Autómatas Finitos

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Idea general

Autómata que reconoce comentarios C
- Pruebas
- Autómata a programa

Resumen

Dónde vamos

Tenemos una manera simple, pero poderosa, de denotar lenguajes: las expresiones regulares.
Nos interesa poder describir cómo *reconocer* un lenguaje, es decir, determinar si una palabra particular pertenece o no al lenguaje. Al efecto definiremos un tipo de máquina abstracta simple (un *autómata*) que responda a esta pregunta. Veremos luego que es simple traducir el autómata a un programa.

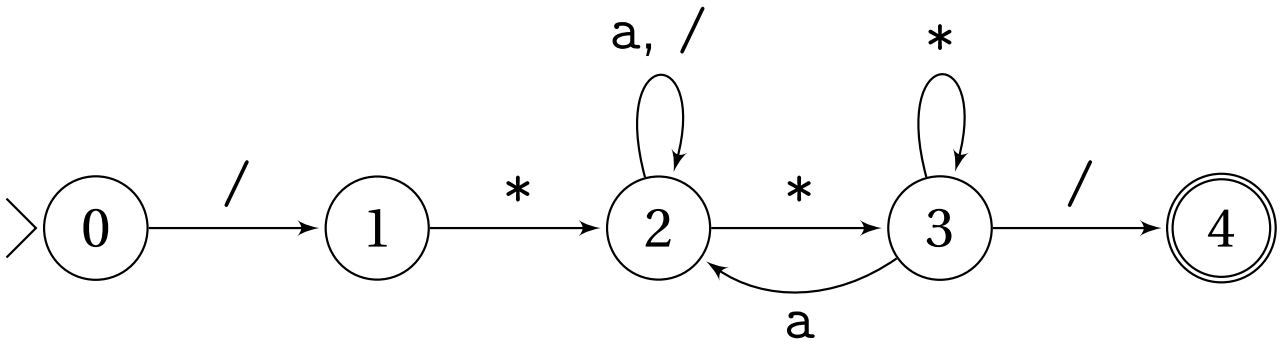
Ejemplo: comentarios en C

Consideremos los comentarios de C, como */* *este ejemplo* **/*.
// este estilo es de C++ (en C desde C99)
La idea es ir hilando *estados* y cómo moverse entre ellos según vienen llegando caracteres. Cada estado significa que hemos visto parte del comentario, nos dice qué falta ver para terminar.
Para simplificar, consideraremos un alfabeto solo con los símbolos /, * y a (este último para «todos los demás»).

Notación gráfica

Describiremos autómatas mediante diagramas, indicando mediante círculos los *estados* en que puede encontrarse y mediante flechas las *transiciones* entre estados. Marcamos con una punta de flecha el *estado inicial* en que comienza el procesamiento, indicamos con doble círculo *estados finales*, en los cuales consideramos debe aceptarse lo leído hasta allí.

Comentarios en C



Inicio
Un ‘/’, luego ‘*’
Repetir ‘a’, ‘/’
Rematar con ‘*/’
Repetir ‘*’
Atrás con ‘a’
También ‘/’

Tortura...

Algunos ejemplos de comentarios:

`/*aaaaa*/`	Común y corriente
`/**/`	No tiene chiste, pero es legal
`/*******/`	Armar cajitas...
`/*a*a*a*a*/`	No falta el decorativo `/*-*-*-*/`
`/*aaa/*aaa*/`	`/* x = 1; /* Descomente para... */`
`/*/////*/`	Raro, pero válido
`/*/aaaaa/*/`	Raro, pero válido
`/*/a****a/*/`	Raro, pero válido

Revise su texto de C y vea si explica correctamente *todas* las variantes de comentarios...

Modelar como autómata

Hay muchas situaciones que es útil modelar como autómatas finitos. Por ejemplo, la página Buttons as Finite Automata modela la interacción de un botón con el *mouse* en una interfaz gráfica.
Circuitos digitales son otro ejemplo cercano.
Muchas máquinas se mueven entre estados, como por ejemplo máquinas expendedoras aceptan monedas y solo dan el producto seleccionado si el importe es correcto.

Tabla de transición

Describir el autómata mediante un diagrama es poco formal. La *tabla de transición* dice cuáles son las transiciones.

