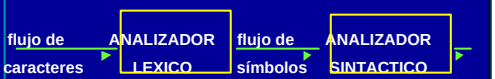
Análisis Léxico

1

FASE DE ANÁLISIS LÉXICO

- El código fuente es un flujo de caracteres.
- La tarea del analizador léxico es reconocer símbolos en este flujo de caracteres y presentarlos en una representación más útil para el análisis sintáctico



2

FUNCIONES DE UN ANALIZADOR LÉXICO

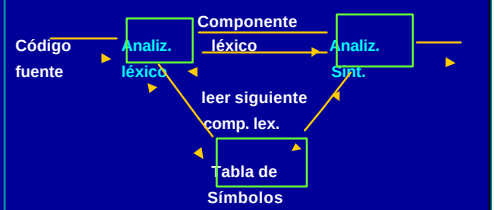
- **FUNCION PRINCIPAL**

  Leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos (tokens) que utiliza el Analizador Sintáctico para hacer el análisis.
  (Puede ser una subrutina del A.Sintáctico)

- **FUNCIONES CARACTERÍSTICAS**
  - Eliminar espacios, comentarios, caracteres de tabulación, saltos de línea, etc..
  - Reconocer identificadores y palabras claves
  - Introducir los identificadores en la Tabla de Símbolos
  - Reconocer constantes y numerales
  - Generar un listado para el compilador

3

INTERACCIÓN ENTRE EL A.LEXICO Y EL A.SINT.



- Recibida desde el AS , la orden "obtener el siguiente componente léxico (o símbolo)" , el AL lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico

4

PROCESO DE ANÁLISIS LÉXICO

- El reconocimiento del AL puede verse como una transformación en un flujo de símbolos *reducido* (entrada filtrada)
- También debe relacionar los mensajes de error del compilador con el programa fuente (Ej: asociar un número de línea a un mensaje de error) En algunos casos se hace una copia del fuente con los errores marcados
- Puede realizar preprocesamiento de macros
- Se pueden dividir en dos etapas : EXAMEN y AN. LEXICO

5

TÉCNICAS DE LOS ANALIZADORES LÉXICOS

- Estas técnicas se pueden aplicar también a lenguajes de consulta y sistemas de recuperación de información
- El problema es la especificación y diseño de programas que ejecuten *acciones activadas por patrones* dentro de las cadenas (*programación dirigida por patrones*)
- Un *lenguaje patrón-acción* para la especificación de AL : LEX

Los patrones se especifican con expresiones regulares y un compilador de LEX genera un reconocedor de la ER mediante un autómata finito eficiente.

- Otros leng. usan ER para describir patrones: shell (UNIX)

6

COMPONENTES LÉXICOS, PATRONES Y LEXEMAS

- En general hay un conjunto de cadenas en la entrada para el cual se produce como salida el mismo componente léxico (o token)
- Este conjunto de cadenas se describe mediante una regla llamada *patrón asociado* al componente léxico
- Se dice que el patrón concuerda con una cadena del conjunto
- Un *lexema* es un secuencia de caracteres en el programa fuente con la que concuerda el patrón para un componente léxico

7

COMPONENTES LÉXICOS, PATRONES Y LEXEMAS

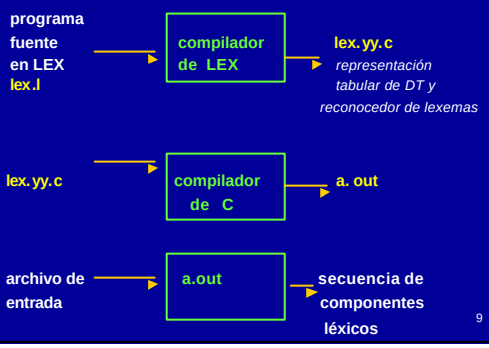
- Ejemplos:

CLex	if	id
Ej. Lexema	if	cuenta
Descrip.patrón	if	letra seguida de letras y dígitos
ER	if	*letra* *[A-Za-z]*
		dígito *[0- 9]*
		id *{ letra} ({letra} {dígito})**

8

Un lenguaje para especificación de AL : LEX

LEX: Lenguaje de patrón-acción



9

RAZONES PARA SEPARAR EL ANÁLISIS LÉXICO

- Se simplifican las fases
- Se mejora la eficiencia del compilador (Por ejemplo, con técnicas de manejo de buffers)
- Se mejora la transportabilidad. Las particularidades del alfabeto de entrada y de los dispositivos pueden limitarse al AL (Ej.: Símbolos especiales como- en Pascal) Iden para distintos códigos (EBCDIC, ASCII)
- Eliminación de espacios en blanco , tabuladores, saltos de línea y comentarios

10

EL AN. LÉXICO en el ejemplo de notación postfija

- Constantes:

Una constante entera es una secuencia de dígitos.

