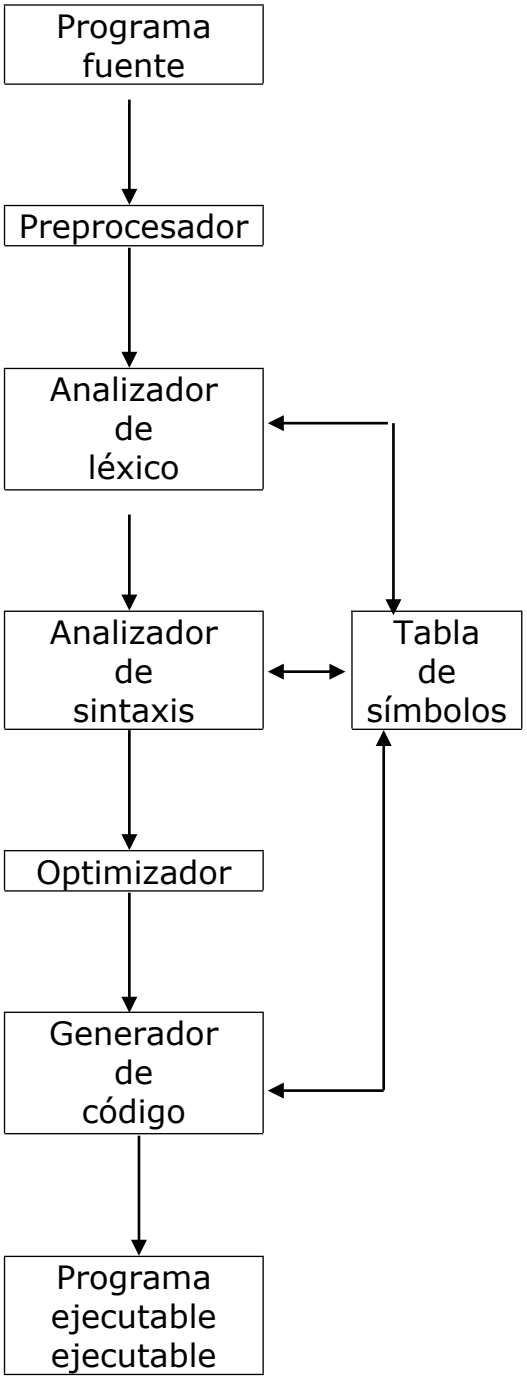
En nuestro curso, Teoría de lenguajes, presentamos los aspectos más importantes en la construcción de un compilador. Estos aspectos incluyen una serie de técnicas, las cuales son aplicables, no sólo en la construcción de un compilador, sino también en el análisis y desarrollo de otras aplicaciones propias del ejercicio profesional de un ingeniero de sistemas.

Presentamos inicialmente el esquema general de un proceso de compilación.



- El primer paso, el preprocesamiento, consiste básicamente en eliminar todos los comentarios presentes en un programa fuente, expandir macros e incluir programas de librerías creadas por el usuario que sean invocadas por el programa a analizar.

- El segundo paso, el analizador de léxico, conocido también como scanner o tokenizador, reconoce y convierte la entrada a una forma más trabajable para el resto del proceso de compilación. El analizador de léxico toma la hilera de entrada como una colección de elementos básicos del lenguaje, llamados tokens. Un token es una unidad indivisible del léxico, la cual está compuesta por dos datos: clase y valor.

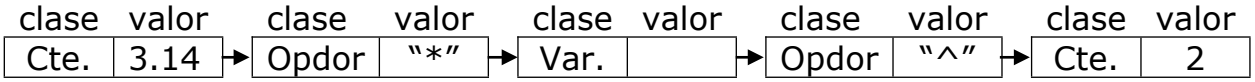
  El analizador de léxico, por lo general se efectúa utilizando autómatas finitos.

  En un lenguaje de programación, palabras reservadas como for, if, while son tokens; símbolos como >=, ||, &&, ==, son tokens; nombres de variables y constantes son tokens, y así sucesivamente.

  La hilera de símbolos que conforma un token se llama lexema. No existe una correspondencia única entre un token y un lexema. Una constante o una variable son tokens que pueden tener infinitos lexemas. Hay otros tokens como while, for, if que sólo tienen un lexema asociado a él.

  El analizador de léxico identifica lexemas y los convierte en tokens.

  Si tenemos la expresión 3.14*radio^2, al efectuar el proceso de análisis lexicográfico o tokenización, obtenemos:



En el campo de valor asociado a la clase variable tendremos un apuntador hacia una entrada en la tabla de símbolos donde se halla especificada totalmente la variable "radio".

- El tercer paso, el analizador de sintaxis, conocido también como "parser", controla que la secuencia de tokens creada por el analizador de léxico corresponda al lenguaje definido por una gramática. Además, produce un código intermedio, a partir del cual se generará el código ejecutable. Por lo general, los parsers se efectúan con autómatas de pila de un estado.

- El siguiente paso, el proceso de optimización, por lo general se efectúa sobre el código intermedio. Existe una variedad de técnicas para mejorar la eficiencia, en cuanto a tiempo de ejecución, de un algoritmo. Algunos

compiladores usan técnicas muy sofisticadas, produciendo códigos ejecutables bastante más veloces.

- Por último, se ejecuta la generación del programa ejecutable, la cual depende del procesador en el cual se vaya a ejecutar el programa.

El presente texto trata lo correspondiente al análisis lexicográfico y al análisis sintáctico. En los módulos 1 a 3 se presentan los temas correspondientes a las herramientas para el análisis lexicográfico. En el módulo 4 se presenta la herramienta básica para el reconocimiento del lenguaje definido por una gramática. En módulos 5 y 6 se presentan los temas correspondientes a la construcción de gramáticas, las cuales definen formalmente un lenguaje. En los módulos 7 a 11 se presenta la teoría correspondiente a efectuar proceso de reconocimiento descendente, y en los módulos 12 a 16 se presenta la teoría para el reconocimiento ascendente.

MODULO 1
CONCEPTOS BÁSICOS

1.1 INTRODUCCIÓN
El tema de autómatas finitos es un tema imprescindible y necesario en el análisis lexicográfico. En este módulo trataremos los autómatas deterministas, veremos cada uno de los elementos y restricciones de ellos y su construcción y manipulación.

1.2 OBJETIVOS
1. Conocer este modelo matemático, los elementos que lo componen, su funcionamiento y su representación.
2. Construir autómatas finitos.
3. Determinar estados de un autómata finito.
4. Identificar estados de aceptación de un autómata finito.
5. Determinar estados equivalentes de un autómata finito.
6. Determinar estados extraños de un autómata finito.
7. Simplificar un autómata finito.

1.3 PREGUNTAS BÁSICAS
1. Qué es un autómata finito.
2. En cuáles situaciones se debe usar un autómata finito.
3. Cómo se representa un autómata finito.
4. Cómo se definen los estados de un autómata finito.
5. Qué es la secuencia nula.
6. Qué es una transición.
7. Cuándo una hilera es reconocida por un autómata finito.
8. Cuándo un autómata finito reconoce la secuencia nula.
9. Qué son estados equivalentes.
10. Qué son estados extraños.
11. En qué consiste simplificar un autómata finito.

1.4 DEFINICION
Un autómata finito (AF) está compuesto por:
1. Un conjunto finito de símbolos de entrada.
2. Un conjunto finito de estados.
3. Un estado designado como estado inicial.
4. Uno o más estados designados como estados de aceptación.
5. Un conjunto de transiciones.

Una transición es una función con la cual se determina el nuevo estado del autómata finito, con base en el estado actual y el símbolo de entrada:

NuevoEstado = **f**(estadoActual, símboloDeEntrada)

1.5 PRIMER EJEMPLO.
Construir autómata finito que reconozca cualquier secuencia de ceros y unos tal que el número de unos sea impar.

Ejemplos: 1; 01; 10; 00010; 11100101010001;

Quizá la parte más "complicada" en la construcción de un autómata finito es la definición de los estados. Realmente, cada diferente situación que se presente cuando

se está haciendo un proceso de reconocimiento, amerita un estado. Si estamos tratando de reconocer secuencias de ceros y unos en las cuales el número de unos sea impar, se pueden presentar dos situaciones: una, que el número de unos sea par, y otra, que el número de unos sea impar. Por consiguiente, tendremos dos estados: uno, en el cual se identifique que el número de unos es par y otro, en el cual se identifique que el número de unos es impar. Llamemos estos estados UP (unos pares) y UI (unos impares).

Construyamos nuestro autómata finito con base en la definición de autómata finito.

1. Símbolos de entrada = {0, 1}
2. Estados = {UP, UI}
3. Estado inicial = UP
4. Estado de aceptación = UI
5. Transiciones:

Las transiciones se pueden expresar de dos formas: una, mediante un **diagrama de burbujas**, y, otra mediante una **tabla de transiciones**.

Diagrama de burbujas: por cada estado dibujamos un círculo, y luego, analizamos cada estado para todos los símbolos de entrada. El diagrama para nuestro ejemplo es:

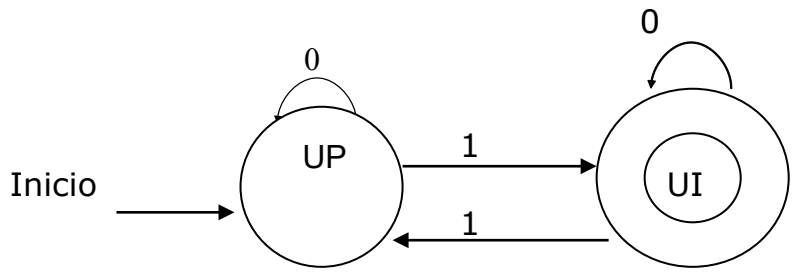


Figura 1.1

El estado inicial se señala con una flecha que diga inicio, y el, o los estados de aceptación se encierran en doble círculo. De esta forma se podrá visualizar fácilmente cuál es el estado inicial y cuáles son los estados de aceptación.
Tabla de transiciones: Es una matriz de **m** filas y **n** columnas, siendo **m** el número de estados y **n** el número de símbolos de entrada. La primera fila representa siempre el estado inicial del autómata finito. En la parte externa derecha de cada fila colocaremos un 1 o un 0. Uno (1) significa que el estado correspondiente a esa fila es de aceptación, y cero (0) significa que estado correspondiente a esa columna es de rechazo. El contenido de cada celda es el estado hacia el cual se hace transición cuando el autómata finito esté en el estado correspondiente a esa fila y el símbolo de entrada sea el de la columna correspondiente.

Para nuestro ejemplo la tabla de transiciones es:

	0	1	
UP	UP	UI	0
UI	UI	UP	1

Tabla 1.1

1.6 SEGUNDO EJEMPLO

Construir autómata finito que reconozca cualquier secuencia de unos y ceros tal que el número de ceros sea par y el número de unos sea impar.

Definamos primero los estados: al tratar de reconocer que el número de ceros sea par y el número de unos sea impar, se nos pueden presentar cuatro situaciones:

una, que el número de ceros sea par y el número de unos también sea par;
dos, que el número de ceros sea par y que el número de unos sea impar;
tres, que el número de ceros sea impar y que el número de unos sea par,
cuatro, que el número de ceros sea impar y que el número de unos también sea impar.

Cada situación amerita un estado. Por consiguiente tendremos cuatro estados, los cuales llamaremos **CPUP** (**C**eros **P**ares **U**nos **P**ares), **CPUI** (**C**eros **P**ares **U**nos **I**mpares), **CIUP** (**C**eros **I**mpares **U**nos **P**ares) y **CIUI** (**C**eros **I**mpares **U**nos **I**mpares), respectivamente.

De estos cuatro estados, **el estado inicial es** el estado **CPUP**, ya que inicialmente no ha entrado ningún símbolo, por lo tanto, el número de ceros es cero y el número de unos también es cero. El cero, para este ejemplo, lo admitiremos como número par.

El estado de aceptación es el que hemos denominado **CPUI**, ya que es el que representa que el número de ceros es par y el número de unos impar, y esto es lo que queremos reconocer.Dado lo anterior, construyamos formalmente nuestro autómata finito:

Símbolos de entrada = {0, 1}.
Estados = {CPUP, CPUI, CIUP, CIUI}.
Estado inicial = CPUP.
Estado de aceptación = CPUI.
Transiciones:

1. **Diagrama de transiciones:**

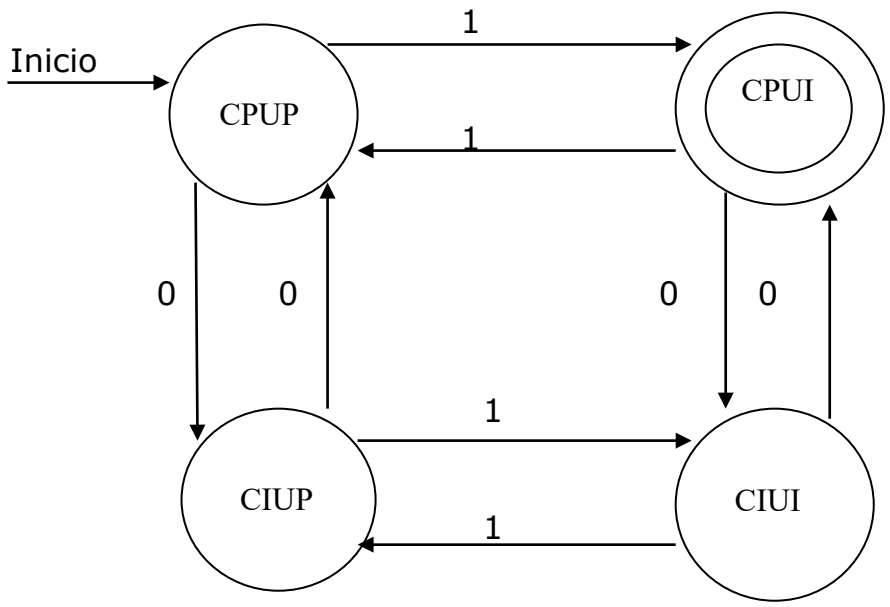


Figura 1.2

2. Tabla de transiciones:

	0	1	
CPUP	CIUP	CPUI	0
CPUI	CIUI	CPUP	1
CIUP	CPUP	CIUI	0
CIUI	CPUI	CIUP	0

Tabla 1.2

Si en el ejemplo anterior cambiamos la condición "que el número de ceros sea par **y** el número de unos impar" por la condición "que el número de ceros sea par **o** el número de unos impar", la modificación que hay que hacerle al autómata anterior está en lo referente a cuáles estados son de aceptación y cuáles de rechazo. El único estado de rechazo es el estado CIUP, los demás estados son de aceptación. El autómata finito quedará como se muestra en la tabla 2a.

	0	1	
CPUP	CIUP	CPUI	1
CPUI	CIUI	CPUP	1
CIUP	CPUP	CIUI	0
CIUI	CPUI	CIUP	1

Tabla 1.2a

Hemos presentado dos ejemplos, en los cuales los estados los hemos definido con base en el enunciado de lo que hay que reconocer.

En general, los estados se definen con base en el enunciado de lo que hay que reconocer o con base en el conocimiento que se tenga de lo que hay que reconocer.

1.7 TERCER EJEMPLO

Construir autómata finito que reconozca constante numéricas.

Para construir este autómata finito definamos los estados, los cuales construiremos, con base en el conocimiento que tenemos de la construcción de constantes numéricas.

Ejemplos de constantes numéricas válidas son:

a) 314
b) -4.85
c) 63.
d) .675
e) 5.14E-45
f) 4E3
g) +562E+32
h) -3

Analicemos los ejemplos anteriores con el fin de definir los estados.

Definamos "0" el estado inicial, el estado en el cual se halla el autómata finito antes de que entre símbolo alguno.

El primer símbolo que puede entrar, para que la constante sea válida, es:

un signo (+ o -),
un dígito (0, 1, 2, 3, 4, 5, 6, 7, 8, 9),
o un punto (.)

Si entra un signo, definimos un estado que identifique que entró un signo: llamemos este estado el estado "1". Por consiguiente, si estamos en el estado inicial y entra un signo, haremos transición hacia el estado "1". El autómata finito quedará en el estado "1".

El estado "1" significa: entró signo para la constante.

Si entra un dígito, definimos un estado que identifique que entró un dígito: llamemos este estado el estado "2". Por consiguiente, si estamos en el estado inicial y entra un dígito, haremos transición hacia el estado "2". El autómata finito quedará en el estado "2".

El estado "2" significa: entró dígito correspondiente a la parte entera de la constante.

Si entra un punto definimos un estado que identifique que entró un punto de primero, es decir, antes de cualquier dígito: llamemos este estado el estado "8". Por consiguiente, si estamos en el estado inicial y entra un punto haremos transición hacia el estado "8". El autómata finito quedará en el estado "8".

El estado "8" significa: entró punto de primero, es decir, antes de cualquier dígito.

Cualquier otro símbolo que entre, estando en el estado "0", significa que la constante es inválida. Por consiguiente, si estando en el estado "0" entra una E, la transición será hacia el estado "9".

El estado "9" significa: constante inválida.

Análisis del estado "1".
El estado "1" significa que entró signo para la constante.

Después del signo, para que la constante sea válida, debe entrar un punto o un dígito:
Si entra un dígito se hace transición hacia el estado "2", si entra un punto se hace transición hacia el estado "8". Cualquier otro símbolo que entre, signo o E, invalida la constante.

Análisis del estado "2".
Cuando el autómata finito se halla en el estado 2, es decir, entró un dígito de la parte entera las opciones son: una, que entre otro dígito, en cuyo caso el autómata finito continúa en el estado 2; dos, que entre un punto, en cuyo caso hacemos transición hacia un nuevo estado, el cual llamaremos es estado "3".

El estado "3" significa: entró punto para parte decimal. OJO, punto después de algún dígito.

Tres, entra la letra E para notación científica. Esta es una nueva situación, la cual amerita un nuevo estado, el cual llamaremos el estado "5"**.**

El estado "5" significa: entró E para constante en notación científica.

Si estamos en el estado "2" y entra un signo haremos transición hacia el estado "9", que como ya hemos dicho representa constante inválida.

Análisis del estado "3".
Cuando el autómata finito se halla en el estado "3", es decir, entró punto para la parte decimal, las opciones son: una, que entre un dígito, lo cual significa que será un dígito perteneciente a la parte decimal, y esto amerita un nuevo estado: los dígitos de la parte decimal son una situación diferente a los dígitos de la parte entera. Cuando se están procesando los dígitos de la parte entera, si entra un punto, la constante sigue siendo válida, pero si se están procesando los dígitos de la parte decimal y aparece un punto la constante se invalida. Por tanto, los dígitos de la parte decimal serán un nuevo estado, el cual llamaremos el estado "4".

El estado "4" significa: entró dígito correspondiente a la parte decimal de la constante.

Dos, que entre una **E**, en cuyo caso hacemos transición hacia el estado "5", el cual, como hemos visto, representa que entró la E para constante en notación científica. Si entra un signo u otro punto se invalida la constante, por tanto la transición será hacia el estado "9".

Análisis del estado "4".
Cuando el autómata finito se halla en el estado "4", es decir, entró un dígito de la parte decimal las opciones son: una, que entre otro dígito, en cuyo caso el autómata finito continúa en el estado 4; dos, que entre una E, en cuyo caso hacemos transición hacia el estado "5", el cual como ya hemos visto, representa que entró la E para constante en notación científica; cualquier otro símbolo que entre, punto o signo, invalida la constante, por tanto la transición será hacia el estado "9".

Análisis del estado "5".
Cuando el autómata finito se halla en el estado "5", es decir, que entró E para constante en notación científica, será válido que entre un signo o que entre un dígito. Si entra un signo significa que es el signo para el exponente. Esta es una nueva situación, por tanto, como ya hemos dicho amerita un nuevo estado, el cual llamaremos el estado "6".

El estado "6" significa: entró signo para exponente de constante en notación científica.

Si entra un dígito, éste será un dígito para el exponente de constante en notación científica, lo cual es una nueva situación: los dígitos del exponente de constante en notación científica son diferentes de los dígitos de la parte entera y de la parte decimal por lo siguiente: en dígitos de la parte entera se acepta que llegue un punto, en dígitos del exponente un punto invalida la constante; en dígitos de la parte

decimal se acepta que llegue una E, en dígitos del exponente una E invalida la constante. Por tanto los dígitos del exponente son una nueva situación y por consiguiente ameritan un nuevo estado. Llamemos este estado el estado "7".

El estado "7" significa: entró dígito de exponente para constante en notación científica.

Estando en el estado "5" si llega un punto o una E se invalida la constante, por tanto la transición será hacia el estado "9".