	/	*	a	**Tipo**
0	1	5	5	Inicial
1	5	2	5	
2	2	3	2	
3	4	3	2	
4	5	5	5	Final
5	5	5	5	(Muerto)

Agregar un *estado muerto* completa la tabla.

Intérprete de la tabla de transición

Esto es tan simple como llenar la tabla, definir los símbolos permitidos (**enum** symbol en el programita), una función que traduzca de caracteres a símbolos (to_sym()), definir una constante para el estado inicial (**int** initial) y una manera de determinar si el estado es final (el *bitmap* final) y echar a correr.

Intérprete de la tabla de transición

```
enum symbol { SLASH, STAR, OTHER };

const int table[][3] = {
      /* 0 */ {1, 5, 5},
      /* 1 */ {5, 2, 5},
      /* 2 */ {2, 3, 2},
      /* 3 */ {4, 3, 2},
      /* 4 */ {5, 5, 5},
      /* 5 */ {5, 5, 5}};

const int initial = 0;
const unsigned int final = 0x1 << 4;
```

Intérprete de la tabla de transición

```
bool interpret(const char *str)
{
    int state = initial;

    for(const char *p = str; *p; p++) {
        state = table[state][to_sym(*p)];
    }
    return final & (0x1 << state);
}
```

Seguir las transiciones

Cada estado se traduce en un rótulo en el programa, si debemos ir a un estado vamos a ese rótulo.

```
q0:
    if(*str == '\0')
        return false;
    c = *str++;
    if(c == '/')
        goto q1;
    else
        goto qM;
```

Seguir las transiciones (cont)

```
q1:
    if(*str == '\0')
        return false;
    c = *str++;
    if(c == '*')
        goto q2;
    else
        goto qM;
```

Seguir las transiciones (cont)

```
q2:
    if(*str == '\0')
        return false;
    c = *str++;
    if(c == '*')
        goto q3;
    else
        goto q2;
```

Seguir las transiciones (cont)

```
q3:
    if(*str == '\0')
        return false;
    c = *str++;
    if(c == '/')
        goto q4;
    else if(c == 'a')
        goto q2;
    else
        goto q3;
```

Seguir las transiciones (cont)

Como el estado final no tiene salidas:

```
q4:
    if(*str == '\0')
        return true;
    str++;
    goto qM;
```

Para el estado muerto:

```
qM:
    if(*str == '\0')
        return false;
    str++;
    goto qM;
```

Usando un **switch**

Cada estado se traduce en un **case** del **switch**, similar al anterior.
Ir a un estado es cambiar la variable state.

```
int state = 0;

while(*str) {
    switch(state) {
    case 0:
        if(*str == '/')
            state = 1;
        else
            state = 5;
        break;
```

Usando un **switch** (cont)

```
case 1:
    if(*str == '*')
        state = 2;
    else
        state = 5;
    break;
case 2:
    if(*str == '*')
        state = 3;
    break;
```

Usando un **switch** (cont)

```
case 3:
    if(*str == '/')
        state = 4;
    else if(*str == 'a')
        state = 2;
    break;
```

Usando un **switch** (cont)

```
        case 4:
        case 5:
            state = 5;
            break;
        }
        str++;
    }
    return state == 4;
```

Otras opciones

Una opción particularmente eficiente se logró generando directamente lenguaje de máquina según el esquema de saltos.

Traducir a programa estructurado

```
bool comment(const char *str)
{
    if(!*str || *str++ != '/') return false;
    if(!*str || *str != '*') return false;

    do {
        str++;
        while(*str && *str != '*')
            str++;
        while(*str && *str == '*')
            str++;
    } while(*str && *str != '/');

    return *str == '/' && *(str + 1) == '\0';
}
```

Traducir a programa estructurado

Un programa debidamente estructurado es muy satisfactorio. Lo malo es que no es nada fácil verificar que el resultado es correcto.

Resumen

- Un *autómata* es un mecanismo simple que determina si acepta una palabra.
- Ilustramos la notación gráfica de autómatas finitos.
- Mostramos cómo pueden expresarse en una *tabla de transición*.
- Vimos cómo autómatas pueden traducirse a programas.