El AL debe agrupar los dígitos para formar enteros.

Cuando un secuencia de dígitos aparece en la entrada, el AL pasará num al AS

El valor entero se pasa como atributo del componente léxico num

11

EL A. L. en ejemplos

- Reconocimiento de identificadores y palabras claves:

Ej: cuenta := cuenta + increm;

se convierte en id = id + id

- El comp. léxico id se asocia a los lexemas cuenta e increm
- El traductor debe saber que lexema corresponde a cada id
- Cuando en la entrada aparece un lexema que forma un identificador, se necesita determinar si el lexema ya apareció antes.
- Se usa una Tabla de símbolos: el lexema se almacena y un puntero a esa entrada es el atributo de id.

12

EL A. L. en ejemplos (cont)

- También se debe distinguir cuando un lexema es una Palabra Clave y cuando un identificador.

  Si las palabras claves son reservadas, resulta fácil

  (una cadena resulta identificador si no es un palabra clave)

- Aislar Componentes Léxicos también es complejo cuando aparecen los mismos lexemas en más de un comp. léxico.

  Ejemplo: <, <=,<>

13

INTERFAZ CON EL EL ANALIZADOR LÉXICO

- El AL lee los caracteres de entrada, los agrupa en lexemas y pasa los comp. léxicos formados por los lexemas, junto con los valores de sus atributos, a las etapas posteriores de compilador.
- A veces el AL tiene que leer algunos caracteres por adelantado para poder decidir el comp. léxico
- El AL y el AS forman un productor-consumidor
- Los comp. léxicos se pueden almacenar en un buffer hasta ser consumidos. La interacción entre los dos está restringida al tamaño del buffer (por lo general es 1)

14

SE RETOMA EL EJEMPLO DEL TRADUCTOR DE INFIJO A POSTFIJO

PERO TAL QUE RECONOZCA NÚMEROS NATURALES (MÁS DE UN DÍGITO)

CONSTRUCCIÓN DE UN ANALIZADOR LÉXICO

15

UN ANALIZ. LÉXICO RUDIMENTARIO para el traductor de expresiones a postfijo

- Se permite que ingresen espacios en blanco y números dentro de las expresiones
- El analizador léxico realiza las siguientes interacciones:

16

UN ANALIZ. LÉXICO RUDIMENTARIO (cont)

- Las rutinas **getchar** y **ungetc** se encargan del manejo del buffer.
- c = getchar (); ungetc (c, stdin); no altera la cadena de entrada. (getchar y ungetc son rutinas del archivo incluido <stdio>
- La función **analex** devuelve un entero cuando observa una secuencia de dígitos , que es el **código de un comp. léxico** (>= 256 para contemplar los caracteres)
- A la variable global **valcomplex** se le asigna **la secuencia de dígitos** que forma el valor del comp. léxico
- En la gramática original que trabajaba con dígitos, deben realizarse modificaciones para admitir números

17

MODIFICACIÓN DE LA GRAM. DEL TRADUCTOR

- La especificación inicial del traductor de infijo a postfijo era:

  expr –> expr + term {print (“+”) }
  expr –> expr - term {print (“-”) }
  expr –> term
  term –> 0 {print (“0”) } | .....| 9 {print (“9”) }

- Debe cambiar por

  expr –> expr + factor {print (“+”) }
  expr –> expr - factor {print (“-”) }
  expr –> factor
  factor –> (expr) | num { print (num.valor) }

18

MODIFICACIÓN DE LA GRAM. DEL TRADUCTOR

- Debe cambiar por

  expr –> expr + factor {print (“+”) }
  expr –> expr - factor {print (“-”) }
  expr –> factor
  factor –> (expr) | num { print (num.valor) }

- **num** es el componente léxico que representa a un entero
- Cuando una secuencia de dígitos aparece en la cadena de entrada, el AL pasará num al AS
- El valor del entero se pasará como atributo del comp. Lex. Num
- Ej: La entrada : **31 + 28 + 29** , se transforma en