Análisis del estado "6".
Cuando el autómata finito se halla en el estado "6", es decir, entró signo para constante en notación científica, sólo se acepta que llegue un dígito. Cualquier otro símbolo que llegue invalida la constante. Por tanto, si llega un dígito la transición es hacia el estado "7", el cual significa que entró dígito para exponente de constante en notación científica. Cualquier otro símbolo la transición es hacia el estado "9".

Análisis del estado "7".
Cuando el autómata finito se halla en el estado "7", es decir, entró dígito para constante en notación científica, sólo se acepta que llegue otro dígito. Cualquier otro símbolo que llegue invalida la constante. Por tanto, si llega un dígito la transición es permanecer en el estado "7", el cual significa que entró dígito para exponente de constante en notación científica. Cualquier otro símbolo la transición es hacia el estado "9".

Análisis del estado "8".
Cuando el autómata finito se halla en el estado "8", es decir, entró punto antes de cualquier dígito, sólo se acepta que llegue un dígito. Cualquier otro símbolo que llegue invalida la constante. Por tanto, si llega un dígito la transición es hacia el estado "4", el cual significa que entró dígito para parte decimal de la constante. Cualquier otro símbolo la transición es hacia el estado "9".

Ahora, para determinar cuáles son los estados de aceptación veamos en qué situaciones puede terminar una constante para que sea válida.

Una constante válida puede terminar en: dígito de parte entera (estado "2"), punto después de dígito (estado "3"), dígito de parte decimal (estado "4") y dígito de exponente (estado "7"). Por consiguiente los estados de aceptación son los estados "2", "3", "4" y "7".

Con base en todo lo anterior construyamos nuestro autómata finito para reconocer constantes numéricas.

En la tabla 3 presentamos la tabla de transiciones correspondiente a dicho autómata.

En los símbolos de entrada utilizamos una **d** para denotar dígito y una **s** para denotar signo.

	d	.	E	s	
0	2	8	9	1	0
1	2	8	9	9	0
2	2	3	5	9	1
3	4	9	5	9	1
4	4	9	5	9	1
5	7	9	9	6	0
6	7	9	9	9	0
7	7	9	9	9	1
8	4	9	9	9	0
9	9	9	9	9	0

Tabla 1.3

Antes de continuar con autómatas finitos definamos unos conceptos que serán de gran utilidad en el resto del curso.

1.8 CONCEPTOS BASICOS

Símbolo de fin de secuencia: es el símbolo que permitirá identificar cuando se ha terminado de procesar una secuencia de símbolos de entrada. Este símbolo será: ┤

Secuencia nula: es la ausencia de símbolos. Es necesario definir un símbolo que represente la ausencia de símbolos. En nuestro curso usaremos el símbolo **λ**. En muchos otros textos y en páginas web utilizan el símbolo **Є** con el mismo propósito. **Un autómata finito acepta la secuencia nula, sí y sólo sí, su estado inicial es de aceptación.**

Conjunto regular: es el conjunto de secuencias reconocidas por un autómata finito. Por consiguiente, para los ejemplos que hemos desarrollado, cualquier secuencia de ceros y unos tal que el número de unos sea impar, es un conjunto regular (primer ejemplo); cualquier secuencia de ceros y unos tal que el número de ceros sea par y el número de unos sea impar, es otro conjunto regular (segundo ejemplo).

Con base en un autómata finito se elabora el programa de computador con el cual se reconoce ese conjunto de secuencias. Si consideramos el autómata finito de la tabla 3 un algoritmo para reconocer constantes numéricas tendría la siguiente forma:

```
estado = 0
lea(símbolo)
while  símbolo != ┤  do
      casos de estado
            "0": casos de símbolo
                        "d": estado = 2
                        ".": estado = 8
                        "E": estado = 9
                        "s": estado = 1
            "1": casos de símbolo
                        "d": estado = 2
```

```
                    ".": estado = 8
                    "E": estado = 9
                    "s": estado = 9
            "2": casos de símbolo
                    "d": estado = 2
                    ".": estado = 8
                    "E": estado = 5
                    "s": estado = 9
            .          .     .
            .          .     .
            .          .     .
        fin(casos de estado)
        lea(símbolo)
    end(while)
```

Como hemos visto, se ha construido un autómata finito para reconocer constantes numéricas. Este autómata finito lo hemos construido de una manera artesanal, es decir, con base en el conocimiento que tenemos de cómo se construyen las constantes numéricas en lenguajes de programación. En general, la construcción de autómatas finitos, en muchas situaciones, es algo que está en el estado del arte. Es algo parecido a elaborar algoritmos.

1.9 Estados equivalentes

Considerando nuevamente el autómata finito de la tabla 3, observamos que los estados "3" y "4" tienen exactamente las mismas transiciones y además, ambos estados son de aceptación.

Estados con esta característica se denominan **estados equivalentes**.

Cuando un autómata finito tiene estados equivalentes es conveniente simplificarlo ya que esto redunda en que nuestro programa sea más corto y más eficiente.

Para simplificar el autómata finito de la tabla 3, basta con agrupar los estados "3" y "4" en uno solo.

Llamemos este estado, el estado "34", para significar que se está agrupando el estado "3" con el estado "4". Y cualquier transición que haya hacia el estado "3" o hacia el estado "4" ya será hacia el estado "34". El autómata finito de la tabla 3, simplificado aparece en la tabla 4.

	d	.	E	s	
0	2	8	9	1	0
1	2	8	9	9	0
2	2	34	5	9	1
34	34	9	5	9	1
5	7	9	9	6	0
6	7	9	9	9	0
7	7	9	9	9	1
8	34	9	9	9	0
9	9	9	9	9	0

Tabla 1.4

La relación de equivalencia en autómatas finitos es similar a la relación de equivalencia en sentido matemático, es decir, tiene las mismas tres características: es reflexiva, es simétrica y es transitiva: reflexiva, todo estado es equivalente consigo mismo; simétrica, si el estado "a" es equivalente con el estado "b" entonces el estado "b" es equivalente con el estado "a", y transitiva, si "a" es equivalente a "b" y "b" es equivalente a "c" entonces "a" es equivalente a "c".

Por lo general, los estados equivalentes no se detectan por inspección, como en el caso del autómata finito que reconoce constantes numéricas. Por consiguiente se hace necesario definir un procedimiento para detectar estados equivalentes. Este procedimiento se basa en las características que deben tener los estados equivalentes.

Dos o más estados son equivalentes sí y sólo sí cumplen dos condiciones:

1. **Condición de compatibilidad:** dos o más estados son equivalentes si todos ellos son de aceptación o de rechazo. El hecho de que un estado sea de aceptación y el otro de rechazo es condición suficiente para decir que dichos estados no son equivalentes.
2. **Condición de propagación:** dos o más estados son equivalentes si para un mismo símbolo de entrada, hacen transición hacia estados equivalentes.

Con base en estas dos condiciones se ha desarrollado un método para determinar estados equivalentes. Dicho método se denomina el método de particiones.

Para ilustrar cómo funciona el método de particiones consideremos el autómata finito de la tabla 5.

	a	b	
0	0	3	0
1	2	5	0
2	2	7	0
3	6	7	0
4	1	6	1
5	6	5	0
6	6	3	1
7	6	3	0

Tabla 1.5

La aplicación de la condición de compatibilidad consiste en crear inicialmente dos particiones: una con los estados de aceptación y otra con los estados de rechazo.

Nuestras particiones son:

$$P_0 = \{0, 1, 2, 3, 5, 7\}$$
$$P_1 = \{4, 6\}$$

La aplicación de la condición de compatibilidad consiste en analizar los estados de cada una de las particiones para cada uno de los símbolos de entrada y examinar en cuáles particiones quedan los estados resultantes de las transiciones con ese símbolo de entrada. Si los estados resultantes de las transiciones están todos en una misma partición no habrá necesidad de dividir la partición que se está analizando, si dichos

estados están en diferentes particiones hay que dividir la partición que se está analizando de acuerdo a las particiones hacia las cuales se haya hecho transición.

Comencemos analizando la partición **0** cuando el símbolo de entrada es una **a**.

$P_{0/a}$
0 → 0
1 → 2
2 → 2
3 → 6
5 → 6
7 → 6

Como se podrá observar las transiciones son hacia los estados 0, 2 y 6.
Los estados 0 y 2 están en una partición (la partición 0) y el estado 6 está en otra partición (la partición 1). Por consiguiente, se deduce que los estado 0, 1 y 2 NO son equivalentes con los estados 3, 5 y 7, puesto que, cuando el símbolo de entrada es una **a** las transiciones son hacia estados que están en diferentes particiones. Por lo tanto hay que dividir la partición 0. La partición 0 quedará: $P_0 = \{0, 1, 2\}$, y creamos una nueva partición, la cual llamaremos partición 2. La partición 2 queda: $P_2 = \{3, 5, 7\}$.

Continuamos analizando la partición 0 cuando el símbolo de entrada sea una **b**.

$P_{0/b}$
0 → 3
1 → 5
2 → 7

Las transiciones son hacia los estados 3, 5 y 7, los cuales se hallan en una misma partición (la partición 2), por tanto no da lugar a dividir la partición 0 nuevamente.

Analicemos ahora la partición 1 cuando el símbolo de entrada es una **a**.

$P_{1/a}$
4 → 1
6 → 6

Las transiciones son hacia los estado 1 y 6, los cuales se hallan en diferentes particiones. El estado 1 se halla en la partición 0, mientras que el estado 6 se halla en la partición 1, por tanto hay que dividir la partición 1. La partición 1 queda: $P_1 = \{4\}$, y creamos una nueva partición, la cual llamaremos la partición 3: $P_3 = \{6\}$.

El hecho de haber tenido que dividir la partición 1 implica que hay que revisar nuevamente las particiones que han sido analizadas previamente, ya que puede suceder que en un análisis anterior algunos estados hayan sido encontrados equivalentes porque los estados 4 y 6 (según el ejemplo) estaban en una misma partición, y ya quedaron en particiones diferentes.

En general, cada vez que haya que dividir una partición hay que analizar nuevamente todas las particiones.

Este proceso se repite hasta que se haya dado una pasada completa y no hubo que dividir ninguna partición.
Volviendo a nuestro ejemplo, tenemos hasta el momento 4 particiones:

$P_0 = \{0, 1, 2\}$
$P_1 = \{4\}$
$P_2 = \{3, 5, 7\}$
$P_3 = \{6\}$

Las cuales, al analizarlas nuevamente para cada uno de los símbolos de entrada tenemos:

$P_{0/a}$	$P_{0/b}$	$P_{2/a}$	$P_{2/b}$
0 → 0	0 → 3	3 → 6	3 → 7
1 → 2	1 → 5	5 → 6	5 → 5
2 → 2	2 → 7	7 → 6	7 → 3

Y se podrá observar que las transiciones son hacia estados que hallan en una misma partición, por tanto no hay que efectuar más divisiones.

Estados que queden en una misma partición son estados equivalentes.

Por consiguiente los estados 0, 1 y 2 son equivalentes, y los estados 3, 5 y 7 también son equivalentes.

Procedamos entonces a simplificar el autómata finito, el cual queda como en la tabla 6.

	a	b	
012	012	357	0
357	6	357	0
4	012	6	1
6	6	357	1

Tabla 1.6

1.10 Estados extraños

Sucede también, que en un autómata finito puede haber otra clase de estados, los cuales se denomina estados extraños, que son estados hacia los cuales el autómata finito nunca llega.

Tener estados extraños en un autómata finito es algo que no es aceptable, dado que lo único que se logra con ello es que el programa resultante tendrá instrucciones inoficiosas. Por tanto debemos proceder a eliminar los estados extraños.

Igual que con los estados equivalentes, hay situaciones en las cuales algunos estados extraños se pueden detectar por inspección, sin embargo, hay situaciones en las cuales éstos no se detectan por inspección. Consideremos el autómata finito de la tabla 7.

Si observamos bien las transiciones, nos damos cuenta que no hay una sola transición hacia el estado 4. Qué está haciendo el estado 4 en dicho autómata?.

Absolutamente nada, por consiguiente, podremos eliminarlo y no afecta para nada el funcionamiento del autómata finito.

Sin embargo, puede que haya más estados extraños.

	a	b	
0	1	5	0
1	2	7	1
2	2	5	1
3	5	7	0
4	5	6	0
5	3	1	0
6	8	0	1
7	0	1	1
8	3	6	0

Tabla 1.7

Para detectar estados extraños y eliminarlos, hacemos el siguiente proceso: reconstruir el autómata finito, comenzando con el estado inicial, e incluyendo únicamente los estados hacia los cuales se haga transición.

Veamos cómo queda nuestro autómata finito.

	a	b	
0	1	5	0
1	2	7	1
5	3	1	0
2	2	5	1
7	0	1	1
3	5	7	0

Tabla 1.8

Los estados 4, 6 y 8 eran estados extraños.

En general, cuando se nos pida simplificar un autómata finito, seguimos dos pasos: primero eliminamos estados extraños y luego buscamos estados equivalentes y procedemos a terminar la simplificación, agrupando los estados equivalentes.

EJERCICIOS PROPUESTOS

1. Construya autómata finito que reconozca secuencias de ceros y unos con las siguientes características

    a) El número de unos sea par y el número de ceros impar. Para que el número de unos sea par tiene que haber mínimo dos unos.
    b) Cada pareja de ceros debe estar entre unos.
    c) Haya un número impar de veces de la ocurrencia del patrón 00 admitiendo traslapos.

d) Haya un número par de veces de la ocurrencia del patrón 00 sin admitir traslapos.
e) Cada ocurrencia del patrón 11 esté seguida de un cero.
f) Cada tercer símbolo sea un uno.
g) Haya al menos un uno.
h) Toda secuencia que contenga exactamente tres unos.
i) Toda secuencia en la cual cada uno esté precedido y seguido por un cero.
j) Cualquier secuencia que termine en 001.
k) Contenga la pareja 00 ó la pareja 11.
l) Contenga la secuencia 11 y no contenga la secuencia 00.
m) Comience con 00 y termina en 1

2. Describa con palabras el conjunto de secuencias reconocido por cada una de los siguientes autómatas finitos

	0	1	
A	B	A	0
B	B	C	0
C	C	C	1

(a)

	0	1	
A	B	C	0
B	C	B	1
C	C	C	0

(b)

	0	1	
A	B	C	0
B	D	B	1
C	C	D	1
D	D	D	0

(c)

	0	1	
A	B	A	0
B	D	C	0
C	C	D	1
D	D	D	0

(d)

	1	*	C	
A	B	F	F	0
B	F	C	F	0
C	C	D	C	0
D	E	C	C	0
E	F	F	F	1
F	F	F	F	0

(e)

	0	1	
A	B	C	0
B	B	D	0
C	C	C	0
D	C	D	1

(f)

3. Simplifique los siguientes autómatas finitos

	0	1	
A	A	C	0
B	G	D	1
C	F	E	0
D	A	D	1
E	A	D	0
F	G	F	1
G	G	C	0

(a)

	0	1	
P	Q	R	0
Q	S	P	1
S	R	U	0
U	R	Q	1
T	T	S	0
R	V	P	1
V	Q	U	0

(b)

	0	1	
A	D	A	1
B	E	A	1
C	D	E	0
D	B	F	0
E	A	G	0
F	A	D	1
G	B	E	1

(c)

SOLUCIONES

1a. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que el número de unos sea par y el número de ceros impar. Para que el número de unos sea par tiene que haber mínimo dos unos.

Símbolos de entrada = {0, 1}
Estados = {NUCP: no unos, ceros pares;
NUCI: no unos, ceros impares;
CPUP: ceros pares, unos pares;
CPUI: ceros pares, unos impares;
CIUP: ceros impares, unos pares;
CIUI: ceros impares unos impares}
Estado inicial = NUCP
Estado de aceptación = CIUP
Tabla de transiciones:

	0	1	
NUCP	NUCI	CPUI	0
NUCI	NUCP	CIUI	0
CPUP	CIUP	CPUI	0
CPUI	CIUI	CPUP	0
CIUP	CPUP	CIUI	1
CIUI	CPUI	CIUP	0

1c. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que el patrón 00 aparezca un número impar de veces, admitiendo traslapos.

Símbolos de entrada = {0, 1}
Estados = {A: número par de ocurrencias del patrón 00 que no termina en cero;
B: número par de ocurrencias del patrón 00 que termina en cero;
C: número impar de ocurrencias del patrón 00 que termina en cero;
D: número impar de ocurrencias del patrón 00 que termina en uno}
Estado inicial = A
Estados de aceptación = {C, D}
Tabla de transiciones:

	0	1	
A	B	A	0
B	C	A	0
C	B	D	1
D	C	D	1

1e. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que cada ocurrencia del patrón 11 esté seguida de un cero.

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía o que termina en cero;
B: Hilera que termina en 01 o es un uno solo;
C: hilera que termina en par de unos;
D: hilera que termina en 110;

err: hilera con mínimo tres unos seguidos}
Estado inicial = A
Estados de aceptación = {A, B}
Tabla de transiciones:

	0	1	
A	A	B	1
B	A	C	1
C	A	err	0
err	err	err	0

1g. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que tenga al menos un uno.

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía o de sólo ceros;
B: hilera que tiene al menos un uno}
Estado inicial = A
Estado de aceptación = B
Tabla de transiciones:

	0	1	
A	A	B	0
B	B	B	1

1i. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que cada uno esté precedido y seguido de un cero.

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía;
B: hilera sólo de ceros o que termina en 10;
C: hilera que termina en 1;
err: hilera con mínimo dos unos seguidos}
Estado inicial = A
Estados de aceptación = {A, B}
Tabla de transiciones:

	0	1	
A	B	err	1
B	B	C	1
C	B	err	0
err	err	err	0

1k. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que contenga la pareja 00 ó la pareja 11.

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía;
B: entró un cero, de primero o a continuación de un uno;
C: entró un uno, de primero o a continuación de un cero;
D: entró pareja de ceros o de unos}
Estado inicial = A
Estado de aceptación = {D}
Tabla de transiciones:

	0	1	
A	B	C	0
B	D	C	0
C	B	D	0
D	D	D	1

1m. Construya autómata finito que reconozca cualquier secuencia de ceros y unos tal que comience con 00 y termine en uno.

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía;
B: hilera con un cero;
C: hilera con mínimo dos ceros;
D: hilera que comenzó con dos ceros y termina en uno;
err: hilera que no comienza con dos ceros}
Estado inicial = A
Estado de aceptación = D
Tabla de transiciones:

	0	1	
A	B	err	0
B	C	err	0
C	C	D	0
D	C	D	1
err	err	err	0

2a. Describa con palabras el conjunto de secuencias reconocido por el siguiente autómata finito.

	0	1	
A	B	A	0
B	B	C	0
C	C	C	1

Reconoce secuencias de unos y ceros tal que al inicio puede haber cualquier cantidad de unos o ninguno, seguido mínimo de un cero con cualquier cantidad de ceros a continuación y terminando con un solo uno.

2c. Describa con palabras el conjunto de secuencias reconocido por el siguiente autómata finito.

	0	1	
A	B	C	0
B	D	B	1
C	C	D	1
D	D	D	0

Reconoce secuencias de unos y ceros tales que: si el primer símbolo es un cero, a continuación puede no venir ningún símbolo o cualquier cantidad de unos; si el primer símbolo es un uno, a continuación puede no venir ningún símbolo o cualquier cantidad de ceros.

2e. Describa con palabras el conjunto de secuencias reconocido por el siguiente autómata finito.

	1	*	c	
A	B	F	F	0
B	F	C	F	0
C	C	D	C	0
D	E	C	C	0
E	F	F	F	1
F	F	F	F	0

Reconoce hileras de unos, asteriscos y la letra c, tales que: comiencen con **1*** seguido de cualquier hilera compuesta por unos y la letra c, la cual puede ser nula, seguido por un asterisco, seguido por cualquier hilera compuesta con asteriscos y la letra c, la cual también puede ser nula y terminando con un uno.

3a. Simplifique el siguiente autómata finito.

	0	1	
A	A	C	0
B	G	D	1
C	F	E	0
D	A	D	1
E	A	D	0
F	G	F	1
G	G	C	0

El primer paso es determinar estados extraños y eliminarlos. Hacemos uso de la técnica descrita en el numeral 1.10 y nuestro autómata queda:

	0	1	
A	A	C	0
C	F	E	0
D	A	D	1
E	A	D	0
F	G	F	1
G	G	C	0

Se detectó que el estado B era un estado inalcanzable, por tanto se eliminó del autómata finito.

A continuación procedemos a detectar estados equivalentes: Comenzamos agrupándolos con base en la condición de compatibilidad.

$$P_0 = \{A, C, E, G\}$$
$$P_1 = \{D, F\}$$

Comencemos analizando la partición P_0 cuando el símbolo de entrada sea un cero:

$P_{0/0}$

A → A
C → F
E → A
G → G

El estado C hace transición hacia el estado F, el cual se halla en una partición diferente a la partición donde están los estados A y G, por consiguiente, el estado C no es equivalente con los estados A, E y G. Las particiones quedan:

$P_0 = \{A, E, G\}$
$P_1 = \{D, F\}$
$P_2 = \{C\}$

Continuamos analizando la partición P_0 cuando el símbolo de entrada sea un uno:

$P_{0/1}$
A → C
E → D
G → C

El estado E hace transición hacia el estado D, el cual se halla en una partición diferente a la cual se encuentra el estado C, por consiguiente, el estado E no es equivalente con los estados A y G. Las particiones quedan:

$P_0 = \{A, G\}$
$P_1 = \{D, F\}$
$P_2 = \{C\}$
$P_3 = \{E\}$

El hecho de haber tenido que hacer esta división implica que tendremos que analizar nuevamente la partición P_0 cuando el símbolo de entrada sea un cero.

Por ahora, continuemos analizando la partición P_1 cuando el símbolo de entrada sea un cero.

$P_{1/0}$
F → G
D → A

Aquí no hay lugar a división, puesto que las transiciones son hacia los estados Ay G, los cuales se hallan en una misma partición: la partición P_0.

Analicemos la partición P_1 cuando el símbolo de entrada sea un uno.

$P_{1/1}$
F → F
D → D

Aquí tampoco hay lugar a ninguna división puesto que las transiciones son hacia los estados F y D, los cuales se encuentran en la misma partición.

Analicemos nuevamente la partición P_0 cuando el símbolo de entrada sea un cero.

$P_{0/0}$
A → A
G → G

No hay lugar a división puesto que las transiciones son hacia los estados A y G, los cuales se hallan en una misma partición.

Hemos hecho el recorrido circular de análisis de particiones y no hubo que hacer divisiones, por tanto, las particiones quedaron:

$P_0 = \{A, G\}$
$P_1 = \{D, F\}$
$P_2 = \{C\}$
$P_3 = \{E\}$

Lo cual significa que los estados A y G son equivalentes y los estados D y F también, por consiguiente procedemos a simplificar el autómata finito, el cual queda:

	0	1	
AG	AG	C	0
C	DF	E	0
DF	AG	DF	1
E	AG	DF	0

3c. Simplifique el siguiente autómata finito

	0	1	
A	D	A	1
B	E	A	1
C	D	E	0
D	B	F	0
E	A	G	0
F	A	D	1
G	B	E	1

Comenzamos eliminando estados extraños: se reconstruye el autómata finito, partiendo del estado inicial e incluyendo únicamente los estados ahacia los cuales se haga transición:

	0	1	
A	D	A	1
D	B	F	0
B	E	A	1
F	A	D	1
E	A	G	0
G	B	E	1

Luego procedemos a buscar estados equivalentes.

$P_0 = \{D, E\}$
$P_1 = \{A, B, F, G\}$

Analizamos la partición P_0 cuando el símbolo de entrada sea un cero:
$P_{0/0}$

D → B
E → A

Los estados A y B se hallan en una misma partición, por tanto no se divide P_0.

Continuamos con la partición P0 cuando el símbolo de entrada es un uno:
$P_{0/1}$

D → F
E → G

Los estados F y G se hallan en una misma partición, la partición P_0 sigue igual.

Analicemos la partición P1 cuando el símbolo de entrada sea un 0:
$P_{1/0}$

A → D
B → E
F → A
G → B

Los estados A y B hacen transición hacia los estados D y E que están en la partición P_0, mientras que los estados F y G hacen transición hacia los estados A y B los cuales se hallan en la partición P_1. Por consiguiente, los estados A y B no son equivalentes con los estados F y G, lo que implica dividir la partición P_1. Las particiones quedan:

$P_0 = \{D, E\}$
$P_1 = \{A, B\}$
$P_2 = \{F, G\}$

Continuamos analizando la partición P_1 cuando el símbolo de entrada sea un uno:
$P_{1/1}$

A → A
B → A

Como se puede observar, no hay necesidad de efectuar más análisis, por tanto, los estados equivalentes son los estados D y E que se hallan en la partición P_0, los estados A y B que se hallan en la partición P_1 y los estados F y G que se hallan en la partición P_2.

Simplificando el autómata finito tenemos:

	0	1	
AB	DE	AB	1
DE	AB	FG	0
FG	DE	AB	1

MODULO 2
AUTOMATAS FINITOS NO DETERMINISTICOS

2.1 INTRODUCCION
Los autómatas finitos que hemos venido trabajando hasta ahora se denominan autómatas finitos determinísticos, lo cual significa, que el estado inicial es único y que las transiciones son también únicas. Vamos a tratar en este módulo otro tipo de autómatas finitos, los cuales pueden tener más de un estado inicial y/o las transiciones pueden ser hacia más de un estado. Dicho tipo de autómatas se denominan autómatas finitos no determinísticos.

2.2 OBJETIVOS
1. Conocer este modelo matemático, los elementos que lo componen, su funcionamiento y su representación.
2. Construir autómatas finitos no determinísticos.
3. Evaluar autómatas finitos no determinísticos
4. Convertir autómata finito no determinístico a determinístico.
5. Identificar cuándo usar un autómata finito no determinístico.

2.3 PREGUNTAS BÁSICAS
10. Qué es un autómata finito no determinístico.
11. En cuáles situaciones se debe usar un autómata finito no determinístico.
12. Cómo se identifica que un autómata finito es no determinístico.
13. Cuándo una hilera es reconocida por un autómata finito no determinístico.
14. Cómo se convierte un autómata finito no determinístico a determinístico.

2.4 Definición.
Son autómatas finitos en los cuales puede haber más de un estado inicial o las transiciones pueden ser hacia más de un estado.

Para ilustrar las características que se presentan cuando se tiene un autómata finito **NO determinístico** consideremos cómo es el proceso de reconocimiento cuando se tiene un autómata finito determinístico.

Consideremos el autómata finito de la tabla 1 (reconocer cualquier secuencia de ceros y unos tal que el número de unos sea impar), y la siguiente hilera de entrada: 01011┤.

El proceso de reconocimiento es como sigue:

UP $\xrightarrow{0}$ UP $\xrightarrow{1}$ UI $\xrightarrow{0}$ UI $\xrightarrow{1}$ UP $\xrightarrow{1}$ UI $\xrightarrow{\dashv}$ Acepte

Consideremos ahora el autómata finito no determinístico descrito en la tabla 9 que se muestra a continuación:

	0	**1**	
A	B	**C, D**	0
B	A	B	1
C	C	D	1
D	A	C	0

Tabla 2.1

Este autómata finito es no determinístico porque cuando se está en el estado **A** y el símbolo de entrada es un 1, la transición puede hacerse hacia el estado **C** o hacia el estado **D**. Veamos cómo sería el proceso para reconocer la hilera 10101.

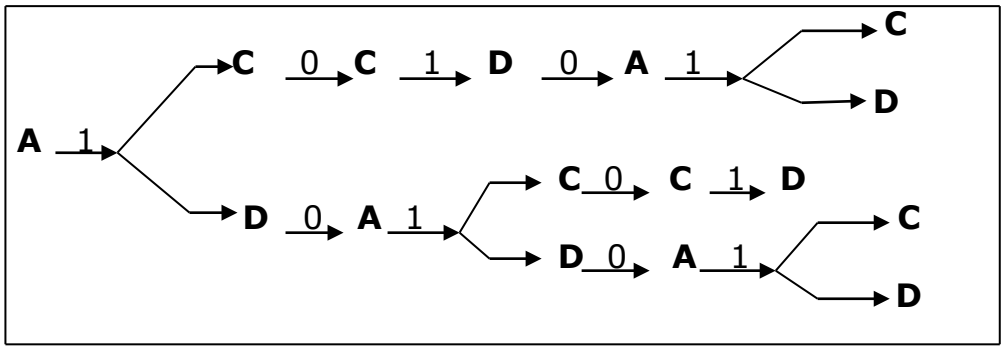


Figura 2.1

Al efectuar el proceso se pueden seguir los cinco diferentes caminos que se muestran en la figura 3. Por tres de esos caminos se llega al estado D, el cual es un estado de rechazo, y por los otros dos caminos se llega al estado C, el cual es de aceptación. La pregunta es: la secuencia 10101 se acepta o se rechaza?

La respuesta es: la secuencia 10101 se acepta.

En general, si se tiene un AFND (Autómata Finito No Determinístico) y una hilera de entrada, al procesar la secuencia para todos los posibles caminos, hay un camino que lo deja en un estado de aceptación, la secuencia se acepta. En otras palabras, funciona exactamente como la conjunción OR. Si se tiene una expresión lógica con sólo conjunciones OR y todas las variables de la expresión son falsas y hay una verdadera, el resultado de evaluar la expresión es verdadero.

El problema que se presenta cuando se va a implementar un AFND como programa de computador es que éste va a tener cierto grado de complejidad ya que se debe programar usando la técnica de retroceso (backtracking).

Con el fin de evitar esta complejidad se dispone de ciertas herramientas que permiten convertir autómatas finitos no determinísticos en autómatas finitos determinísticos, y de esta forma la implementación como programa de computador será bastante sencilla.

El caso es, que hay muchas situaciones en las cuales es más fácil diseñar autómata finito no determinísticos que determinísticos, por consiguiente el autómata finito se diseña no determinístico y se convierte a determinístico para implementarlo como programa de computador. Más aún, usted puede elaborar programa de computador que acepte como entrada un autómata finito no determinístico y lo convierta a determinístico.

2.5 Conversión de autómata finito no determinístico a determinístico.

Para convertir un AFND a AFD procedemos de una forma similar a como se eliminan los estados extraños de un autómata finito. Reconstruimos el autómata finito partiendo del estado inicial, incluyendo sólo los estados que vayan apareciendo, y teniendo cuidado de que **cuando se encuentre una transición no determinística, se hace la unión de dichas transiciones en un solo estado.** Al efectuar la unión

de dos estados, cuando se vayan a efectuar las transiciones correspondientes a esta unión, **se deberán unir también las transiciones correspondientes a ellos.**
Veamos cómo se convierte el AFND de la tabla 9 a AFD.

	0	1	
A	B	CD	0
B	A	B	1
CD	AC	CD	1
AC	BC	CD	1
BC	AC	BD	1
BD	A	BC	1

Tabla 2.2

Comenzamos construyendo el autómata finito de la tabla 10 con el estado inicial, el cual es el estado A. Las transiciones son: cuando entra un 0 la transición es hacia el estado B, y cuando entra un 1 la transición puede ser hacia el estado C o hacia el estado D (tabla 9), por consiguiente, cuando entra un uno hacemos la unión de estos estados, obteniendo el estado CD.
Incluimos en el autómata finito los estados B y CD.

El estado CD es un estado de aceptación puesto que incluye el estado C, el cual es de aceptación en el autómata finito original.

Las transiciones correspondientes al estado B son simplemente hacia A cuando entra un 0 y hacia B cuando entra un 1.

Veamos cómo son las transiciones correspondientes al estado CD.

En el AFND original (tabla 9), si se está en el estado C y entra un 0 la transición es hacia el estado C, y si se está en el estado D y entra un 0 la transición es hacia el estado A. La unión de estas dos transiciones nos da el estado AC, incluimos el estado AC en nuestro nuevo autómata finito. El estado AC también será de aceptación puesto que incluye el estado C, el cual es de aceptación.

En el AFND original (tabla 9), si se está en el estado C y entra un 1 la transición es hacia el estado D, y si se está en el estado D y entra un 1 la transición es hacia el estado C. La unión de estas dos transiciones nos da el estado CD, el cual ya tenemos incluido en nuestro nuevo autómata finito.

Haciendo un proceso similar para el estado AC que incluimos en nuestro nuevo autómata finito obtenemos los estados BC y CD, los cuales vamos incluyendo en nuestro nuevo autómata, y así obtenemos el AFD de la tabla 10.

2.6 PRIMER EJEMPLO

Consideremos un ejemplo en el cual es más sencillo construir AFND: Construir AFND que reconozca una de las siguientes palabras: GO, GOTO, TOO y ON.

Símbolos de entrada = {G, O, T, N}

Estados:

Definiremos el estado 0 como el estado inicial, es decir, el estado en el cual se halla el autómata finito antes de procesar símbolo alguno.

Para cada símbolo de cada palabra definimos un estado:

Para la palabra GO definimos los estados G1 y O1.
Para la palabra GOTO definimos los estados G2, O2, T1 y O3.
Para la palabra TOO definimos los estados T2, O4 y O5.
Para la palabra ON definimos los estados O6 y N.
Adicionalmente definimos el estado **err** para la situación de error.

La tabla de transiciones se muestra en la tabla 11.

Los espacios en blanco son todos, transición hacia el estado de error.

Cuando el autómata finito está en el estado inicial y entra una G, la transición puede ser hacia el estado G1 o hacia el estado G2, es decir, se va a reconocer la palabra GO o la palabra GOTO. Si elegimos ir hacia el estado G1 significa que vamos a reconocer la palabra GO y solamente la palabra GO, no más. Por consiguiente si se hace transición hacia el estado G1, en dicho estado sólo se reconocerá que entre una O, y la transición es hacia el estado O1, no más, puesto que si le damos la opción de ir también hacia O2 significaría que también se podría reconocer la palabra GOTO, y el estado G1 significa que vamos es a reconocer exclusivamente la palabra GO. ok?

Cuando se llegue al estado O1 significa que hemos reconocido la palabra GO. Por consiguiente el estado O1 es un estado de aceptación, y no se reconocerá ningún otro símbolo de ahí en adelante.

Si estando en el estado inicial, y entra una G, y hacemos transición hacia el estado G2, significa que vamos es a reconocer la palabra GOTO, no más. Por lo tanto, en el estado G2 sólo se reconoce que entre una O, cuya transición será hacia el estado O2 (primera O de GOTO). Estando en O2, sólo se acepta una T, y se hace transición hacia el estado T1 (T de GOTO). En T1 sólo se acepta una O, y se hace transición hacia el estado O3 (segunda O de GOTO). El estado O3 es un estado de aceptación.

	G	O	T	N	
0	G1, G2	O6	T2		0
G1		O1			0
O1					1
G2		O2			0
O2			T1		0
T1		O3			0
O3					1
T2		O4			0
O4		O5			0
O5					1
O6				N	0
N					1
err					0

Tabla 2.3

De una manera análoga procedemos con las demás palabras.

2.7 VARIACIÓN AL PRIMER EJEMPLO

Tomemos este autómata, y con base en él, planteemos otro ejercicio:

Construir AFND que reconozca cualquier secuencia construida con las palabras GO, GOTO, TOO y ON.

Ejemplos de secuencias válidas:

GOGOGOTOONTOOONGOTO;
TOOONONGOON;
ONONGOGOTOO;

Para construir este AFND, basta con retomar el autómata finito del ejemplo anterior, con una pequeña modificación. Cuando se esté en un estado de aceptación, significa que se ha reconocido una palabra de las solicitadas. En este punto debemos pasar a reconocer otra palabra, por consiguiente, es como si estuviéramos en el estado inicial, y por lo tanto las transiciones de los estados de aceptación serán las mismas del estado inicial. El autómata finito se muestra en la tabla 2.4, en la cual hemos resaltado con amarillo las transiciones agregadas.

	G	O	T	N	
0	G1, G2	O6	T2		0
G1		O1			0
O1	G1, G2	O6	T2		1
G2		O2			0
O2			T1		0
T1		O3			0
O3	G1, G2	O6	T2		1
T2		O4			0
O4		O5			0
O5	G1, G2	O6	T2		1
O6				N	0
N	G1, G2	O6	T2		1
err					0

Tabla 2.4

2.8 OTRA VARIACIÓN AL PRIMER EJEMPLO

Agreguemos otra condición a este ejercicio.

Construir AFND que reconozca cualquier secuencia construida con las palabras GO, GOTO, TOO y ON, permitiendo traslapos. Un traslapo se presenta cuando una misma letra es compartida por dos palabras. Por ejemplo: GON, la letra O está compartida por la palabra GO y la palabra ON; TOON, la letra O está compartida por las palabras TOO y ON; GOTON, la letra O está compartida por las palabras GOTO y ON.

Teniendo en cuenta los ejemplos anteriores, estas son las únicas situaciones en las que se pueden presentar traslapos. Por consiguiente, para diseñar un autómata finito que reconozca cualquier secuencia construida con las palabras GO, GOTO, TOO y ON, permitiendo traslapos, basta con tomar el autómata finito de la tabla 12 e incluirle una transición hacia el estado N en aquellos sitios donde se identifica una palabra

terminada en O. Es decir, en los estados O1, O3 y O5. El autómata queda como en la tabla 13.

Con este proceso que hemos seguido para construir un autómata finito que reconozca cualquier secuencia construida con las palabras GO, GOTO, TOO y ON, permitiendo traslapos, creo que se puede percibir que es más fácil hacerlo con un AFND que construir AFD de una vez.

	G	O	T	N	
0	G1, G2	O6	T2		0
G1		O1			0
O1	G1, G2	O6	T2	N	1
G2		O2			0
O2			T1		0
T1		O3			0
O3	G1, G2	O6	T2	N	1
T2		O4			0
O4		O5			0
O5	G1, G2	O6	T2	N	1
O6				N	0
N	G1, G2	O6	T2		1
err					0

Tabla 2.5

Si quisiéramos que nuestro autómata finito de la tabla 13, además de reconocer cualquier secuencia construida con las palabras GO, GOTO, TOO y ON, también reconozca la secuencia nula, basta con definir el estado inicial como estado de aceptación.

En general, existen algunas situaciones en las cuales es más cómodo construir AFND. Estas situaciones son:

1. Reconocimiento de hileras que tengan un prefijo dado. Por ejemplo, cualquier secuencia de ceros y unos que comience con 001.

2. Reconocimiento de hileras que tengan un sufijo dado. Por ejemplo, cualquier secuencia de ceros y unos que termine con 011.

3. Reconocimiento de hileras que sean la unión de hileras con condiciones dadas. Por ejemplo, cualquier hilera de ceros y unos con el número de ceros par o el número de unos impar.

2.9 UNIÓN E INTERSECCIÓN DE AUTÓMATAS FINITOS

Consideremos los siguientes autómatas:

1. que reconoce cualquier secuencia de ceros y unos tal que el número de ceros sea par (tabla 14).

	0	1	
CP	CI	CP	1
CI	CP	CI	0

Tabla 2.6

2. que reconoce cualquier secuencia de ceros y unos tal que el número de unos sea impar (tabla 15).

	0	1	
UP	UP	UI	0
UI	UI	UP	1

Tabla 2.7

Con base en estos dos autómatas finitos construiremos otros dos: uno, que reconozca cualquier secuencia de ceros y unos tal que el número de ceros sea par **ó** el número de unos sea impar, y otro, que reconozca cualquier secuencia de ceros y unos tal que el número de ceros sea par **y** el número de unos sea impar.

Comencemos con el primero de ellos: construir autómata finito que reconozca cualquier secuencia de ceros y unos tal que el número de ceros sea par **o** el número de unos sea impar. Vamos a hacer la unión del autómata finito de la tabla 14 con el autómata finito de la tabla 15, considerándolo como un AFND con dos estados iniciales. Cuando un AFND tiene más de un estado inicial, éstos se señalan con una flecha. La unión de estos dos autómatas finitos en uno solo se muestra en la tabla 16.

		0	1	
→	CP	CI	CP	1
	CI	CP	CI	0
→	UP	UP	UI	0
	UI	UI	UP	1

Tabla 2.8

Efectuamos la unión de los estados iniciales CP y UP, obteniendo el estado CPUP, y vamos incluyendo los estados que van apareciendo al hacer la unión de sus respectivas transiciones. **Recuerde que la unión de dos o más estados es de aceptación, cuando incluye mínimo un estado de aceptación**. El autómata finito obtenido se muestra en la tabla 17.

	0	1	
CPUP	CIUP	CPUI	1
CIUP	CPUP	CIUI	0
CPUI	CIUI	CPUP	1
CIUI	CPUI	CIUP	1

Tabla 2.9

Veamos ahora el segundo caso: construir autómata finito que reconozca cualquier secuencia de ceros y unos tal que el número de ceros sea par **y** el número de unos sea impar. Hacemos la intersección del autómata finito de la tabla 14 con el autómata finito de la tabla 15, la cual consiste en la unión de dichos autómatas teniendo en cuenta que la unión de dos o más estados será de aceptación sí y sólo sí todos los estados son de aceptación. El autómata resultante se muestra en la tabla 18.

	0	1	
CPUP	CIUP	CPUI	0
CIUP	CPUP	CIUI	0
CPUI	CIUI	CPUP	1
CIUI	CPUI	CIUP	0

Tabla 2.10

Es importante resaltar en este punto: **La intersección de dos autómatas finitos es similar a la unión de dos autómatas finitos teniendo en cuenta que la unión de dos o más estados sólo será de aceptación si todos los estados son de aceptación**.

EJERCICIOS PROPUESTOS

1. Convierta a determinístico y luego simplifique los siguientes autómatas finitos.

	0	1	
A	A,B		0
B		B,C	0
C	C		1

(a)

	0	1	
A	B		0
B		C,A	0
C			1

(b)

	0	1	
A	B		0
B	C,D	B	0
C	B		0
D			1

(c)

	a	b	
1	2,3	2	0
2		1	0
3	1	3	1

(d)

2. Para el autómata finito (a) del primer punto cuáles de las siguientes secuencias no son aceptadas.

   a) 010101010
   b) 001110100
   c) 000110000
   d) 001110010
   e) 011111110

3. Construya autómata finito que reconozca (0+1)* tal que si contiene la subhilera 001 entonces debe contener la subhilera 110.

4. Construya autómatas finitos no determinísticos que reconozcan:

   a) Cualquier secuencia de ceros y unos que termine en 001.
   b) Cualquier secuencia de ceros y unos tal que comience con 00 y termine con 11.
   c) Enteros impares representados en binario.
   d) Enteros múltiplos de cuatro representados en binario.
   e) (real, read, ready, return)*

SOLUCIONES

1a. Convierta a determinístico y simplifique el siguiente autómata finito

	0	1	
A	A,B		0
B		B,C	0
C	C		1

El primer paso es convertirla a deteminística:

	0	1	
A	AB		0
AB	AB	BC	0
BC	C	BC	1
C	C		1

El proceso de conversión a determinística garantiza que no hay estados extraños.
El autómata resultante, se observa claramente que no tiene estados equivalentes, por tanto el autómata finito determinístico simplificado es el obtenido en el primer paso.

1c. Convierta a determinístico y simplifique el siguiente autómata finito

	0	1	
A	B		0
B	C,D	B	0
C	B		0
D			1

El primer paso es convertirla a deteminística:

	0	1	
A	B		0
B	CD	B	0
CD	C		1

El proceso de conversión a determinística garantiza que no hay estados extraños.
El autómata resultante, se observa claramente que no tiene estados equivalentes, por tanto el autómata finito determinístico simplificado es el obtenido en el primer paso.

3. Construir autómata finito que reconozca cualquier secuencia de ceros y unos tal que si contiene la subhilera 001 entonces debe contener la subhilera 110.

Construiremos este autómata haciendo uso de las propiedades de la lógica de proposiciones: P implica Q es lo mismo que no P o Q.

Por consiguiente, construiremos tres autómatas finitos: uno que reconozca secuencias de ceros y unos que no contengan la subhilera 001; otro que reconozca secuencias de ceros y unos que contengan la subhilera 110, y un tercero que sea la unión de los dos primeros. Este tercer autómata finito es el autómata pedido.

3.1 El autómata que reconoce secuencias de ceros y unos que no contengan la subhilera 001 es:

Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía o que termina en 1;
B: hilera que termina en 10;
C: hilera que termina en 00;
D: hilera que contiene la subhilera 001}
Estado inicial = A
Estados de aceptación = {A, B, C}
Tabla de transiciones:

	0	1	
A	B	A	1
B	C	A	1
C	C	D	1
D	D	D	0

3.2 El autómata finito que reconoce secuencias de ceros y unos tal que contengan la subhilera 110 es:

Símbolos de entrada = {0,1}
Estados = {E: hilera vacía o que termina en 0;
F: hilera que termina en 01;
G: hilera que termina en 11;
H: hilera que contiene la subhilera 110}
Estado inicial = E
Estado de aceptación: H
Tabla de transiciones:

	0	1	
E	E	F	0
F	E	G	0
G	H	G	0
H	H	H	1

3.3 Ahora, haciendo la unión de dichos autómatas, obtenemos un autómata finito no determinístico que reconoce cualquier secuencia de ceros y unos que cumple las condiciones planteadas. El autómata queda:

		0	1	
→	A	B	A	1
	B	C	A	1
	C	C	D	1
	D	D	D	0
→	E	E	F	0
	F	E	G	0
	G	H	G	0
	H	H	H	1

El cual, al convertirlo a determinístico, simplificarlo y renombrar los estados tenemos que nuestro autómata queda:

	0	1	
A	B	C	1
B	G	C	1
C	B	H	1
D	D	E	0
E	D	F	0
F	H	F	0
G	G	E	1
H	H	H	1

4a. Construya autómata finito no determinístico que reconozca cualquier secuencia de ceros y unos tal que termine en 001.

	0	1	
A	A, B	A	0
B	C		0
C		D	0
D			1

4c. Construya autómata finito no determinístico que reconozca cualquier secuencia de ceros y unos tal que represente un entero decimal impar.

	0	1	
A	A	A, B	0
B		B	1

4e. Construya autómata finito no determinístico que reconozca cualquier secuencia construida con las palabras real, read, ready y return, incluyendo la secuencia nula.

Símbolos de entrada = {R, E, A, L, D, Y, T, N}
Estados: {0: hilera vacía,
R1: entró R de REAL,
E1: entró E de REAL,
A1: entró A de REAL,
L: entró L de REAL,
R2: entró R de READ,
E2: entró E de READ,
A2: entró A de READ,
D1: entró L de READ,
R3: entró R de READY,
E3: entró E de READY,
A3: entró A de READY,
D2: entró D de READY,
Y: entró Y de READY,
R4: entró R de RETURN,
E4: entró E de RETURN,
T: entró T de RETURN,
U: entró U de RETURN,
R5: entró R de RETURN,
N: entró N de RETURN,
err: estado de error}

Estado inicial = 0
Estados de aceptación = {0, L, D1, Y, N}

	R	E	A	L	D	Y	T	U	N	
0	R1, R2, R3, R4									1
R1		E1								0
E1			A1							0
A1				L						0
L	R1, R2, R3, R4									1
R2		E2								0
E2			A2							0
A2					D1					0
D1	R1, R2, R3, R4									1
R3		E3								0
E3			A3							0
A3					D2					0
D2						Y				0
Y	R1, R2, R3, R4									1
R4		E4								0
E4							T			0
T								U		0
U	R5									0
R5									N	0
N	R1, R2, R3, R4									1
err										0

Recuerde que las transiciones en blanco son transiciones hacia el estado de error.

MODULO 3
EXPRESIONES REGULARES

3.1 INTRODUCCION
Hasta aquí hemos venido tratando lo que son los autómatas finitos, tanto su construcción como los elementos que los componen. Hemos dicho que los conjuntos de secuencias reconocidos por los autómatas finitos se denominan conjuntos regulares.

Trataremos en este módulo cómo representar de una manera concisa y precisa dichos conjuntos. Esta forma de representar dichos conjuntos se denominan expresiones regulares.

3.2 OBJETIVOS
1. Conocer las notaciones para representación de conjuntos de secuencias.
2. Construir expresión regular con base en un autómata finito.
3. Construir autómata finito con base en una expresión regular.
4. Conocer las propiedades de las expresiones regulares.

3.3 PREGUNTAS BÁSICAS
15. Qué es una expresión regular.
16. Qué significa el asterisco (*) como superíndice.
17. Qué significa el símbolo más (+) como superíndice.
18. Para qué se utiliza el Lema de Arden.
19. Cuál es el significado de una letra cualquiera como superíndice.
20. Qué significa la letra **r** como superíndice.

3.4 Definición.
Una expresión regular es una expresión que representa un conjunto de secuencias.

3.5 Notaciones para conjuntos de secuencias
Sean los conjuntos

A = {a, b, c}
B = {a, e, i}

1. La unión de dos conjuntos la vamos a denotar con los símbolos + o |, por consiguiente la unión de A con B será:

   A+B == A|B == {a, b, c, e, i}

   En nuestras notas, para denotar la unión de dos conjuntos seguiremos utilizando el símbolo +.

2. La concatenación de dos conjuntos se denota con el símbolo punto (**.**). En muchas ocasiones dicho símbolo se puede omitir, por tanto la concatenación de A con B será:

   A**.**B == AB == {aa, ae, ai, ba, be, bi, ca, ce, ci}

Recuerde que la concatenación de dos conjuntos es el producto cartesiano de ellos.

Consideremos ahora que tenemos el conjunto

$$A = \{0, 1\} = A^1$$

Entonces el conjunto A.A será:

$$A.A = \{00, 01, 10, 11\} = A^2$$

el cual representa todas las secuencias de longitud dos (2) que se pueden construir con los elementos de A.

El conjunto $A.A^2$ será:

$$A.A^2 = \{000, 001, 010, 011, 100, 101, 110, 111\} = A^3$$

El cual representa todas las secuencias de longitud tres (3) que se pueden construir con los elementos del conjunto A.
Procediendo de una manera similar podemos construir los conjuntos A^4, A^5, …, A^n.

Adicionalmente, podemos considerar la secuencia nula como $A^0 = \{\}$.

Luego de efectuar todas estas construcciones, procedemos a hacer la unión de dichos conjuntos:

$$A^* = A^0 + A^1 + A^2 + \ldots + A^n$$

El operador asterisco se conoce como el operador Kleene Star (operador estrella) y se refiere a todas las secuencias que se pueden construir con los elementos de un conjunto dado.

Por consiguiente, si queremos representar cualquier secuencia de ceros y unos que incluya la secuencia nula, una forma de hacerlo es: $(0+1)^*$.

Ahora, si queremos representar cualquier secuencia con los elementos de un conjunto, que no incluya la secuencia nula, lo hacemos utilizando el operador + como superíndice. Por consiguiente, la representación de cualquier secuencia de ceros y unos tal que no incluya la secuencia nula es A^+.

$$A^+ = A^1 + A^2 + A^3 + \ldots + A^n = (0+1)^+$$

Este tipo de representación de conjuntos de secuencias que hemos presentado se denomina **expresiones regulares (ER)**.

3.6 Ejemplos expresiones regulares.

Si **A** es un alfabeto, una **expresión regular** sobre este alfabeto se representa de la siguiente forma:

Φ es una expresión regular que denota el lenguaje vacío (Lenguaje inválido).

λ es una expresión regular que denota la secuencia nula {}.

Si **0** pertenece a **A**, **0** es una expresión regular que denota el lenguaje {0}.

Si **r** y **s** son expresiones regulares pertenecientes a los lenguajes **R** y **S** entonces definimos las siguientes operaciones:

Unión: (**r**+**s**) es una expresión regular que denota el lenguaje **R+S**.

Concatenación: (**rs**) es una expresión regular que denota el lenguaje **R.S**.

Clausura: $\mathbf{r}^*$ es una expresión regular que denota el lenguaje $\mathbf{R}^*$.

Sea **A** = {0, 1}, entonces:

- **00** representa la secuencia 00.
- $\mathbf{01}^*+\mathbf{0}$ representa cualquier secuencia de ceros y unos tal que empieza con 0 y después tienen cero o más unos.
- $(\mathbf{1}+\mathbf{10})^*$ Representa cualquier secuencia de ceros y unos en las que los ceros están siempre precedidos por unos.
- $(\mathbf{0}+\mathbf{1})^*\mathbf{011}$ Representa cualquier secuencia de ceros y unos tal que termina en 011.
- $\mathbf{0}^*\mathbf{1}^*$ Representa el conjunto de secuencias formadas por una sucesión de ceros seguida de una sucesión de unos. Ambas sucesiones pueden ser vacías
- $\mathbf{00}^*\mathbf{11}^*$ Representa el conjunto de secuencias formadas por una sucesión de ceros seguida de una sucesión de unos. Ninguna de las sucesiones puede ser vacía.
- $\mathbf{r}^*\mathbf{r}$ se le denota como $\mathbf{r}^+$. La anterior expresión regular quedaría $\mathbf{0}^+\mathbf{1}^+$
- $\mathbf{1}^*(\mathbf{01}^*\mathbf{01}^*)^*$ Representa cualquier secuencia de ceros y unos con un número par de ceros.
- $(\mathbf{0}+\mathbf{1})^*\mathbf{0110}(\mathbf{0}+\mathbf{1})^*$ Representa cualquier secuencia de ceros y unos tal que contiene la subsecuencia 0110.
- $(\mathbf{000})(\mathbf{1}+\mathbf{10}+\mathbf{100})^*$ Representa cualquier secuencia de ceros y unos tal que comienza con 000 y después no vuelve a aparecer dicha subsecuencia.
- $(\mathbf{0}+\mathbf{1})^*(\mathbf{000}+\mathbf{101})(\mathbf{0}+\mathbf{1})^*$ Representa cualquier secuencia de ceros y unos tal que contiene la subsecuencia 000 o la subsecuencia 101.

Los ejemplos presentados, digamos que son expresiones regulares que se pueden construir de una manera intuitiva. Sin embargo, hay situaciones en las cuales construir una ER de una forma intuitiva es bastante complicado.

3.7 Propiedades expresiones regulares

Las expresiones regulares poseen una cantidad de propiedades que me permiten manipularlas para diferentes fines. Dichas propiedades las presentamos en la tabla 19.

1. $r + \Phi = \Phi + r = r$
2. $r.\Phi = \Phi.r = \Phi$
3. $\Phi^* = \lambda$
4. $r.\lambda = \lambda.r = r$
5. $r + s = s + r$
6. $(r + s) + t = r + (s + t)$
7. $(r.s).t = r.(s.t)$
8. $r.(s + t) = r.s + r.t$
9. $(s + t).r = s.r + t.r$
10. $r + r = r$
11. $r.r^* = r^*.r = r^+$
12. $r.r^* + \lambda = r^*$
13. $(r^* + s^*)^* = (r + s)^* = (r^*.s^*)^*$
14. $(r^*)^* = r^*$
15. $r.(r.r)^*.r = (r.r)^+$
16. $(r.s)^*r = r(s.r)^*$
17. $(r^*.s)^*r^* = (r + s)^*$
18. $(r^*.s)^* = (r + s)^*.s + \lambda$
19. $(r.s^*)^* = r.(r + s)^* + \lambda$
20. $r(s.r)^*s = (r.s)^+$

Tabla 3.1

Hay muchas situaciones en las cuales es más sencillo construir un autómata finito que reconozca cierto conjunto de secuencias, y con base en dicho autómata, construir la expresión regular que representa dicho conjunto.

Para efectuar este proceso utilizamos lo que se conoce como el **Lema de Arden**.

3.8 Lema de Arden

El Lema de Arden dice: si se tiene una expresión de la forma:

$$X = aX + b$$

Entonces $X = a^*b$

Siendo a y b expresiones regulares que no involucran a X.

Para aplicar el Lema de Arden debemos construir primero el conjunto de ecuaciones características correspondientes a un autómata finito.
Luego de construido el conjunto de ecuaciones características, resolvemos este conjunto para el símbolo inicial del autómata finito, aplicando el Lema de Arden y las propiedades de las expresiones regulares.

Ilustremos este método con un ejemplo.

Sea el siguiente autómata finito que reconoce cualquier secuencia de ceros y unos, tal que el número de ceros sea par.

	0	1	
A	B	A	1
B	A	B	0

Tabla 3.2

Las ecuaciones características correspondientes a dicho autómata finito son:
Para el estado A: $A = 0B + 1A + \lambda$ (1)
Para el estado B: $B = 0A + 1B$ (2)

Al reacomodar la ecuación (2)

$$B = 1B + 0A$$

Podemos aplicar el Lema de Arden, obteniendo:

$$B = 1^*0A \quad (3)$$

Reemplazando (3) en (1) y reacomodando los términos obtenemos:

$$A = 1A + 01^*0A + \lambda$$

$$A = (1 + 01^*0)A + \lambda$$

Y aplicando el Lema de Arden, nos queda:

$$A = (1 + 01^*0)^*$$

La cual es una expresión regular que representa cualquier secuencia de ceros y unos tal que el número de ceros sea par.

3.9 Ejemplo de uso del lema de Arden

Consideremos un segundo ejemplo. El autómata finito de la tabla 21 reconoce cualquier secuencia de ceros y unos tal que el número de ceros sea par y el número de unos también, además, acepta la secuencia nula.

	0	1	
A	C	B	1
B	D	A	0
C	A	D	0
D	B	C	0

Tabla 3.3

Comencemos construyendo el conjunto de ecuaciones características.

$A = 0C + 1B + \lambda$ (1)
$B = 0D + 1A$ (2)

C = 0A + 1D (3)
D = 0B + 1C (4)

Comencemos eliminando la incógnita B, reemplazando (2) en (1) y en (4)

A = 0C + 1(0D + 1A) + λ (5)
C = 0A + 1D (6)
D = 0(0D + 1A) + 1C (7)

Agrupando y simplificando obtenemos:

A = 0C + 10D + 11A + λ (8)
C = 0A + 1D (9)
D = 00D + 01A + 1C (10)

Ahora, reemplazando (9) en (8) y (10) obtenemos:

A = 0(0A + 1D) + 10D + 11A + λ (11)
D = 00D + 01A + 1(0A + 1D) (12)

Desarrollando y agrupando para (11) y (12) obtenemos:

A = (00 + 11)A + (01 + 10)D + λ (13)
D = (00 + 11)D + (01 + 10)A (14)

Aplicando el Lema de Arden para la ecuación (14) tenemos:

D = (00 + 11)*(01 + 10)A (15)

Reemplazando (15) en (13) y agrupando tenemos:

A = [00 + 11 + (01 + 10)(00 + 11)*(01 + 10)]A + λ (16)

Y aplicando el Lema de Arden en (16) obtenemos:

A = [00 + 11 + (01 + 10)(00 + 11)*(01 + 10)]*

La cual es una expresión regular que representa cualquier secuencia de ceros y unos tal que el número de ceros sea par y el número de unos también sea par, aceptando la secuencia nula.

3.10 Notación para expresiones no regulares.

Consideremos la siguiente notación:

$$0^m1^n \quad m, n > 0$$

Significa que el lenguaje está conformado por cualquier cantidad de ceros seguida de cualquier cantidad de unos. El hecho de especificar que **m** y **n** son mayores que cero implica que la secuencia mínima es 01.

Si tuviéramos la expresión

$$0^n1^n \quad n > 0$$

significa que el lenguaje está definido como **cualquier cantidad de ceros seguido por exactamente por la misma cantidad de unos**. Este ya no es un lenguaje regular, puesto que secuencias de ese estilo no pueden ser reconocidas por un autómata finito.

En general, cualquier letra, **excepto la r**, como superíndice, indica que su base se repite alguna cantidad de veces.

Si tenemos la expresión:

$$(ab)^p a^k \quad p >= 0, \quad k > 0$$

Significa que la hilera ab puede aparecer 0 ó más veces, seguida de mínimo una a. Ejemplos válidos son:

a; abababaaaaaaaaa; aba; abababababababa.

Veamos ahora qué significa la **r** como superíndice. Si tenemos la hilera

$$w = 011101$$

entonces

$$w^r = 101110$$

es decir, la **r** como superíndice significa que su base debe estar en reversa.

Estas notaciones para conjuntos no regulares y su proceso de reconocimiento lo trataremos en el módulo correspondiente a autómatas de pila.

Veamos ahora el proceso inverso, es decir, teniendo una expresión regular, construir el autómata finito que reconoce el conjunto de secuencias descrito en la expresión regular.

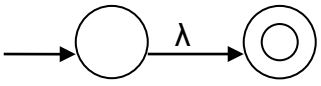
Para ello utilizaremos el método de Thompson.

El método de Thompson se basa en la aplicación de unas construcciones básicas definidas por el mismo Thompson. Dichas construcciones básicas las presentamos a continuación.

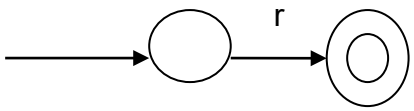
3.11 CONSTRUCCIONES BÁSICAS DE THOMPSON

Sean los símbolos **r** y **s**

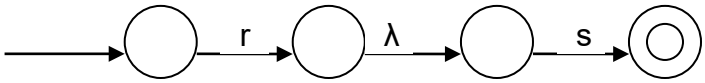
1. Reconoce la secuencia nula



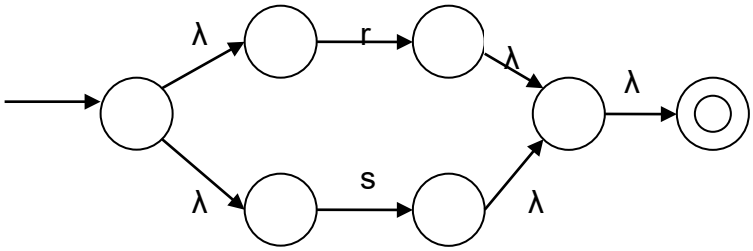
2. Reconoce el símbolo **r**



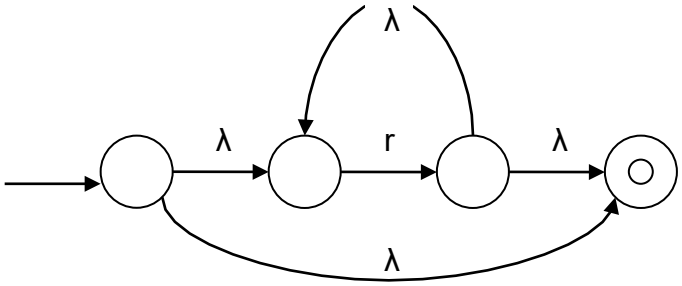
3. Reconoce **(r.s)** siendo **r** y **s** expresiones regulares



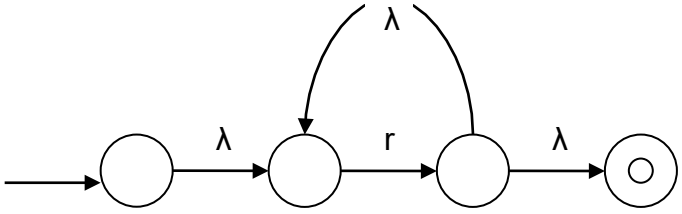
4. Reconoce **(r + s)** ó **(r | s)**



5. Reconoce **r***



6. Reconoce **r+**



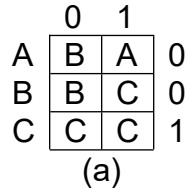
EJERCICIOS PROPUESTOS

1. Construya ER que represente secuencias de ceros y unos con las siguientes características:

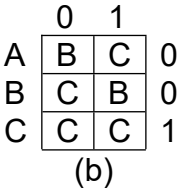
n) Cada pareja de ceros debe estar entre unos.
o) Cualquier secuencia de unos y ceros tal que el número de unos sea par y el número de ceros impar. Para que el número de unos sea par tiene que haber mínimo dos unos.
p) Cada tercer símbolo sea un uno.
q) Haya un número impar de veces de la ocurrencia del patrón 00 admitiendo traslapos.
r) Toda secuencia que contenga exactamente tres unos.
s) Haya un número par de veces de la ocurrencia del patrón 00 sin admitir traslapos.
t) Cada ocurrencia del patrón 11 esté seguida de un cero.

u) Haya al menos un uno.
v) Toda secuencia en la cual cada uno esté precedido y seguido por un cero.
w) Cualquier secuencia que termine en 001.
x) Contenga la pareja 00 ó la pareja 11.
y) Contenga la secuencia 11 y no contenga la secuencia 00.
z) Comience con 00 y termina en 1

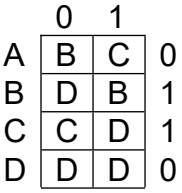
2. Construya ER para cada uno de los siguientes autómata finito



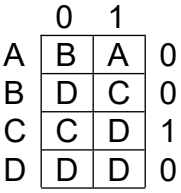
(a)



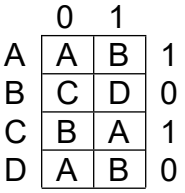
(b)



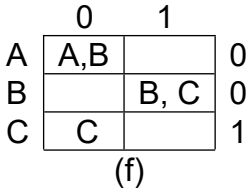
(c)



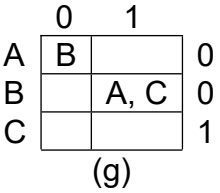
(d)



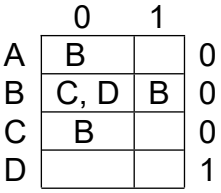
(e)



(f)



(g)

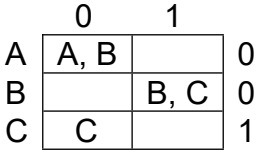


(h)

3. Construya autómata finito simplificado para cada una de las siguientes expresiones regulares

a) $(01)^+$
b) $(0+1)^*$
c) $0(0+1)^*0$
d) $((0+1)(0+1))^+$
e) $(1+01^*0)^*$
f) $(1^*01^*01^*)^*$
g) $(1+0)^*011$
h) $0^*(10^*)^*0^*$
i) $(1^*01^*01^*)^+$
j) $0(0+1)0(0+1+2)^*0$.

4. Dado el siguiente autómata finito



La expresión regular que describe el lenguaje reconocido es:
a) 00110
b) $0^*1^*0^*$
c) $0^+1^+0^*$
d) $0^*1^+0^*$
e) $0^+1^*0^*$

5. La expresión regular $(1*01*01*)*0(1*01*01*)*$ describe el siguiente lenguaje
   a) Cualquier secuencia de unos y ceros.
   b) Cualquier secuencia de unos y ceros con el número de ceros par.
   c) Cualquier secuencia de unos y ceros con el número de ceros impar.
   d) Cualquier secuencia de unos y ceros con el número de unos par.
   e) Cualquier secuencia de unos y ceros con el número de unos impar.

6. El Lema de Arden se utiliza para:
   a) Simplificar un autómata finito.
   b) Resolver una ecuación característica.
   c) Convertir un autómata finito no determinístico a determinístico.
   d) Simplificar expresiones regulares.
   e) Construir expresiones regulares.

7. La expresión $(a^n b^p)^m c^q$ con n >= 0, m > 0, p >= 0 y q > 0 es equivalente a:
   a) $(a*b^+)^+c*$
   b) $(a*b*)*c^+$
   c) $(a*b*)^+c^+$
   d) $(a*b^+)*c^+$
   e) $(a^+b^+)^+c^+$

8. Sea S = {ab, aa, baa}. Cuáles de las siguientes hileras pertenecen a S*:

   a) abaabaaabaa
   b) aaaabaaaa
   c) baaaaabaaaab
   d) baaaaabaa

9. El siguiente autómata finito

	a	b	
1	2	1	0
2	2	3	1
3	3	3	0

   Reconoce secuencias de la forma:
   a) $b^m a^n$ con $m \geq 0$, $n > 0$
   b) $b^m a^n$ con $m > 0$, $n \geq 0$
   c) $b^m a^n$ con $m > 0$, $n > 0$
   d) $b^m a^n$ con $m \geq 0$, $n \geq 0$

10. La expresión $(a+b)^*$ significa:
   a) La suma de a con b elevada a una potencia cualquiera.
   b) Cualquier cantidad de aes seguida de la misma cantidad de bees.
   c) Cualquier secuencia de aes y bees sin incluir la secuencia nula.
   d) Ninguna de las anteriores.

11. La expresión $a^m b^n$ con m > 0 y n ≥ 0 es equivalente a:
   a) a^*b^+
   b) a^*b^*
   c) a^+b^+
   d) a^+b^*

SOLUCIONES

1a. Construya expresión regular que reconozca cualquier secuencia de ceros y unos tal que cada pareja de ceros se halle entre unos.

Construyamos primero el autómata finito que reconoce este tipo de secuencias:
Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía;
B: entró un cero de primero;
C: entró un uno de primero o un uno después del primer cero o después de una pareja de ceros;
D: entró el primer cero después de un uno;
E: entró un cero después de otro cero;
err: entraron dos ceros de primeros o tres ceros seguidos}
Estado inicial: A
Estados de aceptación = {A, B, C}
Tabla de transiciones:

	0	1	
A	B	C	1
B	err	C	1
C	D	C	1
D	E	C	0
E	err	C	0
err	err	Err	0

Con base en el autómata finito construimos el conjunto de ecuaciones características.

$A = 0B + 1C + \lambda$ (1)
$B = 1C + \lambda$ (2)
$C = 0D + 1C + \lambda$ (3)
$D = 0E + 1C$ (4)
$E = 1C$ (5)

Reemplazando (5) en (4) obtenemos: $D = 01C + 1C$

$D = (01 + 1)C$ (6)

Reemplazando (6) en (3) tenemos: $C = 0(01 + 1)C + 1C + \lambda$
La cual, despejando parenthesis y agrupando C obtenemos:

$C = (001 + 01 + 1)C + \lambda$

Y aplicando el Lema de Arden nos queda:

$C = (001 + 01 + 1)^*$ (7)

Reemplazando (7) en (2)

$B = 1(001 + 01 + 1)^* + \lambda$ (8)

Reemplazando (7) y (8) en (1) y simplificando, obtenemos:

$A = (01 + 1)(001 + 01 + 1)^* + 1 + \lambda$

1c. Construya expresión regular que reconozca cualquier secuencia de ceros y unos tal que cada tercer símbolo sea un uno.

Construyamos primero el autómata finito que reconoce este tipo de secuencias:
Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía o el tercer símbolo es un uno;
B: entró el primer símbolo, sea un cero o un uno;
C: entró el segundo símbolo, sea un cero o un uno;
err: el tercer símbolo es un cero}
Estado inicial: A
Estados de aceptación = {A, B, C}
Tabla de transiciones:

	0	1	
A	B	B	1
B	C	C	1
C	err	A	1
err	err	err	0

Con base en el autómata finito construimos el conjunto de ecuaciones características.

$A = (0 + 1)B + \lambda$ (1)
$B = (0 + 1)C + \lambda$ (2)
$C = 1A + \lambda$ (3)

Reemplazando (3) en (2) obtenemos:

$B = (0 + 1)(1A + \lambda) + \lambda$

La cual, al despejar paréntesis nos queda:

$B = 01A + 11A + 0 + 1 + \lambda$ (4)

Reemplazando (4) en (1) obtenemos:

$A = (0 + 1)(01A + 11A + 0 + 1 + \lambda) + \lambda$

La cual, al despejar paréntesis y agrupar los términos con A, obtenemos:

$A = (001 + 011 + 101 + 111)A + (0 + 1 + 00 + 01 + 10 + 11 + \lambda)$

Aplicando el Lema de Arden tenemos:

$$\mathbf{A = (001 + 011 + 101 + 111)^*(0 + 1 + 00 + 01 + 10 + 11 + \lambda)}$$

1e. Construir expresión regular que represente cualquier secuencia de ceros y unos que tenga exactamente tres unos.

Considero que esta expresión regular se puede construir de una manera intuitiva:

$$\mathbf{0^*10^*10^*10^*}$$

1g. Construya expresión regular que represente cualquier secuencia de ceros y unos tal que cada ocurrencia del patrón 11 esté seguida de un cero.

Construyamos primero el autómata finito que reconoce este tipo de secuencias:
Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía o que cumple la condición;
B: entró primer uno para pareja de unos;
C: entró segundo uno consecutivo;
err: entraron tres unos seguidos}
Estado inicial: A
Estados de aceptación = {A, B}
Tabla de transiciones:

	0	1	
A	A	B	1
B	A	C	1
C	A	err	0
err	err	err	0

Con base en el autómata finito construimos el conjunto de ecuaciones características.

$A = 0A + 1B + \lambda$ (1)
$B = 0A + 1C + \lambda$ (2)
$C = 0A$ (3)

Reemplazando (3) en (2)

$B = 0A + 10A + \lambda$ (4)

Reemplazando (4) en (1)

$A = 0A + 1(0A + 1A + \lambda) + \lambda$

Despejando paréntesis y agrupando tenemos:

$A = (0 + 10 + 110)A + (1 + \lambda)$

Aplicando el Lema de Arden

$$\mathbf{A = (0 + 10 + 110)^*(1 + \lambda)}$$

1i. Construir expresión regular que reconozca secuencias de ceros y unos tal que cada uno esté precedido y seguido de un cero.

Construyamos primero el autómata finito que reconoce este tipo de secuencias:
Símbolos de entrada = {0, 1}
Estados = {A: hilera vacía;
B: el último símbolo es un cero;
C: entró un uno;
err: entraron dos unos seguidos o el primer símbolo es un uno}
Estado inicial: A
Estados de aceptación = {A, B}

Tabla de transiciones:

	0	1	
A	B		1
B	B	C	1
C	B		0
err	err	err	0

Con base en el autómata finito construimos el conjunto de ecuaciones características.

$A = 0B + \lambda$ (1)
$B = 0B + 1C + \lambda$ (2)
$C = 0B$ (3)

Reemplazando (3) en (2)

$B = 0B + 10B + \lambda$

Aplicando el Lema de Arden

$B = (0 + 10)^*$ (4)

Reemplazando (4) en (1) nos queda

$$\mathbf{A = 0(0 + 10)^* + \lambda}$$

1k. Construir expresión regular que represente secuencias de ceros y unos tal que contenga la pareja 00 o la pareja 11.

$$\mathbf{(0 + 1)^*(00 + 11)(0 + 1)^*}$$

1m. Construir expresión regular que represente secuencias de ceros y unos que comiencen con 00 y terminen en 1.

$$\mathbf{00(0 + 1)^*1}$$

3. Hacer manualmente literales a), c), e), g) e i) para que los tecleen en ude@.

5. La expresión regular **(1*01*01*)*0(1*01*01*)*** describe el siguiente lenguaje

a) Cualquier secuencia de unos y ceros.
b) Cualquier secuencia de unos y ceros con el número de ceros par.
c) Cualquier secuencia de unos y ceros con el número de ceros impar.
d) Cualquier secuencia de unos y ceros con el número de unos par.
e) Cualquier secuencia de unos y ceros con el número de unos impar.

La opción correcta es la **c)**.

La primera parte de la expresión, al aplicarse una vez, genera cualquier cantidad de unos y una pareja de ceros. Cualquiera que sea el número de veces que se aplique, el número de ceros que genera es par. La última parte de la expresión es idéntica a la primera parte, por consiguiente el número de ceros que se generan es par. El cero de la mitad hará que el total de ceros generados sea impar.

7. La expresión $(a^n b^p)^m c^q$ con n >= 0, m > 0, p >= 0 y q > 0 es equivalente a:

a) $(a*b^+)^+c*$
b) $(a*b*)*c^+$
c) $(a*b*)^+c^+$
d) $(a*b^+)*c^+$
e) $(a^+b^+)^+c^+$

La opción correcta es la **c)**.

Como n >= 0, significa que la **a** debe estar afectada por el operador *****, por tanto, se descarta la opción **e)**. Como m > 0, significa que el término entre paréntesis debe estar afectado por el operador **+**, por consiguiente se descartan las opciones **b)** y **d)**. Como p >= 0, significa que la **b** debe estar afectada por el operador *, por tanto se descarta la opción **a)**. Sólo queda la opción **c)**, la cual cumple la condición q > 0, ya que la **c** está afectada por el operador **+**.

9. El siguiente autómata finito

	a	b	
1	2	1	0
2	2	3	1
3	3	3	0

Reconoce secuencias de la forma:

a) $b^m a^n$ con $m \geq 0$, $n > 0$
b) $b^m a^n$ con $m > 0$, $n \geq 0$
c) $b^m a^n$ con $m > 0$, $n > 0$
d) $b^m a^n$ con $m \geq 0$, $n \geq 0$

La opción correcta es la **a)**.

Cuando el autómata se halla en el estado 1 acepta una **a** o una **b**: Si entra una **b** permanece en el estado 1; si entra una **a** hace transición hacia el estado 2; puede suceder que no llegue ninguna **b**, sino que el primer símbolo sea una **a**, por tanto, el número de bees puede ser cero o más, lo cual implica que el exponente de la **b** (la m) es mayor o igual que cero (**m >= 0**). Esto significa que las opciones **b)** y **c)** se descartan. El estado de aceptación es el estado 2. El autómata llega al estado 2 cuando entre una **a**, y permanece en el estado mientras sigan llegando aes. El autómata se sale del estado 2 cuando llegue una **b**, la cual ocasiona que vaya al estado 3: el estado 3 es un estado de rechazo y cualquiera que sea el símbolo que entre, el autómata permanecerá en ese estado, lo que significa que el estado 3 es un estado de error. Como vimos,

para llegar al estado 2 debe entrar una **a**, lo cual implica que el número de aes debe ser mayor que cero, es decir **n > 0**.

11. La expresión $a^m b^n$ con m > 0 y n ≥ 0 es equivalente a:

a) $a^* b^+$
b) $a^* b^*$
c) $a^+ b^+$
d) $a^+ b^*$

La opción correcta es la **d)**.

Como m es mayor que cero, el símbolo a debe estar afectado por el operador **+**, por consiguiente, se descartan las opciones **a** y **b**; el exponente **n** está definido mayor o igual que cero, por tanto se descarta la opción **c**.

MODULO 4
AUTOMATAS DE PILA

4.1 INTRODUCCIÓN

Los autómatas que hemos venido tratando hasta ahora, es decir, los autómatas finitos, digamos que no tienen memoria. Cuando digo que no tienen memoria me refiero a que no recuerdan cuáles símbolos han procesado. Simplemente saben que están en un estado y no más. En los procesos de reconocimiento, es necesario en muchas situaciones, conocer cuáles símbolos se han procesado con anterioridad. Para ello, se requiere de un mecanismo más potente, con el cual se pueda recordar cuáles símbolos se han procesado. Este mecanismo se conoce como autómata de pila, que como su nombre lo dice utiliza una pila para el procesamiento.

4.2 OBJETIVOS

1. Conocer este modelo matemático, los elementos que lo componen, su funcionamiento y su representación.
2. Construir autómatas de pila.
3. Evaluar autómatas de pila.
4. Hacer traducciones con autómatas de pila.
5. Identificar cuándo usar un autómata de pila.

4.3 PREGUNTAS BÁSICAS

21. Qué es un autómata de pila.
22. En cuáles situaciones se debe usar un autómata de pila.
23. Cómo se representa un autómata de pila.
24. Cómo se definen los estados de un autómata de pila.
25. Cómo puedo reducir estados en un autómata de pila.
26. Cuáles son las operaciones de una transición.
27. Cuándo una hilera es reconocida por un autómata de pila.
28. Cómo se hacen traducciones con un autómata de pila.

4.4 Definición

Un autómata de pila (AP) se define como:

1. Un conjunto finito de símbolos de entrada, incluyendo el símbolo de fin de secuencia (┤).

2. Un conjunto finito de símbolos en la pila, incluyendo un símbolo que indique que la pila está vacía (▼).

3. Un conjunto finito de estados.

4. Un estado designado como estado inicial.

5. Una configuración inicial de la pila.

6. Un conjunto de transiciones.

   Cada transición consta de:

1. Una operación en la pila.
2. Una operación de estado.
3. Una operación de entrada.

La operación en la pila puede ser una de las siguientes: apilar, desapilar o ninguna.
La operación para apilar sólo apila un elemento.
La operación para desapilar sólo elimina un elemento, el que se halla en el tope de la pila. Dichas operaciones son excluyentes, es decir, sólo se puede efectuar una de ellas o ninguna.

La operación de estado consiste en cambiar de estado o permanecer en el estado actual.

La operación de entrada consiste en leer el siguiente símbolo o conservar el símbolo actual. De aquí en adelante, leer el siguiente símbolo lo denominaremos **avance**, y conservar el símbolo actual lo llamaremos **retenga**.

4.5 Primer ejemplo

Consideremos un primer ejemplo de utilización de autómata de pila.

Construir AP que reconozca que los paréntesis estén correctamente igualados en una expresión, es decir, que a cada abre paréntesis le corresponda un cierre paréntesis.

Sea una expresión que tiene la siguiente secuencia de paréntesis:

(( ) ( )) ┤

Nuestro autómata de pila funcionará así:

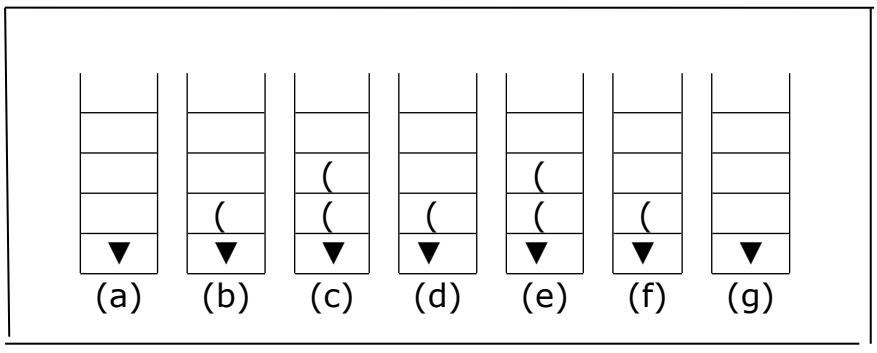


Figura 4.1

La configuración inicial de la pila es pila vacía, tal como se muestra en el caso (a) de la figura 1.

Al leer el primer "abre paréntesis" simplemente lo apilamos y la pila queda como en el caso (b) de la figura 1. Al leer el segundo "abre paréntesis" también lo apilamos y la pila queda como en la caso (c) de la figura 1. Cuando se lee el primer "cierre paréntesis" procedemos a desapilar el "abre paréntesis" que hay en el tope de la pila obteniendo el caso (d) de la figura 1. Luego se lee el siguiente "abre paréntesis" y lo apilamos nuevamente, quedando la pila como en el caso (e) de la figura 1. Nuevamente se lee otro "cierre paréntesis", el cual, hace que se desapile el "abre

paréntesis" que se halla en el tope de la pila obteniendo el caso (f) de la figura 1. Se lee nuevamente otro "cierre paréntesis", por consiguiente se desapila el "abre paréntesis que hay en el tope de la pila y obtenemos el caso (g) de la figura 1. El siguiente símbolo que se lee es el símbolo de fin de secuencia (┤), y como en el tope de la pila se halla el símbolo de pila vacía procedemos a aceptar la secuencia de entrada.

Veamos cómo es el proceso para construir un AP que reconozca la concordancia de paréntesis:

1. Símbolos de entrada = {(,), ┤}
2. Símbolos en la pila = {(, ▼}
3. Estados = $\{S_0\}$
4. Estado inicial = S_0
5. Configuración inicial de la pila = ▼
6. Transiciones:

	(	)	┤
(	#1	#2	R
▼	#1	R	A

S_0

#1: Apile("("), permanezca en el estado S_0, avance.
#2: Desapile, permanezca en el estado S_0, avance.
R: Rechace.
A: Acepte.

Por cada estado se tendrá una tabla de transiciones, cada una de las cuales es una matriz de **m** filas y **n** columnas, siendo **m** el número de símbolos en la pila y **n** el número de símbolos de entrada.

En nuestro ejemplo, sólo tenemos un estado, por consiguiente sólo se tiene una tabla de transiciones.

En nuestro ejemplo, tenemos dos transiciones con "Rechace". Es importante en estas transiciones aprender a describir con palabras la situación de error que se presenta, con el fin de que el usuario detecte y corrija más fácilmente el error que ocurrió. Por ejemplo, en la intersección de "(" en el tope de la pila con el símbolo de entrada "┤" un mensaje de error apropiado sería: "hay paréntesis izquierdos sin su correspondiente paréntesis derecho", y en la intersección de ▼ con ")" un mensaje de error apropiado sería: "hay más paréntesis derechos que izquierdos".

4.6 Segundo ejemplo

Consideremos un segundo ejemplo: Construir AP que reconozca secuencias de la forma

$$0^n1^n \quad \mathbf{n} > 0.$$

Es decir, cualquier cantidad de ceros seguida exactamente por la misma cantidad de unos, siendo la secuencia mínima 01, ya que estamos definiendo n > 0. Ejemplos de secuencias válidas son: 01; 000111; 00000001111111; 0011.

Consideremos cómo sería el proceso de reconocimiento para la secuencia **000111**┤, utilizando un autómata de pila:

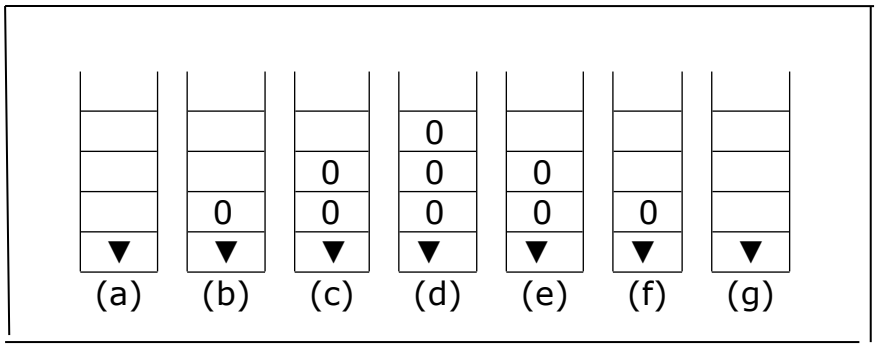


Figura 4.2

Inicialmente la pila está vacía como en el caso (a) de la figura 2.

Al leer el primer cero, simplemente lo apilamos y la pila queda como en el caso (b) de la figura 2.

Al leer el segundo cero, también lo apilamos y la pila queda como en el caso (c) de la figura 2.

Al leer el tercer cero, también lo apilamos y la pila queda como en el caso (d) de la figura 2.

Cuando se lea el primer uno desapilamos el cero que se halla en el tope de la pila y la pila queda como en el caso (e) de la figura 2.

Al leer el siguiente uno desapilamos el cero que se encuentra en el tope de la pila y la pila queda como en el caso (f) de la figura 2.

Al leer el tercer cero desapilamos nuevamente el cero que se halla en el tope de la pila y la pila queda como en el caso (g) de la figura 2.

Por último, se lee el símbolo de fin de secuencia (┤) y como en el tope de la pila se halla el símbolo de pila vacía (▼) procedemos a **aceptar** la hilera de entrada, es decir 000111.

Fijémonos entonces que cuando la pila está vacía, o en el tope de la pila se halla un cero, y el símbolo de entrada es un cero simplemente apilamos el cero y avanzamos en la hilera de entrada, es decir, se lee el siguiente símbolo.

Cuando el símbolo de entrada sea un uno, y en el tope de la pila se halle un cero, simplemente se desapila el cero que se halla en el tope de la pila y avanzamos en la hilera de entrada.

Consideremos la siguiente hilera: **0001110011**┤, la cual es una hilera no válida.

Al procesar los primeros tres ceros, seguidos de los primeros tres unos la pila queda como en el caso (g) de la figura 2, el cual es idéntico al caso (a). El siguiente símbolo de entrada es un cero: si estuviéramos en el caso (a) procederíamos a apilar el cero,

pero como ya hemos reconocido una hilera de la forma 0^n1^n debemos rechazar la hilera de entrada. Esto significa que el caso (g) es diferente del caso (a). Para establecer esta diferencia utilizamos los estados.

Definimos dos estados: uno, el cual será el estado inicial, en el cual se aceptan ceros, hasta que llegue el primer uno. Cuando llegue el primer uno, nos cambiamos hacia un nuevo estado en el cual sólo se aceptan unos como símbolo de entrada y el símbolo de fin de secuencia (┤). Llamemos estos estados los estado S_0 y S_1.

Veamos cómo es el proceso para construir un AP que reconozca este tipo de secuencias:

1. Símbolos de entrada = {0, 1, ┤}
2. Símbolos en la pila = {0, ▼}
3. Estados = $\{S_0, S_1\}$
4. Estado inicial = S_0
5. Configuración inicial de la pila = ▼
6. Transiciones:

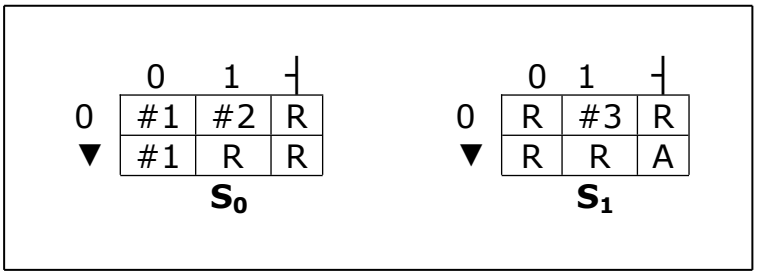


Figura 4.3

#1: Apile("0"), permanezca en el estado **S_0**, avance.
#2: Desapile, cambie al estado **S_1**, avance.
#3: Desapile, permanezca en el estado S_1, avance.
R: Rechace.
A: Acepte.

Es importante tener en cuenta los detalles. En nuestro ejemplo tenemos definido n > 0; si tuviéramos definido n >= 0, significa que se acepta la secuencia nula, por tanto, en el estado **S_0**, en la intersección del símbolo de pila vacía con el de fin de secuencia la transición sería también acepte.

Recuerde que es importante saber describir con palabras cada una de las situaciones de rechazo que se presentan.

4.7 Tercer ejemplo

Consideremos un tercer ejemplo. Construir AP que reconozca secuencias de la forma:

$w2w^r$ con w en (0+1)*

En este caso tendremos también dos estados: un estado S_0 en el cual apilaremos todos los símbolos que lleguen antes del 2; cuando llegue el dos, pasaremos hacia otro estado S_1 en el cual estaremos confrontando que el símbolo en el tope de la pila sea igual al símbolo de entrada.

Nuestro AP es:

Símbolos de entrada = {0, 1, 2, ┤}
Símbolos en la pila = {▼, 0, 1}
Estados = {S_0, S_1}
Estado inicial = S_0
Configuración inicial de la pila = ▼
Transiciones:

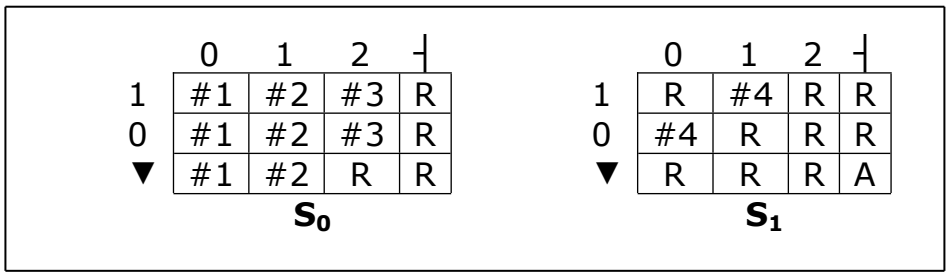


Figura 4.4

#1: apile(0), permanezca en el estado S_0, avance.
#2: apile(1), permanezca en el estado S_0, avance.
#3: cambie al estado S_1, avance.
#4: desapile, permanezca en el estado S_1, avance.

Ojo: no se le olvide tratar de describir con palabras cada una de las situaciones de rechazo.

4.8 Operaciones adicionales en autómatas de pila

1. Operación **out**. Se utiliza para hacer traducciones. La forma general de esta operación es: out(α), siendo α una hilera cualquiera de símbolos. Ejemplo, si tenemos la operación out(xyz), el resultado de ejecutar dicha operación es que escribe xyz.

2. Operación **replace**. Se utiliza para construir AP con menos estados. La forma general de esta operación es: replace(α), siendo α una hilera cualquiera de símbolos. Ejemplo:

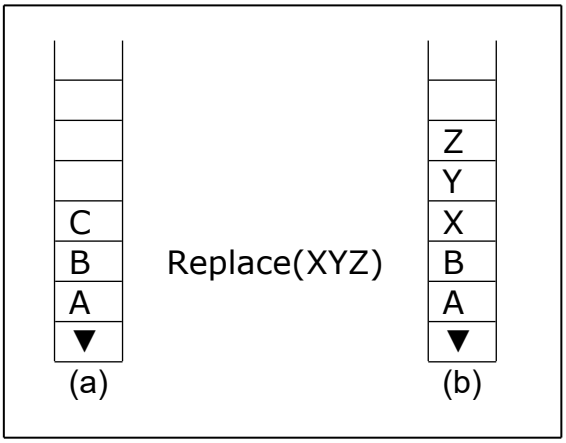


Figura 4.5

Si tenemos una pila como en el caso (a) de la figura 5 y ejecutamos la operación replace(XYZ) la pila quedará como en el caso (b) de la misma figura. Es decir, desapila el símbolo que se halla en el tope de la pila y apila los símbolos de la hilera α en el orden en el cual fueron enviados.

4.9 Ejemplo de utilización de la operación out

Construir AP que acepte como entrada cualquier secuencia de ceros y unos, (0+1)*, y que produzca como salida 0^m1^n siendo m y n el número de ceros y unos, respectivamente, en la hilera de entrada.

Nuestro AP tendrá un solo estado, en el cual apilaremos sólo los unos.

Cuando el símbolo de entrada sea un cero, simplemente se escribe el cero y avanza.

Cuando el símbolo de entrada sea un uno, lo llevamos a la pila y avanzamos.

Cuando el símbolo de entrada sea el símbolo de fin de secuencia (┤) y el símbolo en el tope de la pila sea un uno, desapilamos el uno, escribimos el uno y retenemos el símbolo leído (┤). Este proceso lo repetimos hasta que la pila quede vacía.

Cuando en el tope de la pila se encuentre el símbolo de pila vacía (▼) y el símbolo de entrada es el símbolo de fin de secuencia (┤), simplemente se acepta y nos salimos del AP.

Nuestro AP quedará:

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, 1}
Estados = S_0
Estado inicial = S_0
Configuración inicial de la pila = ▼
Transiciones:

	0	1	┤
1	#1	#2	#3
▼	#1	#2	A

S_0

Figura 4.6

#1: out(0), avance.
#2: Apile(1), avance.
#3: Desapile, out(1), retenga

Observe que hemos omitido la operación correspondiente al estado, ya que nuestro AP es de un solo estado.

Observe también que en la transición denominada #3, utilizamos la operación "retenga", cuyo efecto es que no lee el siguiente símbolo, sino que sigue procesando con el símbolo de fin de secuencia (┤).

4.10 Ejemplo de utilización de la operación replace.

Consideremos ahora un ejemplo de utilización de la operación "replace". Como dijimos anteriormente, la operación replace se usa para construir AP con menos estados.

Construir AP de un estado que reconozca secuencias de la forma 0^n1^n con $n > 0$. (Recuerde que ya hemos construido un AP **con dos estados** que reconoce el mismo conjunto de secuencias).

Ilustremos el proceso con la hilera 000111┤.

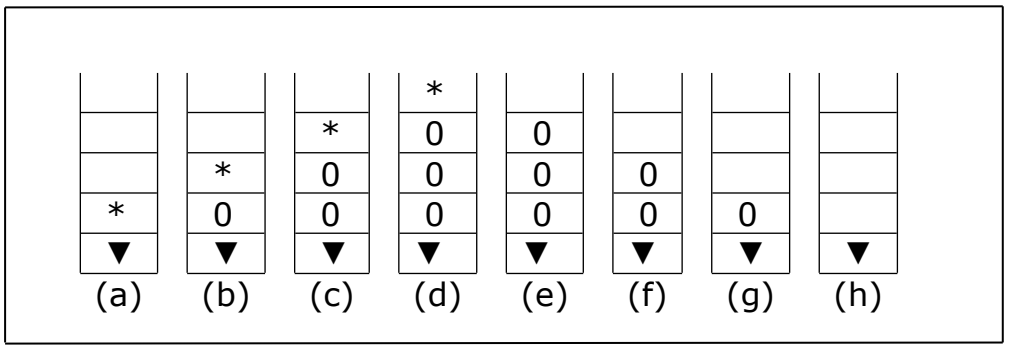


Figura 4.7

Observe que la configuración inicial de la pila es el símbolo de pila vacía (▼) y un asterisco (*), como se muestra en el caso (a) de la figura 7.

Cuando se lea el primer cero, en el tope de la pila hay un asterisco y el símbolo de entrada es un cero, la transición será "**replace(0*)**" y avance, quedando la pila como en el caso (b) de la figura 7.

Al leer el segundo cero, en el tope de la pila tenemos un asterisco y el símbolo de entrada es un cero: la transición será la misma "replace(0*) y avance, quedando la pila como en el caso (c) de la figura 7.

Al leer el tercer cero, nuestra situación es idéntica a las anteriores: en el tope de la pila hay un asterisco y el símbolo de entrada es un cero. La transición es la misma: replace(0*) y avance, y la pila queda como en el caso (d) de la figura 7.

Cuando aparece el primer uno, simplemente desapilamos el símbolo que está en el tope de la pila, es decir, el asterisco y retenemos el símbolo de entrada, es decir, el uno, quedando la pila como en el caso (e) de la figura 7.

Como se retuvo el símbolo de entrada, continuamos el proceso con el mismo uno que tenemos leído, es decir el primer uno.

En este momento, en el tope de la pila hay un cero y el símbolo de entrada es un uno. La transición es desapile y avance, quedando la pila como en el caso (f) de la figura 7.

Al leer y procesar cada uno de los dos unos que faltan, la situación es que en el tope de la pila hay un cero y el símbolo de entrada es un uno, en la cual la transición es desapile y avance quedando la pila como en el caso (h) de la figura 7.

En este punto se tiene leído el símbolo de fin de secuencia (┤) y en el tope de la pila está el símbolo de pila vacía (▼), por lo tanto la transición es acepte.

Nuestro AP de pila es:

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, *, 0}
Estados = S0
Estado inicial = S0
Configuración inicial de la pila = ▼*
Transiciones:

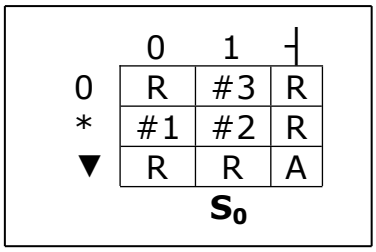


Figura 4.8

#1: Replace(0*), avance.
#2: Desapile, retenga.
#3: Desapile, avance.

En general, el hecho de eliminar estados de un AP, implica manejar más símbolos en la pila.

EJERCICIOS PROPUESTOS

1. Construya AP que reconozca secuencias de la forma
   a) 1^n0^m n > m > 0.
   b) 1^n0^m n >= m > 0.
   c) 1^n0^m m > n > 0.
   d) $1^n0^n1^m0^m$ n, m >= 0.
   e) $1^n0^n1^m0^m$ n, m > 0.
   f) $\{1^n0^n\} + \{0^m1^{2m}\}$ m, n > 0.
   g) (0+1)* tal que el número de ceros sea igual al número de unos.
   h) $b_i2b^r_{i+1}$ siendo b_i la representación en binario del entero i.

2. El conjunto de secuencias reconocido por el siguiente autómata de pila

	0	1	┤
*	Apile(*), Avance	Desapile, Avance	Desapile, Retenga
▼	Apile(*), Avance	Rechace	Acepte

Con configuración inicial: ▼ es:
a) 0^*1^+
b) 0^+1^*
c) 0(0|1)*
d) Ninguno de los anteriores

3. Cuáles de las siguientes secuencias son aceptadas por el siguiente autómata de pila:

	0	1	┤
A	replace(AA), avance	desapile, avance	
▼			Acepte

Con configuración inicial: ▼A es:

a) 1┤
b) 000111111┤
c) 0101011┤
d) 0001111┤

4. Construya autómata de pila que reconozca expresiones lógicas verdaderas. Considere que la expresión lógica está construida con: **T** y **F** para verdadero y falso; **&** para la conjunción y; **|** para la conjunción o, y **!** para la negación.

5. Trate de construir AP de un estado para los ejercicios del punto 1.

6. La operación REPLACE en AP se usa para:
   a) Construir AP con menos estados
   b) Hacer traducciones
   c) Desapilar un símbolo y apilar varios símbolos
   d) Manejar menos símbolos en la pila

SOLUCIONES

1a. Construya autómata de pila que reconozca secuencias de la forma

$$1^n0^m \quad \text{con } n > m > 0$$

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, 1}
Estados = {S_0: Sólo acepta unos hasta que llega el primer cero,
S_1: Sólo acepta ceros hasta que llegue fin de secuencia (┤)}
Estado inicial = S_0
Configuración inicial de la pila = ▼
Tablas de transición:

S_0

	0	1	┤
1	#2	#1	
▼		#1	

S_1

	0	1	┤
1	#3		A
▼			

#1: Apile(1), continúe en S_0, avance.
#2: Desapile, cambie a S_1, avance.
#3: Desapile, continúe en S_1, avance.

1c. Construya autómata de pila que reconozca secuencias de la forma

$$1^n0^m \quad \text{con } m > n > 0$$

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, 1}
Estados = {S_0: Sólo acepta unos hasta que llega el primer cero,
S_1: Sólo acepta ceros hasta que llegue fin de secuencia (┤)}
Estado inicial = S_0
Configuración inicial de la pila = ▼
Tablas de transición:

S_0

	0	1	┤
1	#2	#1	
▼		#1	

S_1

	0	1	┤
1	#3		
▼	#4		A

#1: Apile(1), continúe en S_0, avance.
#2: Apile(1), cambie a S_1, retenga.
#3: Desapile, continúe en S_1, avance.
#4: Permanezca en S_1, avance.

1e. Construya autómata de pila que reconozca secuencias de la forma

$$1^n0^n1^m0^m \quad \text{con } m, n > 0$$

Símbolos de entrada = {0, 1, ┤}

Símbolos en la pila = {▼, 1}
Estados = {S_0: Sólo acepta unos hasta que llega el primer cero de la primera parte de unos y ceros,
S_1: Sólo acepta ceros hasta que llegue el primer uno de la segunda parte de unos y ceros,
S_2: Sólo acepta unos hasta que llegue el primer cero de la segunda parte de unos y ceros,
S_3: Sólo acepta ceros hasta que llegue fin de secuencia (┤)}
Estado inicial = S_0
Configuración inicial de la pila = ▼
Tablas de transición:

	0	1	┤
1	#2	#1	
▼		#1	

S_0

	0	1	┤
1	#2		
▼		#3	

S_1

	0	1	┤
1	#4	#3	
▼			

S_2

	0	1	┤
1	#4		
▼			A

S_3

#1: Apile(1), continúe en S_0, avance.
#2: Desapile, cambie a S_1, avance.
#3: Apile(1), cambie a S_2, avance.
#4: Desapile, cambie a S_3, avance.

1g. Construya autómata de pila que reconozca secuencias de la forma (0+1)* tal que el número de ceros sea igual al número de unos.

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, 0, 1}
Estados = {S_0}
Estado inicial = S_0
Configuración inicial de la pila = ▼
Tabla de transición:

	0	1	┤
1	#3	#2	
0	#1	#3	
▼	#1	#2	A

S_0

#1: Apile(0), continúe en S_0, avance.
#2: Apile(1), continúe en S_0, avance.
#3: Desapile, continúe en S_0, avance.

3. Cuáles de las siguientes secuencias son aceptadas por el siguiente autómata de pila:

	0	1	┤
A	replace(AA), avance	desapile, avance	
▼			Acepte

Con configuración inicial: ▼A

a. 1┤
b. 000111111┤
c. 0101011┤
d. 0001111┤

Las secuencias aceptadas son: **a**, **c** y **d**

5a. Construya autómata de pila de un estado que reconozca secuencias de la forma

$$1^n0^m \quad \text{con } n > m > 0$$

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, *, 1}
Estados = $\{S_0\}$
Estado inicial = S_0
Configuración inicial de la pila = ▼*
Tablas de transición:

	0	1	┤
1	#2		A
*	#2	#1	
▼			

S_0

#1: Replace(1*), avance.
#2: Desapile, avance.

5c. Construya autómata de pila de un estado que reconozca secuencias de la forma

$$1^n0^m \quad \text{con } m > n > 0$$

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, *, ®, 1}
Estados = $\{S_0\}$
Estado inicial = S_0
Configuración inicial de la pila = ▼®
Tablas de transición:

	0	1	┤
1	#3		
*	#3	#2	
®		#1	
▼	#4		A

S_0

#1: Replace(*), retenga.
#2: Replace(1*), avance.
#3: Desapile, avance.
#4: Avance.

5e. Construya autómata de pila de un estado que reconozca secuencias de la forma

$$1^n0^n1^m0^m \quad \text{con } n, m > 0$$

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, *, 1}
Estados = $\{S_0\}$
Estado inicial = S_0
Configuración inicial de la pila = ▼**
Tablas de transición:

S_0

	0	1	┤
1	#3		
*	#2	#1	
▼			A

#1: Replace(1*), avance.
#2: Desapile, retenga.
#3: Desapile, avance.

MODULO 5
GRAMATICAS

5.1 INTRODUCCIÓN
El proceso de reconocimiento de hileras y de lenguajes requiere de un conjunto de normas que definan cómo se construye la hilera o el lenguaje. Presentamos en este módulo un formalismo para definir lenguajes y su sintaxis.

5.2 OBJETIVOS
1. Conocer lo que son las gramáticas, su clasificación y su uso.
2. Construir gramáticas intuitivamente.
3. Construir gramáticas con base en autómatas finitos.
4. Identificar cuándo una gramática es ambigua.

5.3 PREGUNTAS BÁSICAS
29. Qué es una gramática.
30. Para qué se utilizan las gramáticas.
31. Cómo se clasifican las gramáticas.
32. Qué es un terminal.
33. Qué es un no terminal.
34. En qué consiste un proceso de derivación.
35. Qué es una gramática regular.
36. Cómo se identifica una gramática de la forma especial.
37. Qué es una producción.
38. Cuándo una gramática es ambigua.

5.4 Definición
Una gramática es un conjunto de normas que definen la construcción de un lenguaje.

Ejemplo

1. <frase> → <sujeto> <verbo> <predicado>
2. <sujeto> → <artículo> <sustantivo>
3. <artículo> → el
4. <artículo> → la
5. <sustantivo> → niño
6. <sustantivo> → niña
7. <verbo> → quiere
8. <predicado> → <artículo> <sustantivo>

5.5 Conceptos y terminología
En el ejemplo tenemos 8 normas.

Cada una de las normas que se definen en una gramática se denomina **producción**.

Toda producción consta de dos partes: lado izquierdo y lado derecho, y el separador es el símbolo "→", el cual se lee "se define como". Así, considerando la producción número 1 tendremos: <frase> se define como <sujeto>, <verbo> y <predicado>.

Los elementos que están entre ángulos (**<>**) se denominan **No terminales (N)**, y representan las componentes estructurales del lenguaje.

Los elementos que no están entre ángulos (**<>**) se denominan **terminales (T)** y representan los elementos con los que se construye el lenguaje. Si consideramos el idioma español los terminales son todas las palabras que están en el diccionario.

El objetivo de toda gramática es generar hileras finales de terminales.

La obtención de una hilera final de terminales se hace con un proceso llamado **derivación**.

Derivar consiste en reemplazar un no terminal por el lado derecho de alguna producción que defina ese no terminal.

Todo proceso de derivación debe comenzar con el **símbolo inicial de la gramática**.

El **símbolo inicial de la gramática** es el No terminal del lado izquierdo de la primera producción.

Veamos un ejemplo de derivación con la gramática dada.

<frase>┤
↑ producción 1

<sujeto> <verbo> <predicado>┤
↑ producción 2

<artículo> <sustantivo> <verbo> <predicado>┤
↑ producción 3

el <sustantivo> <verbo> <predicado>┤
↑ producción 5

el niño <verbo> <predicado>┤
↑ producción 7

el niño quiere <predicado>┤
↑ producción 8

el niño quiere <artículo> <sustantivo>┤
↑ producción 4

el niño quiere la <sustantivo>┤
↑ producción 6

el niño quiere la niña┤

y hemos obtenido una hilera final de terminales.

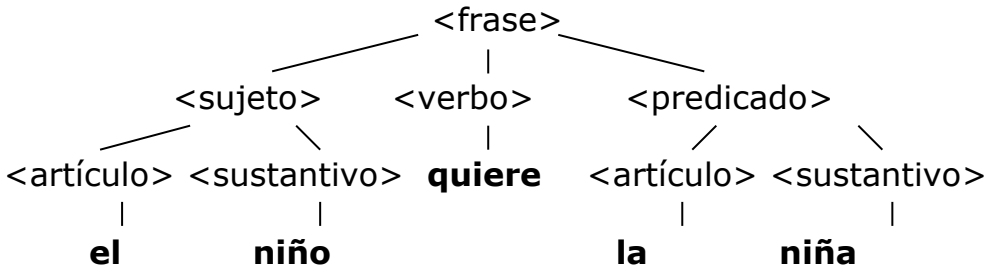
Por consiguiente, la hilera "el niño quiere la niña" pertenece al lenguaje que genera nuestra gramática del ejemplo.

Observe que en nuestro proceso de derivación hemos derivado siempre por el no terminal más a la izquierda que va apareciendo en la hilera generada. Esta forma de derivar se denomina **derivación por la izquierda**.

Si hubiéramos derivado reemplazando siempre por el no terminal más a la derecha que va apareciendo en la hilera generada, se llamaría **derivación por la derecha**.

En general, el usuario puede derivar como quiera, sin embargo es importante tener presente estas formas de derivación, puesto que los procesos de reconocimiento funcionan considerando que se derivó por la izquierda o por la derecha.

A todo proceso de derivación le corresponde un árbol de derivación:



5.6 Clasificación de gramáticas de Chomsky

1. Gramáticas de tipo 0 ó con estructura de frase. Son gramáticas sin restricciones, las cuales incluyen a todas las gramáticas formales. Las producciones son de la forma:

$$\alpha \rightarrow \beta$$

con α en $(N+T)^*$ y β en $(N+T)^+$.

2. Gramáticas de tipo 1 ó gramáticas sensibles al contexto. Son gramáticas cuyas producciones son de la forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

siendo **A** un no terminal, α y β en $(N+T)^*$ y γ en $(N+T)^+$.

Un ejemplo, es una gramática que genera $a^n b^n c^n$ con $n > 0$.

1. <S> → a<S><B>c
2. <S> → a<B><C>
3. b<B> → bb
4. b<C> → bc
5. <C><B> → <B><C>
6. c<C> → cc
7. a<B> → ab

3. Gramáticas de tipo 2 ó gramáticas independientes del contexto. Generan los lenguajes independientes del contexto. Las producciones son de la forma:

$$<N> \rightarrow \alpha$$

con α en (N+T)*. Estos lenguajes pueden ser reconocidos por un autómata de pila.

4. Gramáticas de tipo 3 ó gramáticas regulares. Generan los lenguajes regulares. Estos lenguajes pueden ser reconocidos con autómatas finitos.

Las gramáticas de tipo 3, en nuestro curso, las clasificaremos en gramáticas de la forma especial y gramáticas lineales por la derecha, tal como veremos más adelante.

5.7 Ejemplos de gramáticas

Uno de los principales objetivos de nuestro curso es construir gramáticas.

Hay cierto tipo de lenguajes, para los cuales construir una gramática que los genere, se puede hacer de una manera intuitiva.

Presentemos algunos ejemplos.

5.7.1 Construir una gramática que genere $1^n0^n \quad n > 0$

Presentaremos dos gramáticas que generan este lenguaje:

1. <S> → 1<A>0
2. <A> → 1<A>0
3. <A> → λ

Veamos cómo sería un proceso de derivación:

<S>┤
↑ producción 1
1<A>0┤
↑ producción2
11<A>00┤
↑ producción 3
1100┤

Otra gramática que genera el mismo lenguaje es:

1. <S> → 1<S>0
2. <S> → 10

Es importante que el estudiante capte bien la diferencia entre ambas gramáticas.

5.7.2 Construir gramática que genere $1^n0^n \quad n \geq 0$

1. <S> → 1<S>0
2. <S> → λ

<S>┤
↑ producción 1
1<S>0┤
↑ producción 1
11<S>00┤
↑ producción 2
1100┤

5.7.3 Construir gramática que genere $(0+1)^*$

Presentaremos dos gramáticas para este lenguaje:

1. <S> → 1<S>
2. <S> → 0<S>
3. <S> → λ

Gramática **a**

1. <S> → <S>1
2. <S> → <S>0
3. <S> → λ

Gramática **b**

Ambas gramáticas, la **a** y la **b** generan exactamente el mismo lenguaje. Sin embargo, es importante hacer notar la diferencia entre ellas.

En la gramática **b**, cuando se efectúa algún proceso de derivación, se conoce con cuál símbolo terminará la hilera generada. Si la primera producción que se aplica es la producción 1 se sabe que la hilera generada terminará en 1. Si la primera producción que se aplica es la producción 2 se sabe que la hilera generada terminará en 0.

Esta cualidad de la gramática **b** se puede usar para construir gramáticas que tengan ciertas características.

Por ejemplo, si nos piden construir una gramática que genere cualquier entero positivo par representado en binario, sabemos que los enteros pares representados en binario son aquellos que terminan con 0. Por consiguiente una gramática para ello sería:

1. <S> → <A>0
2. <A> → 1<A>
3. <A> → 0<A>
4. <A> → λ

Observe que el símbolo inicial de la gramática es el no terminal <S> y que sólo hay una producción que define este no terminal, por consiguiente todo proceso de derivación debe comenzar con la producción 1, en la cual, el último símbolo es un cero, lo que garantiza que la hilera generada termina en 0. Es decir, se genera una hilera de ceros y unos que termina en 0, la cual representa un entero par en binario.

Si lo que nos hubieran pedido es construir una gramática que genere cualquier entero positivo impar representado en binario nuestra gramática queda:

1. <S> → <A>1
2. <A> → 1<A>
3. <A> → 0<A>
4. <A> → λ

Observe que basta con cambiar el cero de la producción 1 por un uno y nuestra gramática ya generará cualquier secuencia de ceros y unos que representa un número entero impar representado en binario, porque la hilera generada terminará en uno.

Ahora, si lo solicitado es: construir gramática que genere cualquier entero positivo, múltiplo de cuatro representado en binario nuestra gramática sería:

1. <S> → <A>00
2. <A> → 1<A>
3. <A> → 0<A>
4. <A> → λ

Recuerde que los múltiplos de cuatro representados en binario es cualquier secuencia de ceros y unos que termine en 00.

5.8 Construcción de gramáticas con base en autómatas finitos

Como vimos en el numeral anterior, hay ciertas gramáticas que se construyen en forma intuitiva. En general digamos que la construcción de gramáticas es algo que se encuentra en el estado del arte. Basta con conocer cuál es lenguaje que se desea generar, y con base en las herramientas de las que disponemos (producciones, terminales, no terminales), procedemos a construir una gramática que genere dicho lenguaje.

Sin embargo, hay situaciones en las cuales es más cómodo construir un autómata finito que reconozca dicho lenguaje, y con base en el autómata finito, construir una gramática que genere dicho lenguaje.

Los pasos a seguir para construir una gramática con base en un autómata finito son:

1. Los estados del autómata finito serán los no terminales de la gramática.
2. El estado inicial del autómata finito será el símbolo inicial de la gramática.
3. Los símbolos de entrada al autómata finito serán los terminales de la gramática.
4. Cada transición del autómata finito será una producción: el estado actual del autómata finito será el símbolo del lado izquierdo de la producción; el símbolo de entrada de la transición será el primer símbolo del lado derecho de la producción, y el nuevo estado de la transición será el siguiente símbolo del lado derecho de la producción.
5. Cada estado de aceptación del autómata finito implica una producción cuyo símbolo del lado izquierdo es el estado actual y el lado derecho es la secuencia nula (λ).

Consideremos como ejemplo, construir gramática que genere cualquier secuencia de unos y ceros tal que el número de ceros sea par y el número de unos impar.

Retomemos el autómata finito construido en el módulo 1:

	0	1	
CPUP	CIUP	CPUI	0
CPUI	CIUI	CPUP	1
CIUP	CPUP	CIUI	0
CIUI	CPUI	CIUP	0

Aplicando los pasos definidos tendremos:
Los no terminales serán: <CPUP>, <CPUI>, <CIUP> y <CIUI>.
El símbolo inicial de la gramática será <CPUP>.
Los terminales serán el 0 y el 1.
La gramática será:

1. <CPUP> → 0 <CIUP>
2. <CPUP> → 1 <CPUI>
3. <CPUI> → 0 <CIUI>
4. <CPUI> → 1 <CPUP>
5. <CPUI> → λ
6. <CIUP> → 0 <CPUP>
7. <CIUP> → 1 <CIUI>
8. <CIUI> → 0 <CPUI>
9. <CIUI> → 1 <CIUP>

La única manera de que nuestra gramática genere una hilera final de terminales es cuando se aplique la producción 5. Cuando se aplique esta producción se tiene la certeza de que el número de ceros es par y el número de unos es impar, ya que es equivalente a que el autómata finito se halle en el estado CPUI, el cual representa que el número de ceros es par y el número de unos es impar.

5.9 Gramáticas de la forma especial

Si observamos la gramática obtenida a partir del autómata finito nos podremos dar cuenta que cada producción, en su lado derecho, sólo tiene un terminal seguido de un no terminal o es la secuencia nula.

Una gramática con estas características se denomina gramática de la forma especial.

Definiendo formalmente, gramáticas de la forma especial son gramáticas cuyas producciones son de la forma:

<A> → x <B>
<A> → λ

Siendo <A> y <B> no terminales cualesquiera y x un solo terminal.

Como uno de los objetivos principales del curso es construir reconocedores del lenguaje que genera una gramática, cuando se detecte que la gramática es de la forma especial, el reconocedor del lenguaje que genera la gramática es un autómata

finito, el cual se construye haciendo el proceso inverso al de construir una gramática con base en un autómata finito.

Por consiguiente, dada una gramática de la forma especial, el proceso de construcción del autómata finito que reconoce el lenguaje generado por dicha gramática es:

1. Los no terminales de la gramática serán los estados del autómata finito.
2. El símbolo inicial de la gramática será el estado inicial del autómata finito.
3. Los terminales de la gramática serán los símbolos de entrada al autómata finito.
4. Cada producción será una transición en el autómata finito.
5. Cada producción cuyo lado derecho sea la secuencia nula significa que ese es un estado de aceptación.

5.10 Gramáticas lineales por la derecha

Son gramáticas cuyas producciones son de la forma:

$$<A> \rightarrow \alpha<B>$$
$$<A> \rightarrow \lambda$$

Siendo α una hilera cualquiera de terminales, la cual puede ser la secuencia nula.

Dada una gramática lineal por la derecha, podremos construir un autómata finito que reconoce el lenguaje que genera dicha gramática. Para construir este reconocedor debemos primero convertir la gramática lineal por la derecha a la forma especial.

Para ver cómo es este proceso de conversión consideremos el siguiente ejemplo.

1. $<S> \rightarrow a<A>$
2. $<S> \rightarrow b$
3. $<S> \rightarrow <A>$
4. $<A> \rightarrow abb<S>$
5. $<A> \rightarrow c<A>$
6. $<A> \rightarrow \lambda$

Las producciones 1, 5 y 6 cumplen las características de la forma especial, pero las producciones 2, 3 y 4 no cumplen estas condiciones, sin embargo cumplen las condiciones de ser lineales por la derecha.

A la producción 2, para que sea de la forma especial le falta un no Terminal a continuación del Terminal b.

A la producción 3 le falta un terminal antes del no terminal <A> para que sea de la forma especial.

A la producción 4 le sobran dos terminales para que sea de la forma especial.

Veamos cómo convertir dichas producciones para que cumplan las características de la forma especial.

Comencemos con la producción 2.

Para que la producción 2 sea de la forma especial debe tener un no terminal a continuación del terminal b. Para lograr esto, simplemente nos inventamos un nuevo no terminal y lo colocamos a continuación del terminal b. Llamemos este nuevo no terminal <nulo>.

La producción 2 la podremos reemplazar por estas dos nuevas producciones:

2a. <S> → b<nulo>
2b. <nulo> → λ

A la producción 4, para que sea de la forma especial, le sobran dos terminales. Hagamos lo siguiente: reemplazaremos la producción 4 por estas producciones:

4a. <A> → a<bbS>
4b. <bbS> → b<bS>
4c. <bs> → b<S>

Fíjese que cada una de esas producciones cumple las características de la forma especial y que en conjunto generan exactamente lo mismo que la producción 4. Nos hemos inventado nuevos no terminales en los cuales agrupamos los terminales que aparecen de exceso.

Consideremos ahora la producción 3.

Para que sea de la forma especial debemos colocar un nuevo terminal antes del no terminal <A>. Desafortunadamente no podemos inventar nuevos terminales.
Por consiguiente, lo que haremos será crear nuevas producciones cuyo lado derecho será el lado derecho de las producciones que definen el no terminal del lado derecho de la producción 3. En nuestro ejemplo es el lado derecho de las producciones 4, 5 y 6.

La producción 3 la reemplazaremos por las siguientes producciones:

3a. <S> → a<bbS> // Se obtiene de la transformación anterior
3b. <S> → c<A>
3c. <S> → λ

Al efectuar los reemplazos descritos anteriormente nuestra gramática quedará:

1. <S> → a<A>
2. <S> → b<nulo>
3. <nulo> → λ
4. <S> → a<bbS>
5. <S> → c<A>
6. <S> → λ
7. <A> → a<bbS>
8. <bbS> → b<bS>
9. <bS> → b<S>
10. <A> → c<A>
11. <A> → λ

Esta gramática obtenida genera exactamente el mismo lenguaje que genera la gramática inicial.

El autómata finito que reconoce el lenguaje que genera esta gramática se presenta a continuación:

	A	b	c	
<S>	<A>,<bbS>	<nulo>	<A>	1
<nulo>				1
<A>	<bbS>		<A>	1
<bbS>		<bS>		0
<bS>		<S>		0

El autómata finito obtenido es no determinístico. No hay problema, ya sabemos cómo convertir un autómata finito no determinístico a determinístico.

Las transiciones que quedan en blanco significan que son transiciones hacia el estado de error.

Resumiendo, cuando le presenten una gramática y le soliciten construir el reconocedor del lenguaje que genera esa gramática, debemos fijarnos qué clase de gramática es.

Si la gramática es de la forma especial, el reconocedor del lenguaje que genera esa gramática, es el autómata finito construido con base en la gramática.

Si la gramática es lineal por la derecha, basta con convertir dicha gramática a la forma especial y luego construir el autómata finito correspondiente.

5.11 No terminales muertos

Consideremos ahora que nos solicitan crear una gramática para generar constantes numéricas.

Recuerde que una gramática se puede construir de una manera intuitiva. Basta con conocer exactamente el lenguaje que se desea generar.

Con base en el conocimiento que tenemos de cómo se conforman las constantes construimos la siguiente gramática.

1. <constante> → <signo><numeroDecimal>
2. <constante> → <signo><numeroDecimal>E<entero>
3. <signo> → +|-| λ
4. <entero> → <signo><enteroSinSigno>
5. <entero> → <enteroSinSigno>
6. <enteroSinSigno> → <dígito><enteroSinSigno>
7. <enteroSinSigno> → <dígito>
8. <dígito> → 0|1|2|3|4|5|6|7|8|9
9. <numeroDecimal> → <enteroSinSigno>
10. <numeroDecimal> → <enteroSinSigno>**.**
11. <numeroDecimal> → <enteroSinSigno>**.**<enteroSinSigno>
12. <numeroDecimal> → **.**<enteroSinSigno>

Recordemos también, que en la tabla 3 del módulo 1 tenemos un autómata finito que reconoce constantes numéricas. A partir de dicho autómata finito construimos la siguiente gramática, la cual, también genera constantes numéricas.

1. <0> → d<2>
2. <0> → . <8>
3. <0> → E<9>
4. <0> → s<1>
5. <1> → d<2>
6. <1> → .<8>
7. <1> → E<9>
8. <1> → s<9>
8. <2> → d<2>
9. <2> → .<3>
10. <2> → E<5>
11. <2> → s<9>
12. <2> → λ

.
.
.

41. <9> → d<9>
42. <9> → .<9>
43. <9> → E<9>
44. <9> → s<9>

El símbolo inicial de esta gramática es <0>. Si deseamos hacer un proceso de derivación con esta gramática podremos comenzar con cualquiera de las primeras cuatro producciones.

Si iniciamos el proceso de derivación con la producción 3 tendremos:

<0>
↑ producción 3
E<9>
↑ producción 41
Ed<9>
↑ producción 44
Eds<9>

Y, cualquiera que sea la producción que apliquemos de aquí en adelante, siempre tendremos una hilera en la cual aparece el no terminal <9>.

En otras palabras, nunca obtendremos una hilera final de terminales.

Si una gramática tiene algún no terminal, a partir del cual no es posible obtener una hilera final de terminales, dicho no terminal se denomina no terminal muerto.

No terminales muertos son aquellos no terminales a partir de los cuales no se puede generar una hilera final de terminales.

En una gramática que tenga no terminales muertos se pueden eliminar todas las producciones en las cuales aparezcan dichos no terminales, ya sea en el lado izquierdo o en el lado derecho de la producción.

Nuestro objetivo será entonces: dada una gramática cualquiera, detectar no terminales muertos y eliminar todas las producciones en las cuales aparezcan dichos no terminales.

Para detectar no terminales muertos en una gramática haremos uso de lo que se conoce como la propiedad A.

Propiedad A: Si todos los símbolos del lado derecho de una producción son vivos, entonces el símbolo del lado izquierdo también es vivo.
Los símbolos vivos, por naturaleza son los terminales y el símbolo de secuencia nula.

Para ilustrar el uso de la propiedad A consideremos la siguiente gramática.

1. <S> → a<A><B><S>
2. <S> → b<C><A><C> d
3. <A> → b<A><B>
4. <A> → c<S><A>
5. <A> → c<C><C>
6. <B> → b<A><B>
7. <B> → c<S><B>
8. <C> → c<S>
9. <C> → c

Con base en la propiedad A, construiremos un conjunto en el cual estén los no terminales vivos de la gramática. Dicho conjunto lo inicializamos vacío y hacemos una primera pasada sobre todas las producciones, incluyendo en el conjunto de los no terminales vivos, los no terminales del lado izquierdo de las producciones en las cuales el lado derecho está conformado únicamente por terminales o es la secuencia nula.

En nuestro ejemplo, al efectuar esta primera pasada nuestro conjunto queda:

$$\text{Conjunto de } N_{vivos} = \{<C>\}$$

El no terminal <C> lo incluimos en el conjunto en virtud de la producción 9, ya que es la única producción cuyo lado derecho está conformado sólo por terminales.

Como en esta pasada se incluyó un no terminal en dicho conjunto, implica que debemos volver a hacer el mismo análisis sobre todas las producciones, buscando producciones cuyo lado derecho esté conformado únicamente por terminales y/o el no terminal <C>.

Al efectuar esta segunda pasada nuestro conjunto queda:

$$\text{Conjunto de } N_{vivos} = \{<C>, <A>\}$$

En esta pasada tocó incluir el no terminal <A> en virtud de la producción 5, cuyo lado derecho está conformado por un terminal y dos veces el no Terminal <C>.

Como en esta pasada se incluyó un nuevo no terminal en dicho conjunto, implica que debemos volver a hacer el mismo análisis sobre todas las producciones, buscando producciones cuyo lado derecho esté conformado únicamente por terminales y/o los no terminales <A> y <C>.

Continuamos procediendo de una manera similar hasta efectuar una pasada en la cual no se incluya ningún no terminal en el conjunto de los vivos.

En nuestro ejemplo, el conjunto de no terminales vivos es:

$$\text{Conjunto de } N_{vivos} = \{<C>, <A>, <S>\}$$

Terminado el proceso, nos fijamos cuáles no terminales no quedaron en el conjunto de los vivos. Esos no terminales son los no terminales muertos, y procedemos a eliminar las producciones en las cuales aparezcan dichos no terminales.

En nuestro ejemplo, el no terminal <B> es un no terminal muerto, por consiguiente eliminaremos todas las producciones en las cuales aparezca el no terminal <B>.

Nuestra gramática queda:

1. <S> → b<C><A><C>d
2. <A> → c<S><A>
3. <A> → c<C><C>
4. <c> → c<S>
5. <C> → c

5.12 No terminales inalcanzables

Son no terminales cuyas producciones nunca se aplicarán, cualquiera que sea el proceso de derivación que se efectúe.

Es evidente, que tener en una gramática no terminales cuyas producciones nunca se aplican es una tontería.

Nuestro objetivo será detectar dichos no terminales y eliminar las producciones correspondientes a estos no terminales.

Para detectar no terminales inalcanzables en una gramática haremos uso de lo que se conoce como la propiedad B.

Propiedad B: Si el símbolo del lado izquierdo de una producción es alcanzable, también lo serán todos los símbolos del lado derecho.

El símbolo alcanzable por naturaleza es el símbolo inicial de la gramática, ya que todo proceso de derivación se debe comenzar con dicho símbolo.

Para ilustra el uso de la propiedad B consideremos la siguiente gramática:

1. <S> → a<A><B>
2. <S> → λ
3. <A> → d<D><A>

4. <A> → e
5. <B> → b<E>
6. <B> → f
7. <C> → c<A><B>
8. <C> → d<S><D>
9. <C> → a
10. <D> → e<A>
11. <E> → f<A>
12. <E> → g

Al igual que en el uso de la propiedad A nuestro método consiste en construir una lista con los no terminales alcanzables. No terminal que no aparezca en esta lista es un no terminal inalcanzable.

El conjunto de no terminales alcanzables lo inicializamos con el símbolo inicial de la gramática, en nuestro ejemplo, el no Terminal <S>.

$$\text{Conjunto de } N_{\text{alcanzables}} = \{<S>\}$$

A partir de aquí analizaremos cada producción incluyendo todos los no terminales que aparezcan en el lado derecho de las producciones cuyo símbolo del lado izquierdo sea el no terminal <S>.

En virtud de la producción 1 incluimos los no terminales <A> y <B>.

Nuestro conjunto de alcanzables queda:

$$\text{Conjunto de } N_{\text{alcanzables}} = \{<S>, <A>, <B>\}$$

Luego continuamos analizando todas las producciones cuyo símbolo del lado izquierdo sea el no terminal <A> y el no terminal <B>, incluyendo en el conjunto de no terminales alcanzables los no terminales del lado derecho de cada una de esas producciones.

Al terminar el proceso nuestro conjunto de no terminales alcanzables queda:

$$\text{Conjunto de } N_{\text{alcanzables}} = \{<S>, <A>, <B>, <D>, <E>\}$$

Como se puede observar el no terminal <C> no pertenece a este conjunto, por consiguiente es un no terminal inalcanzable, por consiguiente, podremos eliminar las producciones 7, 8 y 9.

Nuestra gramática queda:

1. <S> → a<A><B>
2. <S> → λ
3. <A> → d<D><A>
4. <A> → e
5. <B> → b<E>
6. <B> → f
7. <D> → e<A>
8. <E> → f<A>

9. <E> → g

En general, si a uno le presentan una gramática y le piden que simplifique dicha gramática, lo que hay que hacer es buscar no terminales muertos e inalcanzables y eliminar las producciones correspondientes, de acuerdo a los procesos vistos anteriormente.

5.13 Gramáticas para expresiones aritméticas

Vamos a considerar ahora una gramática bastante importante en los lenguajes de programación, la gramática para generar expresiones. Dicha gramática es importante porque debe considerar lo concerniente a la prioridad y a la asociatividad de los operadores al evaluar una expresión. Una primera gramática es:

1. <E> → <E>+<E>
2. <E> → <E>-<E>
3. <E> → <E>*<E>
4. <E> → <E>/<E>
5. <E> → <E>%<E>
6. <E> → <E>^<E>
7. <E> → I

Siendo I un símbolo que representa un operando: un identificador o una constante.

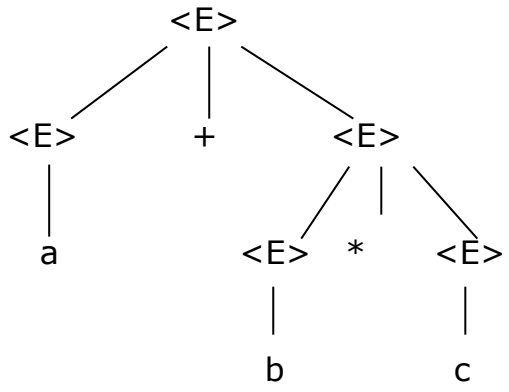
Con base en esta gramática vamos a ver cuál sería el proceso de derivación para obtener la expresión **a+b*c**.

El símbolo inicial de la gramática es el no Terminal <E>. Un proceso de derivación, el cual llamaremos proceso 1, sería:

```
<E>
 ↑ producción 1
<E>+<E>
       ↑ producción 3
<E>+<E>*<E>
↑     ↑    ↑ producción 7
 a   +  b  *  c
```

y hemos obtenido la expresión a+b*c.

Como vimos anteriormente, a todo proceso de derivación le corresponde un árbol de derivación. El árbol de derivación correspondiente a este proceso de derivación es:

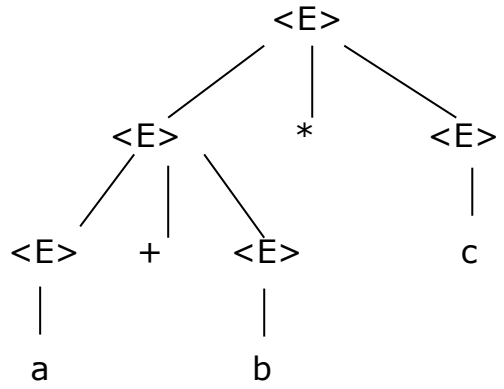


Consideremos ahora otro proceso de derivación, el cual llamaremos proceso 2:

```
<E>
 ↑ producción 3
<E>*<E>
 ↑ producción 1
<E>+<E>*<E>
 ↑     ↑    ↑ producción 7
 a   +  b  *  c
```

y hemos obtenido nuevamente la expresión a+b*c.

El árbol de derivación correspondiente a este nuevo proceso de derivación es:



Fijémonos que los árboles de derivación correspondientes a los procesos de derivación 1 y 2 son topológicamente diferentes.

Cuando esto sucede se dice que la gramática es **ambigua**.

Una gramática es ambigua si es posible obtener una misma hilera final de terminales con procesos de derivación, cuyos árboles de derivación son topológicamente diferentes.

Las gramáticas ambiguas se deben evitar, ya que no se garantiza que el proceso de reconocimiento del lenguaje que generan, funcione apropiadamente.

Determinar si una gramática es ambigua es algo que está dentro de los problemas que se clasifican como indecidibles, es decir, no existe una metodología formal para determinar si la gramática es ambigua o no. Simplemente, si se puede obtener una hilera final de terminales, con procesos de derivación cuyos árboles de derivación son topológicamente diferentes, entonces la gramática es ambigua.

Dado lo anterior, la gramática que hemos escrito, la debemos modificar con varios propósitos: que la gramática no sea ambigua, que considere la prioridad y la asociatividad de los operadores y que tenga en cuenta los paréntesis.

Para lograr estos propósitos debemos hacer algunas transformaciones.

Agrupamos los operadores con la misma prioridad bajo un solo no terminal, ubicando en la gramática de primeros, las producciones correspondientes a los operadores con menor prioridad. En nuestro ejemplo, los operadores de suma y resta, que son los de

menor prioridad los manejaremos con el no terminal <E>; los operadores con la siguiente prioridad, es decir, los operadores de multiplicación, división y módulo los agruparemos bajo el no terminal <T>; el operador de potenciación lo manejaremos con el no terminal <F>, y las expresiones entre paréntesis y los operandos los agruparemos bajo el no terminal <P>.

Nuestra gramática queda:

1. <E> → <E>+<T>
2. <E> → <E>-<T>
3. <E> → <T>
4. <T> → <T>*<F>
5. <T> → <T>/<F>
6. <T> → <T>%<F>
7. <T> → <F>
8. <F> → <F>^<P>
9. <F> → <P>
10. <P> → (<E>)
11. <P> → I

Con esa transformación se suprime la ambigüedad y se maneja la prioridad de los operadores.

La asociatividad de los operadores es: sumas, restas, multiplicaciones, divisiones y módulo son asociativas por la izquierda, es decir, operaciones consecutivas con la misma prioridad se ejecutan de izquierda a derecha en el orden en que aparecen; la potenciación es asociativa por la derecha, es decir operaciones consecutivas de potenciación se ejecutan de derecha a izquierda.

La asociatividad por la izquierda se logra construyendo la producción de tal manera que el símbolo del lado izquierdo sea el mismo con el que comienza el lado derecho. Por consiguiente, en la gramática que hemos construido, hemos definido todas las operaciones asociativas por la izquierda. Pero, como la potenciación debe ser asociativa por la derecha debemos modificar la producción 8.

Para lograr asociatividad por la derecha basta con construir la producción de tal manera que el símbolo del lado izquierdo sea el mismo símbolo con el cual termina el lado derecho de la producción.

Al hacer esta modificación, nuestra gramática queda:

1. <E> → <E>+<T>
2. <E> → <E>-<T>
3. <E> → <T>
4. <T> → <T>*<F>
5. <T> → <T>/<F>
6. <T> → <T>%<F>
7. <T> → <F>
8. <F> → <P>^<F>
9. <F> → <P>
10. <P> → (<E>)
11. <P> → I

Veamos cómo sería el proceso de derivación para obtener la expresión a+b*c

```
<E>
 ↑ producción 1
<E>+<T>
 ↑3    ↑4   producciones 3 y 4
<T>+<T>*<F>
 ↑7    ↑7   ↑9   producciones 7 y 9
<F>+<F>*<P>
 ↑9    ↑9   ↑11   producciones 9 y 11
<P>+<P>*c
 ↑11  ↑11   producción  11
 a+b*c
```

Veamos el proceso para generar la expresión (a+b)*(c+d)

```
<E>
 ↑3   producción 3
<T>
 ↑4  producción  4
<T>*<F>
 ↑7    ↑9  producciones 7 y 9
<F>*<P>
 ↑9    ↑10  producciones 9 y 10
<P>*(<E>)
 ↑10   producción  10
(<E>)*(<E>)
  ↑1     ↑1   producción 1
(<E>+<T>)*(<E>+<T>)
```

Dejamos como ejercicio para el estudiante terminar el proceso de derivación para obtener la expresión pedida.

Otra gramática para expresiones aritméticas

1. <E> → <T><ListaE>
2. <ListaE> → +<T><ListaE>
3. <ListaE> → -<T><ListaE>
4. <ListaE> → λ
5. <T> → <F><Lista T>
6. <ListaT> → *<F><ListaT>
7. <ListaT> → /<F><ListaT>
8. <ListaT> → %<F><ListaT>
9. <ListaT> → λ
10. <F> → <P>^<F>
11. <F> → <P>
12. <P> → (<E>)
13. <P> → I

Con esta nueva gramática veamos cómo sería el proceso de derivación para obtener la expresión a+b*c

<E>
↑1 producción 1
<T><ListaE>
↑5 ↑2 producciones 5 y 2
<F><Lista T>+<T><ListaE>
↑11 ↑9 ↑5 ↑4 producciones 11, 9, 5 y 4
<P>+<F><ListaT>
↑13 ↑11 ↑6 producciones 13, 11 y 6
a+<P>*<F><ListaT>
↑13 ↑11 ↑9 producciones 13, 11 y 9
a+b*<P>
↑13 producción 13
a+b*c

Es importante adquirir destreza en los proceso de derivación para obtener alguna hilera dada. Esto facilitará una mejor comprensión de los procesos de reconocimiento.

EJERCICIOS PROPUESTOS

1. Construya gramática para cada uno de los siguientes lenguajes:
   a) $0^n1^m2^n$ n, m >= 1.
   b) $0^n1^m2^{n+2m}$ n >=0, m >= 1.
   c) $0^n1^{n+m}2^m$ n >= 1, m >= 0.
   d) 0^n1^m n >= m >= 1.
   e) 0^n1^m n > m >= 0.
   f) $0^m1^n2^k$ m > n + k; n, k >= 0.
   g) 0^n1^m m <= n <= 2m; n, m >= 0.
   h) $1^n0^m1^p$ n+p > m >= 0.
   i) 0^n1^m n <> m; n, m > 0.
   j) $1^n0^n + 0^m1^m$ m, n >= 0.
   k) $1^{3n+2}0^n$ n > 0.
   l) waw^r con w en (0+1)*.
   m) 0^n1^m con n par y mayor que cero y m impar.
   n) $(0+1+2)^+$ tal que contiene exactamente dos o tres ceros en cualquier posición.
   o) $(a^* + b)c^+a^*$.
   p) (0+1)* tal que el número de ceros sea igual al número de unos.
   q) (GO, GOTO, TOO, ON)*.
   r) (GO, GOTO, TOO, ON)$^+$ permitiendo traslapos.
   s) $(0+1)^+$ tal que represente enteros múltiplos de tres representados en binario.

2. Convierta la siguiente gramática a la forma especial.

   1. <S> → <A><B>
   2. <A> → <C>
   3. <A> → <D>

4. <B> → c
5. <B> → c<B>
6. <C> → a
7. <C> → a<C>
8. <D> → b
9. <D> → b<D>

3. Un lenguaje tiene definidos los siguientes No Terminales:
<dígito> → 0|1|2|3|4|5|6|7|8|9
<letra> → <minúscula>|<mayúscula>
<minúscula> → a|b|c|d|...|x|y|z
<mayúscula> → A|B|C|D|...|X|Y|Z

Si <vares> es un No Terminal para palabras reservadas y se quiere que éstas se escriban en mayúsculas una producción que cumple este propósito es:

a) <vares> → <letra>|<vares><mayúscula>
b) <vares> → <mayúscula>|<vares><letra>
c) <vares> → <mayúscula><minúscula>|<vares><mayúscula>
d) <vares> → <mayúscula>|<vares><mayúscula>
e) <vares> → <mayúscula>|<vares><minúscula>

4. Una gramática lineal por la derecha es:

a) Una gramática con lado derecho la secuencia nula
b) Una gramática con sólo un NT en el lado derecho y que sea el último.
c) Una gramática con sólo terminales en el lado derecho
d) Todas las anteriores

5. Convierta la siguiente gramática lineal por la derecha a la forma especial y construya el AF que reconoce el lenguaje generado por la gramática.

1. <S> → 01<A>
2. <S> → λ
3. <A> → <B>
4. <A> → 1<S>
5. <A> → 0
6. <B> → 11<A>
7. <B> → λ

6. Construya gramática completa para expresiones en un lenguaje de programación. Su gramática debe incluir los operadores aritméticos (+, -, *, /, %, ^), los operadores relacionales (>, >=, <, <=, ==, !=) y los operadores lógicos (!, &, |). