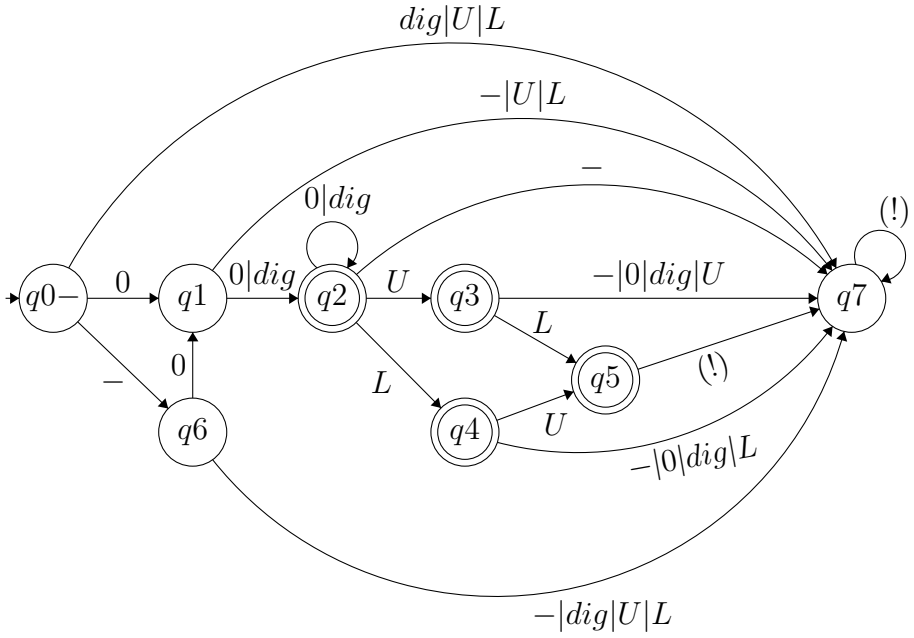
Definimos las siguientes tablas de transición, acompañada de sus diagramas de transición.[2]

El símbolo `"díg"` representa todos los dígitos distintos de '0' válidos para cada lenguaje; es decir, [1-9] para hexadecimal y decimal, y [1-7] para octal.

Constantes octales con sufijo signadas

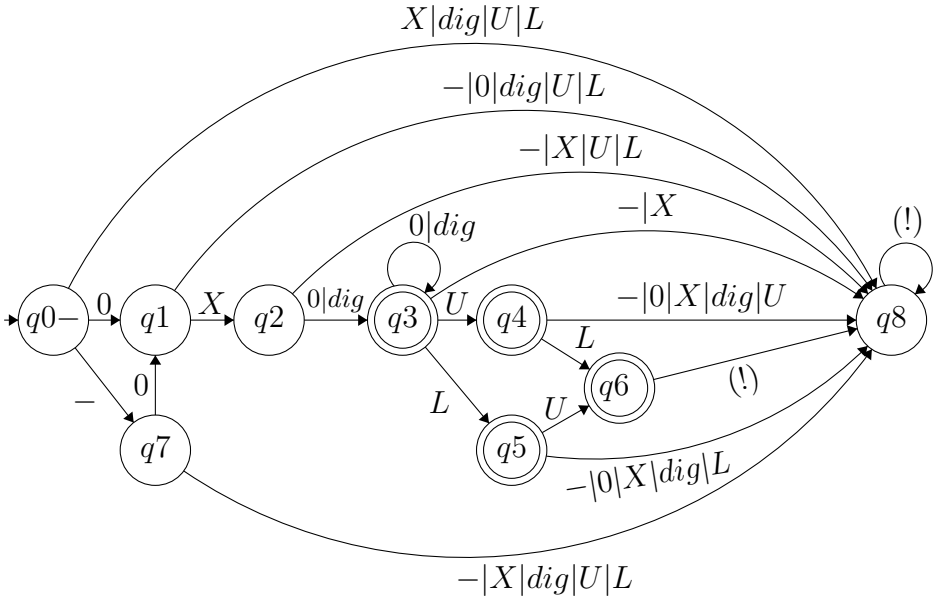
E	-	0	díg	u,U	l,L
q0-	6	1	7	7	7
q1	7	2	2	7	7
q2+	7	2	2	3	4
q3+	7	7	7	7	5
q4+	7	7	7	5	7
q5+	7	7	7	7	7
q6	7	1	7	7	7
q7	7	7	7	7	7

[2]Los dígrafos tienen solo transiciones por símbolos en mayúscula; en todos los casos, los caracteres `L, U, X` corresponden también a las transiciones por sus equivalentes en minúscula. De la misma manera, `(!)` representa cualquier carácter válido del lenguaje. Esto es únicamente una decisión estética.



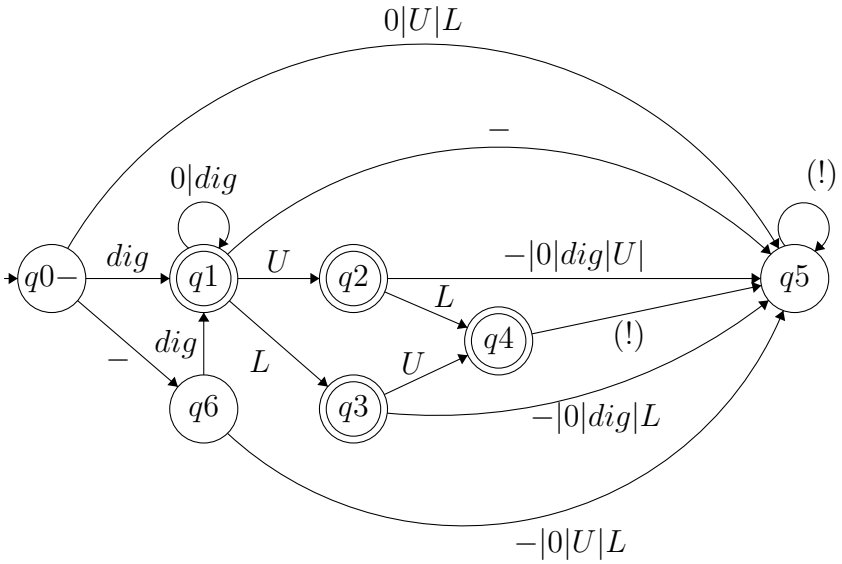
Constantes hexadecimales con sufijo signadas

E	-	0	x,X	díg	u,U	l,L
q0-	7	1	8	8	8	8
q1	8	8	2	8	8	8
q2	8	3	8	3	8	8
q3+	8	3	8	3	4	5
q4+	8	8	8	8	8	6
q5+	8	8	8	8	6	8
q6+	8	8	8	8	8	8
q7	8	1	8	8	8	8
q8	8	8	8	8	8	8



Constantes decimales con sufijo signadas

E	-	0	díg	u,U	l,L
q0-	6	5	1	5	5
q1+	5	1	1	2	3
q2+	5	5	5	5	4
q3+	5	5	5	4	5
q4+	5	5	5	5	5
q5	5	5	5	5	5
q6	5	5	1	5	5



Podemos implementar esto en C de la siguiente forma (ejemplo de implementación de autómata para constantes octales):

```c
int esPalabra (const char *cadena) {
    static int tt [8][5] = {
        // − | 0 | d | u | l
          {6, 1, 7, 7, 7}, //0−
          {7, 2, 2, 7, 7}, //1
          {7, 2, 2, 3, 4}, //2+
          {7, 7, 7, 7, 5}, //3+
          {7, 7, 7, 5, 7}, //4+
          {7, 7, 7, 7, 7}, //5+
          {7, 1, 7, 7, 7}, //6
          {7, 7, 7, 7, 7}  //7
    };
    int estado = 0; // Estado inicial
    unsigned int i = 0;
    int caracter = cadena[0];

    while ( caracter != '\0' ) {
        estado = tt[estado][columnaOctal(caracter)]; // Cambio de estado
        caracter = cadena[++i]; // Avanzo una posicion en la cadena
    }

```

```
22      if (estado == 5 || estado == 2 || estado == 3 || estado == 4) { //estados
        finales
23          return 1;
24      }
25
26      return 0;
27  }
```

Esta función lee una cadena carácter a carácter y de acuerdo al estado actual y a la tabla de transiciones cargada en la variable `tt`, modifica el estado actual.
Finalmente, al llegar al final de la cadena, si el estado actual es un estado final se retorna `1` indicando éxito, o `0`, indicando fracaso.

Se usa función auxiliar que retorna la columna correspondiente a un carácter dado de acuerdo a esta tabla de transiciones, como se puede ver en la línea 18 del ejemplo anterior.
Esta función, `columnaOctal()`, luce así.

```
1  int columnaOctal(int caracter) {
2      switch (caracter) {
3          case '-': return 0;
4          case '0': return 1;
5          case 'u': return 3;
6          case 'U': return 3;
7          case 'l': return 4;
8          case 'L': return 4;
9          case '1': return 2;
10         case '2': return 2;
11         case '3': return 2;
12         case '4': return 2;
13         case '5': return 2;
14         case '6': return 2;
15         case '7': return 2;
16     }
17 }
```

Finalmente, la ejecución principal sencillamente controla que no haya caracteres que no pertenezcan al alfabeto de este lenguaje, y de no haberlos, controla si pertenece al lenguaje usando `esPalabra()`.[3]

```
1  int verifica (char *s){
2      unsigned int i = 0;
3      for (i; s[i]; i++){
4          if(!(
5              (isdigit(s[i]) && (s[i] != '8' && s[i] != '9')) || //num del 0 al 6
6              s[i] == 'u' ||
7              s[i] == 'U' ||
8              s[i] == 'l' ||
9              s[i] == 'L' ||
10             s[i] == '-')){
11             return 0;
12         }
13     }
14     return 1;
```

[3]En este y todos los otros programas vemos el uso aparentemente innecesario de "strcpy" en el `main()` para almacenar el valor de la cadena en una variable auxiliar. Esto es el remanente de crear la variable `s1` con un valor asignado a mano para ayudar con la depuración del programa, y más adelante la 'perezosa' utilización de la misma para permitir el ingreso de argumentos por consola. Preferimos dejarlo así.

```
}

int main(int argc, char *argv[]){
    char s1[] = "";
    strcpy(s1, argv[1]);

    // Caso que tenga caracteres invalidos
    if (!verifica(s1)){
        printf("Caracteres invalidos!\n");
        return EXIT_FAILURE;
    };

    // Caso que pertenezca al lenguaje
    if (esPalabra(s1)){
        printf("Pertenece al lenguaje!\n");
        return EXIT_SUCCESS;
    }

    // Caso que no pertezca al lenguaje
    printf("No pertenece al lenguaje!\n");
    return EXIT_FAILURE;
}
```

Los otros programas (reconocedor de constantes hexadecimales y decimales) tienen exactamente la misma estructura, solamente cambiando la tabla de transiciones, la función `columna()`, y la función `verifica()` por lo que corresponda para reconocer cada alfabeto y lenguaje.

Se puede ver estos programas en funcionamiento en las imágenes 1 y 2 en la página 19, así como el programa que reconoce constantes decimales en la figura 4 (página 20).

Este último, como además retorna el valor numérico de la constante y esto corresponde a la resolución del problema desarrollado en la sección 3.3, se explicará oportunamente allí.

3.2. Programa que determina tipo de constante

Para realizar el programa que dada una constante reconoce de qué tipo es (si es válida), armamos una pequeña biblioteca con los autómatas del punto anterior. Veamos brevemente cómo luce el archivo *header*:

```
////////////////////////////// VERIFICADORES ////////////////////////////////

#ifndef VERIFICADORES
#define VERIFICADORES
// Funcion que verifica si los caracteres pertenecen al alfabeto
int verificaOctal (char *s);

// Funcion que verifica si los caracteres pertenecen al alfabeto
int verificaHexa (char *s);

// Funcion que verifica si los caracteres pertenecen al alfabeto
int verificaDecimal (char *s);

#endif

```

```
16 /////////////////////////// AUTOMATAS ////////////////////////////
17
18 #ifndef AUTOMATAS
19 #define AUTOMATAS
20 //Verifica si la cadena OCTAL pertenece al lenguaje
21 int automataOctal (const char *cadena);
22
23 //Verifica si la cadena HEXA pertenece al lenguaje
24 int automataHexa (const char *cadena);
25
26 //Este automata para decimales SI acepta sufijos con "u,U" si la cadena comienza
       con "-"
27 //Verifica si la cadena DECIMAL pertenece al lenguaje
28 int automataDecimal (const char *cadena);
29
30 #endif
```

Lo lógico es intentar evaluar si la constante es octal, hexadecimal, o decimal utilizando las funciones `automataOctal(), automataHexa(), automataDecimal()`.
Como estos tres lenguajes tienen intersección vacía entre ellos (es decir, no hay ninguna constante que pertenezca a más de uno de estos lenguajes a la vez), esto se puede hacer sin problemas.[4]

La definición de las funciones en el archivo `constantes.c` que corresponde a este encabezado está incluida en el anexo, página 15.

El programa principal simplemente consta de una serie de bucles `if` que evalúan la pertenencia de la constante ingresada como argumento a los lenguajes que definimos.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "constantes.h"
5
6  // Usa "automataDecimalConSufijov2.c" (reconoce decimales con - y u,U)
7
8  int esOctal(char cadena[]){
9    if(!verificaOctal(cadena)) return 0;
10   if(automataOctal(cadena)) return 1;
11   else return 0;
12 }
13 int esHexa(char cadena[]){
14   if(!verificaHexa(cadena)) return 0;
15   if(automataHexa(cadena)) return 1;
16   else return 0;
17 }
18 int esDecimal(char cadena[]){
19   if(!verificaDecimal(cadena)) return 0;
20   if(automataDecimal(cadena)) return 1;
21   else return 0;
22 }
23
24 int main(int argc, char *argv[]){
25   char cadena[] = "";
26   strcpy(cadena, argv[1]);
```

[4] El "0" debería reconocerse como constante octal de acuerdo a la gramática presentada, pero el valor de este número es en realidad el mismo en cualquier base. Además, no se tomó en cuenta este número para nuestra implementación de los autómatas, por lo que figuraría como "constante inválida" de ingresarse en este programa.

```
27
28    if(esDecimal(cadena)){
29      printf("%s","Es una constante decimal.");
30      return EXIT_SUCCESS;
31    }
32    if(esHexa(cadena)){
33      printf("%s","Es una constante hexadecimal.");
34      return EXIT_SUCCESS;
35    }
36    if(esOctal(cadena)){
37      printf("%s","Es una constante octal.");
38      return EXIT_SUCCESS;
39    }
40
41    printf("%s", "Es una constante invalida.");
42    return EXIT_FAILURE;
43  }
```

Entonces, al ingresar una constante, el programa mostrará en pantalla un mensaje indicando a qué tipo de constantes pertenece, o si no es válida en absoluto.
Este programa reconoce constantes hexadecimales, decimales y octales, en los tres casos soportando implementación signada y con sufijos.
Su funcionamiento en consola se puede ver en la imagen 3, en la página 20.

3.3. Programa que retorna valor decimal de constante decimal

Este programa fue realizado originalmente para la resolución parcial del punto 3.1, y fue posteriormente modificado para permitir el retorno del valor numérico de la constante ingresada como cadena de texto.

Si el programa lee un carácter, y el mismo es un dígito, entonces realiza la asignación

```
num = num*10 + (carácter - '0');
```

a una variable tipo entera `num` inicializada en 0.

Esto se vale de algunos "trucos":

- Multiplicar un entero por su base y sumarle una cifra equivale a realizar una concatenación de un símbolo al final de una cadena de caracteres.
- El valor ASCII de las cifras numéricas es consecutivo, por lo que al restarle `'0'` obtenemos el desplazamiento desde el 0, es decir, por definición, su valor numérico.

Además, si el primer carácter leído es el símbolo `-`, una variable llamada "signo"guarda el valor `-1`. Esa misma variable se inicializa en `1` para mantener el signo positivo en el caso de que la cadena represente un número positivo.

Posteriormente, al terminar la lectura de la cadena, se multiplica el valor almacenado en `num` por el valor almacenado en `signo` para conformar el valor final correcto.

Por lo tanto, el programa modificado para implementar el retorno del valor numérico de la constante luce como se muestra a continuación.

```
1  // Funcion que verifica si la cadena pertenece al lenguaje
2  int esPalabraDecimal (const char *cadena, int *num) {
3      static int tt [7][5] = {
4      {6,5,1,5,5}, //0-
5      {5,1,1,2,3}, //1+
6      {5,5,5,5,4}, //2+
7      {5,5,5,4,5}, //3+
8      {5,5,5,5,5}, //4+
9      {5,5,5,5,5}, //5
10     {5,5,1,5,5}  //6
11     };
12     int estado = 0; // Estado inicial
13     unsigned int i = 0;
14     int caracter = cadena[0];
15     int signo = 1;
16
17     while ( caracter != '\0' ) {
18         estado = tt[estado][columnaDecimal(caracter)]; // Cambio de estado
19         if(isdigit(caracter)) (*num) = (*num)*10 + (caracter - '0'); // Valor
       numerico
20         else if(caracter == '-') signo = -1;
21         caracter = cadena[++i]; // Avanzo una posicion en la cadena
22     }
23     (*num) = (*num)*signo; //Por si es negativo
24
25     if (estado==1 || estado==2 || estado==3 || estado==4) { //estados finales
26         return 1;
27     }
28
29     return 0;
30 }
31
32 int main(int argc, char *argv[]){
33     char s1[] = "";
34     strcpy(s1, argv[1]);
35
36     // Caso que tenga caracteres invalidos
37     if (!verificaDecimal(s1)){
38         printf("Caracteres invalidos!\n");
39         return EXIT_FAILURE;
40     };
41     // Caso que pertenezca al lenguaje
42     int num = 0;
43     if (esPalabraDecimal(s1, &num)){
44         printf("Pertenece al lenguaje!\n");
45         printf("Valor numerico: %d", num);
46         return EXIT_SUCCESS;
47     }
48     // Caso que no pertezca al lenguaje
49     printf("No pertenece al lenguaje!\n");
50     return EXIT_FAILURE;
51 }
```

En la línea 19 se realiza la actualización de `num` ante la presencia de un dígito, y en la 20 el análisis del signo. Finalmente, en la línea 23 se opera y se almacena el valor final signado.

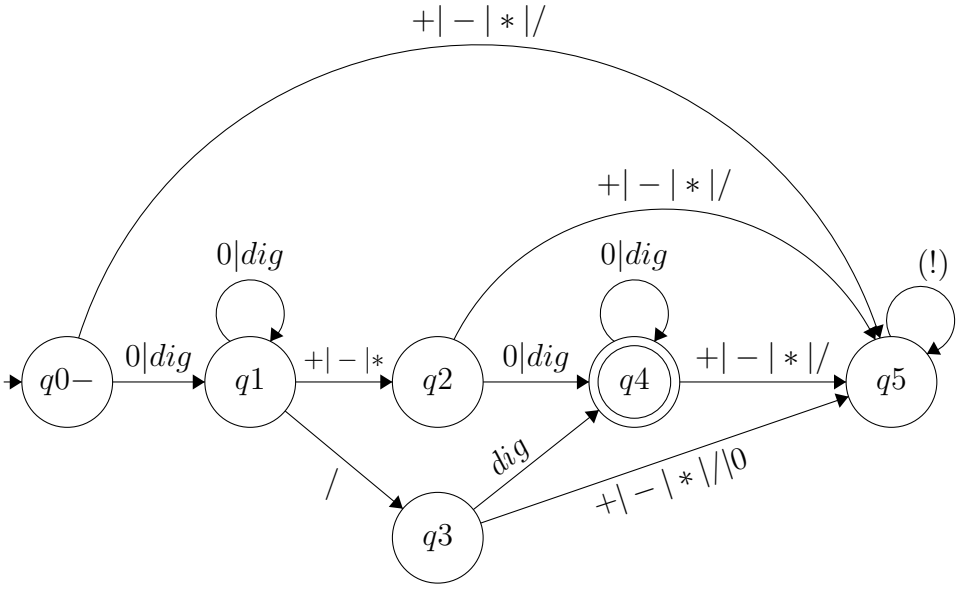
Su funcionamiento se muestra en la imagen 4, página 20.

3.4. Programa que resuelve operación aritmética simple

Primero y principal, elaboramos la tabla de transiciones del lenguaje que describimos en la sección 2. El alfabeto son los dígitos del 0 al 9, y las 4 operaciones básicas: `+, -, *, /`.

También incluimos el dígrafo.

E	díg	+	-	*	/	0
q0-	1	5	5	5	5	1
q1	1	2	2	2	3	1
q2	4	5	5	5	5	4
q3	4	5	5	5	5	5
q4+	4	5	5	5	5	4
q5	5	5	5	5	5	5



Utilizando una lógica similar a la del programa en la consigna "Programa que retorna valor decimal de constante decimal", pero ahora utilizando la información brindada por la variable `estado`, almacenamos los valores de ambos operandos.

Comenzamos con la definición de la función `opercionAritmetica()` que dada una cadena retorna el valor de evaluarla, si es una operación válida de la forma `[num][operando][num]`.
Se estructura prácticamente de la misma forma que las funciones `automataX()` de los reconocedores de octal, hexadecimal, y decimal.

La diferencia en este programa es que mientras el estado sea igual a 1, se almacenará el valor numérico en la variable `num1`, pero si el valor es en cambio 4, se almacenará en `num2` (ver líneas 24 y 26).

Asimismo, si el estado es el 2 o el 3, significa que se ingresó un operando, y este se almacenará en la variable `operador` (línea 25).

El autómata fue diseñado para que rechace todas las palabras que incluyan la secuencia `"/0"`, así que no es necesario incluir un bucle `if` que se fije en eso (línea 35).

Veamos el programa a continuación.

```
1  int operacionAritmetica (const char *cadena, double *out) {
2      static int tt[6][6] = {//D + - * / 0
3                             {1,5,5,5,5,1}, //0-, inicial
4                             {1,2,2,2,3,1}, //1, ingreso de num1
5                             {4,5,5,5,5,4}, //2, se ingreso un operando + - *
6                             {4,5,5,5,5,5}, //3, se ingreso un operando /
7                             {4,5,5,5,5,4}, //4+, ingreso de num2 (final)
8                             {5,5,5,5,5,5}  //5, operacion no valida
9                            };
10     int estado = 0; //estado inicial = 0
11     int estadoFinal = 4; //solo un estado final
12     int c = cadena[0]; //caracter leido
13
14     double num1=0, num2=0, num3=0; //guarda valores numericos (operandos y el
       resultado)
15
16     char operador; //Sera uno de + - * /
17
18     unsigned int i=0;
19     while (c != '\0'){
20         estado = tt[estado][columna(c)]; // Cambio de estado
21
22         //Evaluamos donde guardar el caracter (num1, operando, o num2)
23         //en base al estado en el que estamos
24         if(estado==1) num1 = num1*10 + (c - '0');
25         else if(estado==2 || estado==3) operador = c;
26         else if(estado==4) num2 = num2*10 + (c - '0');
27
28         c = cadena[++i]; //Avanzo una posicion en la cadena
29     }
30     if(estado==estadoFinal){
31         //Si la operacion es valida la evaluamos segun el operando recibido
32         if(operador == '+') num3 = num1 + num2;
33         else if(operador == '-') num3 = num1 - num2;
34         else if(operador=='*') num3 = num1 * num2;
35         else if(operador == '/') num3 = num1 / num2;
36     //no hace falta preguntar por division por 0 porque el automata
37     //directamente no reconoce esa cadena
38
39         *out = num3; //Resultado final asignado al puntero
40         return 1;
41     }
42     else{
43         //operacion no valida
44         return 0;
45     }
46 }
```

El resultado de la operación evaluada según el operador almacenado (líneas 32 a 35) se almacena en la dirección apuntada por el puntero pasado como argumento (línea 39). [5]

Finalmente, el `main()` es muy sencillo y luce así:

```
1  int main(int argc, char *argv[]){
2      char s1[] = "";
3      strcpy(s1, argv[1]);
4      double out; //Almacenaremos en esta variable el resultado
```

[5] No era necesario usar la variable `num3` pero lo hicimos igualmente por cuestiones de organización personal.

```
5
6       if(!verifica(s1))
7       {
8           printf("Caracteres invalidos.\n");
9           return EXIT_FAILURE;
10      }
11      if(operacionAritmetica(s1, &out)){ //operacion valida
12          printf("Resultado de %s = %f", s1, out);
13          return EXIT_SUCCESS;
14      }
15
16      printf("Operacion no valida.");
17      return 0;
18  }
```

El programa se muestra funcionando correctamente en la figura 5, página 21.

4. Conclusiones

Hemos confeccionado nuestra propia versión de algunos de los lenguajes regulares que componen la totalidad del lenguaje ANSI C, así como realizado programas a modo de ejemplo para probar su correcto funcionamiento.

Además, vimos que las expresiones aritméticas más simples (constando únicamente de dos enteros y un operando en notación infija) también se pueden representar como un lenguaje regular.

Pudimos practicar el diseño de autómatas finitos, y el manejo del lenguaje C.

5. Anexo

5.1. Definición de funciones del archivo de cabecera `constantes.h`

```
#include <ctype.h>
#include "constantes.h"

//////////////////////////// COLUMNAS ////////////////////////////////

// Funcion que retorna el indice de la columna de la matriz
static int columnaOctal(int caracter);
static int columnaOctal(int caracter) {
    switch (caracter) {
        case '-': return 0;
        case '0': return 1;
        case 'u': return 3;
        case 'U': return 3;
        case 'l': return 4;
        case 'L': return 4;
        case '1': return 2;
        case '2': return 2;
        case '3': return 2;
        case '4': return 2;
        case '5': return 2;
        case '6': return 2;
        case '7': return 2;
    }
}

// Funcion que retorna el indice de la columna de la matriz
static int columnaDecimal(int caracter);
static int columnaDecimal(int caracter) {
    if(caracter=='-') return 0;
    else if(caracter=='0') return 1;
    else if(caracter=='u' || caracter=='U') return 3;
    else if(caracter=='l' || caracter=='L') return 4;
    else return 2; //Si no es nada de lo anterior, y tiene todos simbolos
    validos, debe ser uno de 1-9
}

// Funcion que retorna el indice de la columnaHexaSuf (columna hexa sufijos) en
    la matriz
static int columnaHexa (char caracter);
static int columnaHexa (char caracter) {
    char a0_F9[] = {'a', 'b', 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E', 'F',
    '1', '2', '3', '4', '5', '6', '7', '8', '9'};
    if (caracter == '-'){
        return 0;
    } else if (caracter == '0') {
        return 1;
    } else if (caracter == 'x' || caracter == 'X') {
        return 2;
    } else if (caracter == 'u' || caracter == 'U') {
        return 4;
    } else if (caracter == 'l' || caracter == 'L') {
        return 5;
    }else {
        for (int i = 0; a0_F9[i]; i++) {
```

```
52              if (a0_F9[i] == caracter) {
53                  return 3;
54              }
55          }
56      }
57  }
58
59  ////////////////////////// VERIFICADORES ////////////////////////////
60
61  // Funcion que verifica si los caracteres pertenecen al alfabeto
62  int verificaOctal (char *s);
63  int verificaOctal (char *s){
64      unsigned int i = 0;
65      for (i; s[i]; i++){
66          if(!( ( isdigit(s[i]) && (s[i] != '8' && s[i] != '9') ) || s[i] == 'u'
    || s[i] == 'U' || s[i] == 'l' || s[i] == 'L' || s[i] == '-')){
67              return 0;
68          }
69      }
70      return 1;
71  }
72
73  // Funcion que reconoce otros simbolos HEXA, se usa en verificaHexa
74  static int perteneceAlfabetoHexa (char c);
75  static int perteneceAlfabetoHexa (char c) {
76      char caracteres[] = {'a', 'b', 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E',
    'F', 'u', 'U', 'l', 'L', 'x', 'X'};
77
78      for (int i = 0; i < 18; i++) {
79          if (caracteres[i] == c) {
80              return 1;
81          }
82      }
83
84      return 0;
85  }
86
87  // Funcion que verifica si los caracteres pertenecen al alfabeto
88  int verificaHexa (char *s);
89  int verificaHexa (char *s) {
90      unsigned int i = 0;
91      for (i; s[i]; i++) {
92          if (!(isdigit(s[i]) || perteneceAlfabetoHexa(s[i]) || s[i]=='-' )){
93              return 0;
94          }
95      }
96      return 1;
97  }
98
99  // Funcion que verifica si los caracteres pertenecen al alfabeto
100 int verificaDecimal (char *s);
101 int verificaDecimal (char *s){
102     unsigned int i = 0;
103     for (i; s[i]; i++){
104         if(!(isdigit(s[i]) || s[i]=='-' || s[i]=='u' || s[i]=='U' || s[i]=='l'
    || s[i]=='L')){
105             return 0;
106         }
```

```
107     }
108     return 1;
109 }
110 
111 ////////////////////////// AUTOMATAS //////////////////////////////
112 
113 //Verifica si la cadena OCTAL pertenece al lenguaje
114 int automataOctal (const char *cadena);
115 int automataOctal (const char *cadena) {
116     static int tt [8][5] = {
117         // - | 0 | d | u | l
118           {6, 1, 7, 7, 7}, //0-
119           {7, 2, 2, 7, 7}, //1
120           {7, 2, 2, 3, 4}, //2+
121           {7, 7, 7, 7, 5}, //3+
122           {7, 7, 7, 5, 7}, //4+
123           {7, 7, 7, 7, 7}, //5+
124           {7, 1, 7, 7, 7}, //6
125           {7, 7, 7, 7, 7}  //7
126     };
127     int estado = 0; // Estado inicial
128     unsigned int i = 0;
129     int caracter = cadena[0];
130 
131     while ( caracter != '\0' ) {
132         estado = tt[estado][columnaOctal(caracter)]; // Cambio de estado
133         caracter = cadena[++i]; // Avanzo una posicion en la cadena
134     }
135 
136     if (estado == 5 || estado == 2 || estado == 3 || estado == 4) { //estados
    finales
137         return 1;
138     }
139 
140     return 0;
141 }
142 
143 //Verifica si la cadena HEXA pertenece al lenguaje
144 int automataHexa (const char *cadena);
145 int automataHexa (const char *cadena) {
146     static int tt [9][6] = {
147                             // - | 0 | x | d | u | l
148                                 {7, 1, 8, 8, 8, 8}, //0-
149                                 {8, 8, 2, 8, 8, 8}, //1
150                                 {8, 3, 8, 3, 8, 8}, //2
151                                 {8, 3, 8, 3, 4, 5}, //3+
152                                 {8, 8, 8, 8, 8, 6}, //4+
153                                 {8, 8, 8, 8, 6, 8}, //5+
154                                 {8, 8, 8, 8, 8, 8}, //6+
155                                 {8, 1, 8, 8, 8, 8}, //7
156                                 {8, 8, 8, 8, 8, 8}  //8
157                             };
158 
159     int estado = 0; // Estado Inicial
160     unsigned int i = 0;
161     int caracter = cadena[0];
162 
163     while (caracter != '\0') {
```

```
164         estado = tt[estado][columnaHexa(caracter)];
165         caracter = cadena[++i];
166     }
167 
168     if (estado == 4 || estado == 5 || estado == 6 || estado == 3) {
169         return 1;
170     }
171 
172     return 0;
173 }
174 
175 //Este automata para decimales SI acepta sufijos con "u,U" si la cadena comienza
        con "-"
176 //Verifica si la cadena DECIMAL pertenece al lenguaje
177 int automataDecimal (const char *cadena);
178 int automataDecimal (const char *cadena) {
179     static int tt [7][5] = {
180         {6,5,1,5,5}, //0-
181         {5,1,1,2,3}, //1+
182         {5,5,5,5,4}, //2+
183         {5,5,5,4,5}, //3+
184         {5,5,5,5,5}, //4+
185         {5,5,5,5,5}, //5
186         {5,5,1,5,5}  //6
187     };
188     int estado = 0; // Estado inicial
189     unsigned int i = 0;
190     int caracter = cadena[0];
191 
192     while ( caracter != '\0' ) {
193         estado = tt[estado][columnaDecimal(caracter)]; // Cambio de estado
194         caracter = cadena[++i]; // Avanzo una posicion en la cadena
195     }
196 
197     if (estado==1 || estado==2 || estado==3 || estado==4) { //estados finales
198         return 1;
199     }
200 
201     return 0;
202 }
```

5.2. Repositorio en GitHub

Utilizamos esta herramienta para llevar cuenta de los cambios realizados en cada uno de los programas.

`https://github.com/GuidoDipietro/tpAutomatas`

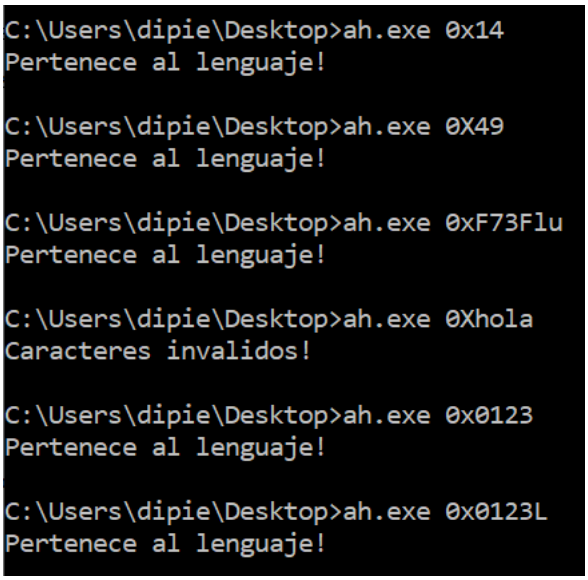


Figura 1: Autómata reconocedor de constantes hexadecimales en consola

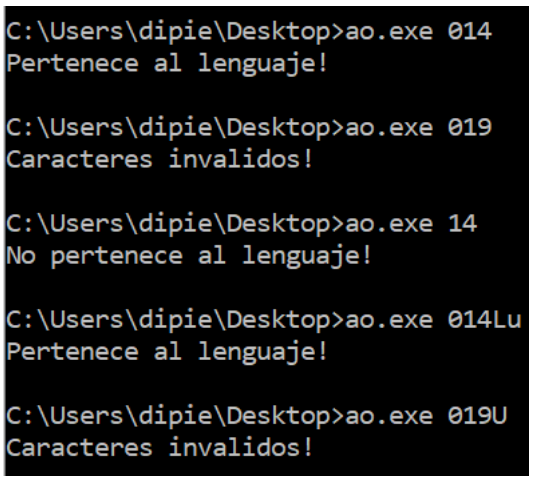


Figura 2: Ejecución en consola de autómata octal

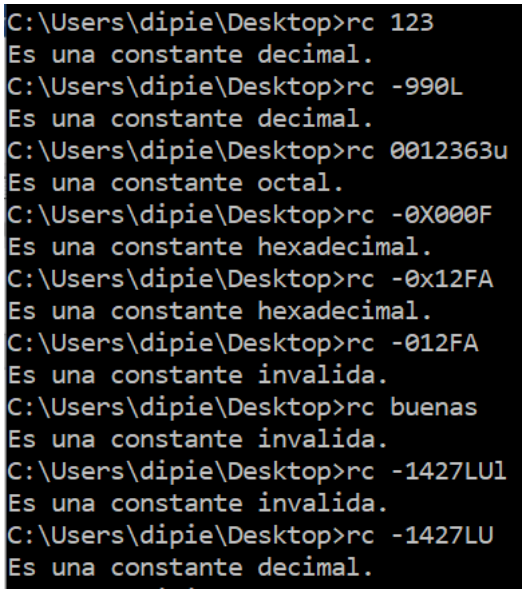


Figura 3: Reconocedor de constantes, ejecución en consola

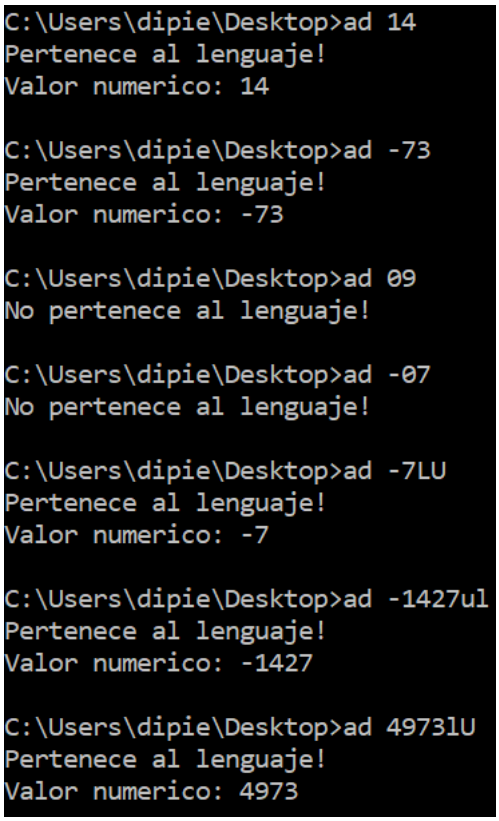


Figura 4: Autómata decimal que además retorna valor numérico de las constantes, en consola

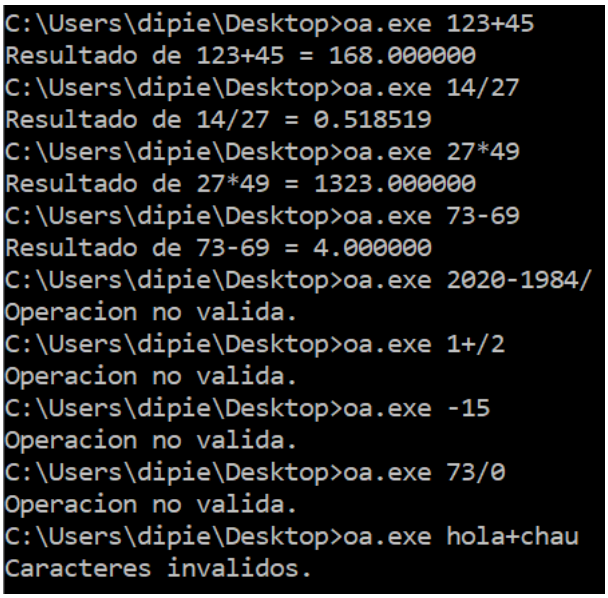


Figura 5: Autómata que reconoce y evalúa una operación aritmética simple