  <num.31> <+.> <num.28> <+.> <num.29>

- El comp.lex. + no tiene atributo asociado.
- Los segundos componentes de las tuplas son atributos y no desempeñan papel alguno durante el AS, pero son necesarios para la traducción

19

EL CODIGO C anterior

```
# include <ctype.h>     /* carga el archivo que tiene isdigit*/
int sp;
main ( )
      { sp = getchar ( ) ; expr ( ) , putchar (‘ \n´) ; / * salto  línea*/ }
expr ( )
     {termino ()
      while ( 1)
         if sp ==‘+’ {.....}else if sp ==‘-’ {.......} else break }
término ( )
      { if ( isdigit (sp) ) { putchar(sp)  ; aparea (sp);}
          else error ( ) ;}
aparea (t)
      int t;
       { if ( sp == t ) sp = getchar  ( ) ; else  error  ( ) ;}
error  ( )
       { printf ( “ error de sintaxis \ n”) ; exit (1) ;}
```

20

EL CODIGO C PARA factor

```
factor ( )
{
 if ( sp== ´( ´ ) { parea( ´ ( ´ ), expr ( ) , parea( ´) ´ );}
else if ( sp == NUM)
    { printf (“ % d “, valcomplex), parea(NUM);}
else error( );
}
```

- El valor del atributo *num.valor* está dado por la variable global *valcomplex*
- “ %d “ indica que se imprimirá la representación decimal del argumento *valcomplex*, con blanco antes y después.

21

EL CODIGO C PARA factor (elimina blancos y reconoce números)

```
# include <stdio.h>   # include<ctype.h>
int numlinea = 1;  int valcomplex = NINGUNO;
int analex ()
  { int t; while (1)
  { t = getchar ();
    if ( t == ´´ | t == ´ \ t´);  /* elimina blancos y tab */
       else if ( t == ´ \ n ´ ) numlinea = numlinea + 1;
            else if ( isdigit ( t) )
                 { valcomplex = t - ‘0’;   t = getchar ();
                   while ( isdigit (t))
                     { valcomplex = valcomplex * 10 + t - ‘0’;
                       t = getchar ();}
                   ungetc (t , stdin); return NUM;}
            else { valcomplex = NINGUNO; return t;} } }
```

22

EL A. LÉXICO Y LA TABLA DE SÍMBOLOS

- Durante el A Léxico los lexemas se guardan en la tabla de Símbolos
- Cuando se guarda un lexema, también se guarda el componente léxico asociado
- Sobre la Tabla de Símbolos se realizan las siguientes operaciones:

  inserta (s,t) : devuelve el índice para la nueva entrada de la cadena **s** y el componente léxico **t**

  busca (s) : devuelve el índice de la entrada para la cadena**s**, o 0 si no encontró a **s**

23

HERRAMIENTAS PARA LA CONSTRUCCIÓN DE ANALIZADOR LÉXICOS

24

COMPONENTES LÉXICOS, PATRONES Y LEXEMAS

- Hay un conjunto de cadenas en la entrada para el cual se produce como salida el mismo componente léxico (o TOKEN).
- Este conjunto de cadenas se describe mediante una regla llamada PATRÓN asociado al comp. Léxico

Se dice que el patrón CONCUERDA con cada cadena del conjunto

- Un LEXEMA es una secuencia de caracteres en el programa fuente con la que concuerda el patrón para un comp. léxico

25

COMPONENTES LÉXICOS, PATRONES Y LEXEMAS

- Un patrón es una regla que describe el conjunto de lexemas que pueden representar a un determinado comp. léxico en el programa fuente
- Los comp. léxicos se tratan como símbolos terminales de la gramática del lenguaje fuente, con nombres en cursiva
- Se consideran componentes léxicos:

palabras claves,
operadores,
identificadores,
constantes,
cadenas literales,
signos de puntuación

26

Ejemplo de COMP. LÉX., PATRONES Y LEXEMAS

Componente Léxico	Lexema de ejemplo	Descripción informal del patrón
const	const	const
if	if	if
relación	<,<=,>,>=, = , <>	< o <= o> o >= o = o <>
id	pi, cuenta, D2	letra seguida de letras y dígitos
num	3.1416, 0, 6.2E23	cualquier constante numérica
literal	" cadena vacía"	cualquier carácter entre " y ", excepto "

27

Reconocimiento de COMP. LÉX.

- En algunos casos existe una dificultad potencial de reconocer los componentes léxicos:

✓ En Fortran los espacios no son significativos.
En : DO 5 I = 1. 25 .... DO 5 I = 1, 25
no se puede saber hasta ver el punto (o la coma) si DO es una palabra clave o es parte del identificador DO5I

✓ En PL/I, las palabras claves no son reservadas, lo que complica distinguir claves de identificadores:
IF THEN THEN THEN = ELSE; ELSE ELSE = THEN;

28

ATRIBUTOS DE LOS COMP. LÉXICOS

- Cuando concuerda con un lexema más de un patrón, el AL debe proporcionar informac. adicional sobre el lexema concreto que concordó
- El AL recoge informac. sobre los comp. léx. en sus atributos asociados
- Los comp. lex. influyen en las decisiones del AS y los atributos en la traducción de los comp.lex.
- Los comp. lex. suelen tener un solo atributo (un puntero a la entrada de la tabla de símbolos donde se guarda la información sobre el comp.lex. (por ej. el lexema de un identificador y el nro. de línea donde se encontró por primera vez.)
- En Fortran : E = M * 2 , los comp. lex y los atributos asociados son: <id.puntero a la TS para E>, <op-asig.>, < id. puntero a la TS para M> , <op-mul > , <num.valor * 2>

29

ERRORES LÉXICOS

- Son pocos los errores que puede detectar el AL por la visión restringida del programa fuente
- En C: fi (a == f(x)) .... No puede distinguir si está mal escrito if, o si fi es identificador de función no declarado

El AL debe devolver el comp. léxico de un identificador y dejar que otra fase se ocupe de los errores.

- Cuando ninguno de los patrones concuerda con el prefijo de la entrada restante, el AL puede seguir una estrategia de recuperación

(Ejemplos: borrar un carácter extraño,
insertar un carácter que falta,
reemplazar uno incorrecto por otro correcto,
intercambiar dos caracteres adyacentes).

30

ERRORES LÉXICOS

- Observar un prefijo de la entrada restante para ver si se puede transformar en un lexema válido en una sola transformación de error.

Se supone que en general un lexema es válido si requiere una sola transformación.

- Un criterio (teórico) de correcciones es el de distancia mínima de un programa

Calcular el número mínimo de transformaciones necesarias para transformar el programa erróneo en otro que esté sintácticamente bien construido.

El prog. erróneo tiene k errores --> secuencia más corta de transformaciones de error de longitud k

Aplicación demasiado costosa

31

DELIMITADORES Y PALABRAS RESERVADAS

Según las palabras reservadas y el comportamiento del blanco , los lenguajes de programación se pueden clasificar en:

- Los blancos son delimitadores y hay palabras reservadas (Pascal, Cobol)
- Los blancos son delimitadores pero no hay palabras reservadas (PL/I) .

Se puede simplificar si es obligatoria la declaración de variables.

- Los blancos se ignoran y no hay palabras reservadas (Fortran ANS). Es el tipo más difícil de compilar
- Los blancos se ignoran pero hay palabras reservadas (Basic)

32

MANEJO DE BUFFERS DE ENTRADA

- El AL es la fase que lee char por char el PF --> consume mucho tiempo, aunque las otras fases sean más complejas
- La velocidad de un AL supone un problema en el diseño de compiladores
- Hay 3 métodos generales para la implantación de un AL
  - Utiizar un generador de AL, como el LEX para producir un AL a partir de una especificación basada en expr. regulares
    (Proporciona rutinas para leer la entrada y manejar buffers)
  - Escribir el AL en un leng. de prog. de sistemas, utilizando las posibilidades de e/s de este leng.
  - Escribir el AL en lenguaje ensamblador y manejar explícitamente la lectura de la entrada

33

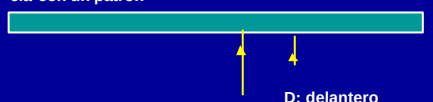
PAREJAS DE BUFFERS

- A veces se necesita preanalizar varios caracteres, además del lexema para un patrón , antes de poder anunciar la concordancia
  (reinsertando caracteres)
- Para reducir el movimiento de caracteres se pueden utilizar buffers.
- EJEMPLO: Un buffer dividido en dos mitades de tamaño N. Se leen N caracteres en cada lectura
  (o bien eof)

34

PAREJAS DE BUFFERS (cont)

- Se usan dos punteros (inicio y fin del lexema en curso)
  P apunta al 1er. caracter del próximo lexema a encontrar
  D examina hacia delante hasta encontrar una concordancia con un patrón



- Después de procesar el lexema ambos apuntan al carácter siguiente.

35

PAREJAS DE BUFFERS (cont.)

- Cuando una mitad del buffer se llena se carga la otra
- Pero este esquema limita la cantidad de caracteres de preanálisis
- En el ejemplo de PL/I:
  DECLARE (ARG1, ARG2, ... , ARGn)
  no se puede determinar si DECLARE es una pal .clave o un nombre de arreglo hasta ver el car que sigue al paréntesis derecho. En cualquier caso el lexema termina en la 2da E
- Código para avanzar el puntero en un buffer de tamaño 2N :

```
if D= N then begin recargar ( N+1, 2N); D=D+1 end;
else if D=2N then begin recargar (1, N); D=1 end;
     else D = D + 1;
```

36

ESPECIFICACIÓN DE LOS COMP. LÉXICOS

- En general, los componentes léxicos que se deben reconocer forman un lenguaje regular (Gramática Tipo 3)
  - El proceso de reconocimiento lee una frase de izq. a der. y la acepta si puede reducirla al símbolo inicial de la gramática
  - Por cada paso o estado hay un NT al principio de la frase parcialmente reducida.
  - Cada reducción depende del NT actual y del siguiente caracter de entrada
- Lenguajes regulares <----> EXPRESIONES REGULARES
- Las ER son una notación importante para especificar patrones.
  Ej: la ER para identificadores en Pascal: letra (letra | dígito) *

37

EXPRESIONES REGULARES

- Definición de Expresiones Regulares del alfabeto S:
  1. l es una ER designada por {l} (conj. de la cad. vacía)
  2. Si a Î S, a es una ER designada por {a}
  3. Si r y s son ER representadas por los leng. L(r) y L (s):
     a) r | s es una ER representada por L(r) U L(s)
     b) r s es una ER representada por L(r) L(s)
     c) (r)* es una ER representada por (L(r))*
     d) (r) es una ER representada por L(r)
- Un lenguaje designado por una ER es un Conjunto Regular

38

EXPRESIONES REGULARES (cont.)

- Se pueden adoptar convenciones para evitar paréntesis:
  1- El Operador * tiene mayor precedencia y es asoc. a izq.
  2- La concat. tiene la 2da mayor precedencia y es asocizq
  3- | tienen menor precedencia y es asoc. Izq.

  Así: (a) | ((b)* (c)) es equivalente a a | b* c
  Conjunto de cadenas que tienen una sola a ,
  o 0 o más b seguidas de una c
- Ejercicios:
  1- a|b 2- (a|b) (b|a) 3- (a|b)* 4- a|a*b
- r es equivalente a s <----> L(r) = L(s)

39

LEYES ALGEBRAICAS DE LAS EXP. REG.

AXIOMA	DESCRIPCIÓN
r |s = s|r	| es conmutativo
r | (s| t) = (r | s) | t	| es asociativo
(r s) t = r(s t)	la concat. es asociativa
r (s| t) = r s | r t	
(s| t) r = s r | t r	la conc. distribuye sobre |
l r = r	
r l = r	l es neutro de la concatenación
r* = (r| l)*	la relación entre * y l
r** = r*	* es idempotente

40

DEFINICIONES REGULARES

- Se pueden dar nombres a las ER y utilizarlos como símbolos para definir otras ER:

  d1 --> r1
  d2 --> r2
  ..... dn --> rn

donde cada ri es una ER sobre los símbolos S U {d1,d2...,d i-}, luego para cada ri hay una ER en S.

- Ejemplo de números en Pascal:

  digito --> 0 | 1| ...|9
  *digitos --> digito digito **
  fraccion-opt --> . *digitos* | l
  expon-opt --> (E (+ | - | l) *digitos*) | l
  num --> digitos fraccion-opt expon-opt

41

ABREVIATURAS DE NOTACIÓN

- UNO O MÁS CASOS:

Si r es ER que designa a L(r) ---> (r)⁺ es una ER de (L(r))⁺
* y ⁺ tienen la misma precedencia y asociatividad
r* = r⁺|l y r⁺= r r*

- CERO O UN CASO:

Si r es ER -----> (r) ? es la ER de L (r) U {l }
r ? = r | l
En el ejemplo de num. en Pascal:
fraccion-opt --> (. Digitos)?
expon-opt--> (E (+| -)? digitos)?

- CLASES DE CARACTERES:

[a b c] designa a la ER : a| b | c; [a-z] designa a: a| b|.....|z
Ejemplo de id: [A-Z a-z] [A-Za-z0-9]*

42

CONJUNTOS NO REGULARES

- Considerando la clasificación de gramáticas:

hay lenguajes que no se pueden describir con ER.

- Ejemplos:

1- El conj. de las cadenas de paréntesis equilibrados (se puede describir con una GLC)

2- Las cadenas {wcw / w es cadena con a y b} no se puede describir con ER ni con GLC

3- Cadenas del tipo (nHa1a2.....an) no se pueden describir

Las ERr solo se pueden utilizar para designar un nro finito de repeticiones o un nro no especificado de repeticiones de una construcción

43

RECONOCIMIENTO DE COMP. LÉXICOS

- Para estudiar el proceso de reconocimiento de comp. léxicos se tomará como ejemplo la siguiente gramática:

prop --> if expr then prop
| if expr then prop else prop | l
expr --> termino oprel termino | termino
termino --> id | num

donde los terminales generan conjuntos de cadenas dados por las siguientes definiciones regulares:

if --> if else --> else oprel --> < | <=| = | > | >= | < >
then --> then id --> letra (letra | digito) *
num --> digito⁺ (. digito⁺)? (E (+|-)? digito⁺)?

y con separaciones de blancos, carac. TAB y saltos de líneas:

delim --> blanco | tab | lineanueva eb --> delim ⁺

44

RECONOCIMIENTO DE COMP. LÉXICOS (cont)

- El AL reconocerá las palabras claves **if**, **then** , **else** , que se supone son reservadas, y los lexemas **oprel**, **id** y **num** (que representa números enteros y reales sin signo)
- Los espacio en blanco son eliminados por el AL:

  Cuando hay concordancia con **eb** , el AL no devuelve comp. lex. al AS, sino que busca un comp. lex. a continuación del espacio en blanco
- El AL debe aislar un lexema del buffer de entrada y producir como salida un par (comp. lex. , valor de atributo)
- Modelo matemático apropiado:

  autómatas y diagramas de transiciones

45

PATRONES DE ER PARA COMP. LEX.

ER	COMP. LEX.	VALOR DEL ATRIBUTO
eb	-	-
if	if	-
then	then	-
else	else	-
id	id	puntero a la entrada en la tabla
num	num	puntero a la entrada en la tabla
<	oprel	MEN (constante simbólica)
< =	oprel	MEI
=	oprel	IGU
< >	oprel	DIF
>	oprel	MAY
> =	oprel	MAI

46

AUTÓMATAS

- **AUTÓMATAS:** Modelos matemáticos de los dispositivos que aceptan una entrada y producen una salida apropiada
- La característica de un autómata es que la entrada pasa por varios estados para producir la salida , por lo que se puede establecer una correspondencia entre el problema de reconocimiento (AL) y un autómata.
- Los autómatas (o MAQUINAS RECONOCEDORAS) son dispositivos formales que permiten definir un lenguaje por medio de la vía de aceptación de cadenas o tiras que lo componen:

Gramáticas	Autómatas
generan las cadenas a partir de las reglas de prod.	deciden si una cadena pertenece o no a un leng. dado

47

AUTÓMATAS

- En general, una máquina reconocedora consta de:
  - ***Una cinta de entrada***: contiene una tira de símbolos del alfabeto, cada uno en una celda y se lee de izquierda a derecha.
  - ***Una memoria***: contiene una función de bùsqueda y de direccionamiento
  - ***Un control de estados***: que determina el comportamiento del mecanismo y debe ser finito
- ***Configuración***: Conj. de elementos que definen la situación del autómata en un instante determinado.
- Un autómata ***acepta o reconoce*** una tira o cadena de la cinta de entrada,si empezando en la config. Inicial y después de una serie de movimientos se llega a una config. final.

48

TIPOS DE MÁQUINAS RECONOCEDORAS

- El lenguaje definido por una maq. Reconocedora es el conj. de todas las cadenas aceptadas por dicho dispositivo
- Los tipos están en relación con el tipo de lenguaje que aceptan.

AUTOMATAS FINITOS: reconocen lenguajes regulares
(Movimiento unidireccional- sin memoria auxiliar- usados para AL)

AUTOM. CON PILA: reconocen leng. libres de contexto
(Movimiento unidireccional - memoria en estructura de pila - usados en AS teóricos)

AUTOM. BIDIRECCIONALES: reconocen lenguajes dependientes del contexto (Cinta con doble movimiento)

MAQUINAS DE TURING: reconocen lenguajes sin restricciones (Son también bidireccionales)

49

AUTÓMATA FINITO DETERMINISTA (cont)

- Un autómata finito se dice determinístico si, para cada estado y caracter de entrada , tiene un solo estado de transición al cual puede cambiar
  (Función de transición, sin entrada l)
- ALGORITMO DE LAS HEURÍSTICA:

```
init ( estado, leer-pos, eot)
while not eot do
        read ( c) ; sigte-estado (c, estado)
end;
if estado in estado-final
  then aceptar-cadena
 else rechazar-cadena
end;
```

50

DEF. DE AUTÓMATA FINITO DETERMINISTA

- Un autómata finito determinístico (o aceptor) puede definirse formalmente como una quintupla
  A = (P, S, M, ei, F) :

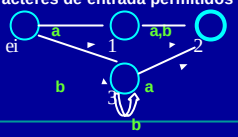
P: conjunto de estados
S: alfabeto
M: función de transición de estados (M: PxS ---> P)
ei: estado inicial
F: conjunto de estados finales

- El lenguaje que acepta un autómata A es:
  L(A) = { x/ x∈ S* y ei --*--> f, f ∈ F}
  (Hay una sec. de transiciones que llevan a un estado final)
- A = A' <--> L(A) = L(A')

51

AUTÓMATAS Y DIAGRAMAS DE TRANSICIÓN

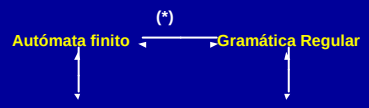
- Un FDA se puede representar por una Tabla de Transiciones (filas estados, columnas entradas, celdas nuevo estado)
- También mediante un Diagrama de Transición:
  Los estados son círculos marcados con el nombre
  Los estados finales se identifican oscuros
  Los estados se conectan por arcos marcados con caracteres de entrada permitidos



52

GRAMÁTICAS - EXP. REG. - AUTÓMATAS

- Un teorema demuestra la correspondencia biunívoca :

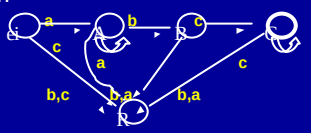


- (*) se obtiene empleando símbolos no terminales para representar estados y terminales para producir las transiciones entre estados.

Se suelen agregar el estado inicial y un estado de fallo o rechazo al cual se puede llegar si la cadena no es válida

Ejemplo:GRAMÁTICAS- EXP. REG. - AUTÓMATAS

- L = { a^m b c^n | m,n > 0}
- G : S={a,b,c} N= {A;B;C} Símbolo inicial: C
  P: C --> Cc | Bc B -->Ab A -->Aa| a
- DT:



Volviendo al ejemplo de expresiones

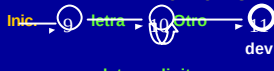
LOS DIAG. DE TRANSICIÓN EN EL EJEMPLO

- DIAG. DE TRANSICIÓN PARA LOS PATRONES >= Y >

- DIAG.DE TRANS. PARA OPERADORES RELACIONALES

LOS DIAG. DE TRAN. EN EL EJEMPLO (cont)

- DT PARA IDENTIFICADORES Y PALABRAS CLAVES



devuelve (obt-complex ( ),
instala-id ( ))

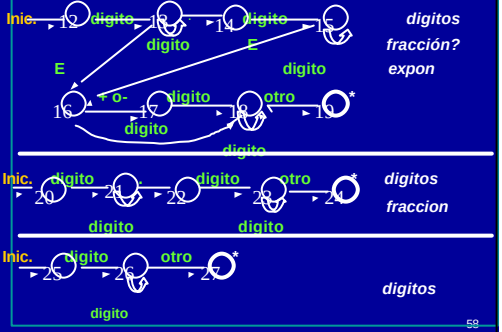
letra o digito

- Cuando se llega al estado de aceptación se determina si es palabra clave o identificador, por ejemplo colocando las palabras claves desde el principio en la tabla se símbolos
- obt-complex : Obtiene el componente léxico
- instala-id : Obtiene el atributo.

Identifica el lexema en el buffer. Examina la Tabla de Símbolos (TS).
Si el lexema está marcado como palabra clave, devuelve 0.
Si lo encuentra y es una variable, devuelve el puntero a TS.
Si no lo encuentra crea una entrada en TS y devuelve el puntero.

LOS DIAG. DE TRAN. EN EL EJEMPLO (cont)

- DT PARA NUMERO SIN SIGNO EN PASCAL



LOS DIAG. DE TRAN. EN EL EJEMPLO (cont)

- La acción cuando se llega a los estados de aceptación 19,24 y 27 es llamar al procedimiento instala-num ( ) que introduce el lexema en una tabla de números y devuelve un puntero a esa entrada. El AL devuelve el comp.lex. num con ese puntero como valor
- Se puede emplear inform. adicional para señalar errores.
  Ej: 1 . < x , se fracasa en 14 y 22 con < como car. sgte.
  Se podría informar un error y continuar como si la entrada fuera 1.0 < x
- En estos diagramas hay concordancia redundante de DT
  Se pueden combinar en uno (laborioso) o modificar la respuesta al fallo (método que avanza después de una aceptación y retrocede al último estado de aceptación cuando hay un fallo)

LOS DIAG. DE TRAN. EN EL EJEMPLO (cont)

- DIAG. DE TRAN. PARA RECONOCER eb

- Cuando se alcanza el estado de aceptación no se devuelve nada al ASintáctico . Se regresa al estado inicial del DT para buscar otro patrón.
- Orden : Se llega a un DT sólo después de haber fallo en todos los diagramas anteriores.
  Conviene buscar comp.lex. que aparezcan con más frecuencia antes que los menos frecuentes.
  En este caso conviene poner el DT de eb al principio

IMPLANTACIÓN DE UN DIAG. DE TRANSICIONES

- Una sec. de DT se puede convertir en un programa que busque los comp.lex. especificados por los diagramas.
- Para cada estado se genera un segmento de código.

Si el estado tiene sucesores debe leer un carácter y seleccionar la arista a seguir (si es posible)

sigtecar ( ) lee el siguiente caracater del buffer, avanza el puntero delantero y devuelve el caracter leído.

fallo ( ) : Se llama cuando no hay estado siguiente y el estado actual no indica que se ha encontrado comp lex . Retrocede el puntero a la posición inicial de la búsqueda e inicia la búsqueda del comp lex especificado por el sgte DT

Si no hay que probar más DT , llama a Rutina-Recup-Err.

61

Seudocodigo para la IMPLANTACIÓN DEL D. T.

```
estado = 0, inicio =0;
valor-lexico   /* contiene los punteros */
fallo ( )
{ delant = inicio-lexema
  switch (inicio) {
        case 0: inicio = 9; break;
        case 9: inicio = 12; break;
        case 12: inicio = 20; break;
        case 20: inicio =25; break;
        case 25: recupera ( ) ; break;
        default: /* error del compilador */
        }
}
end fallo
```

62

Seudocodigo para la IMPLANTACIÓN DEL DT(cont)

```
Complex sigte-complex ()
{ while (1) {
    switch ( estado) {
    case 0:  c = sigtecar ();
            if (c == blanco | c == tab | c== nueva- linea) {
              estado = 0 ; avanza-inicio-lexema++; }
            else if (c ==‘<’)  estado = 1;
            else if ( c == ‘=’) estado = 5; .....va >.....
            else estado= fallo (); break;  ....casos de 1 a 8.....
   case 9: c =sigtecar () ; if ( esletr ( c)  then estado=10
                          else estado = fallo (); break; ...caso 10.....
   case 11: regresa (1); instala-id; return (obt-complex ( ) );  ..de 12 a 26.....
   case 27: regresa ( 1); instala-num; return (NUM);
 } } }
end complex
```

63

Especificaciones en Lex

Un programa Lex consta de tres partes:

- Cada acción es un fragmento de programa que describe cual ha de ser la acción del AL cuando el patrón pi concuerda con un lexema
- Los procedimientos auxiliares que pueden necesitar las acciones (incluso se pueden compilar por separado)

64

Ejemplo en Lex

```
/* definiciones regulares */
delim       [ \t  \n ]
eb          {edelim} +
letra       [A-Za-z ]
digito      [0-9 ]
id          {letra} ({letra} {digito}) *
numero .....................................
%%
{eb}      {/* no se devuelve nada*/}
if        {return (IF);}
Then      {return (THEN);}
{numero }  ÿyylval  = instala-num; return ( NUMERO); }.....................................
%%
Istala-id ( )     { ......procedimientopara instalar un lexema....}
instala-num   { ......procedimientopara instalar un lexema....}
```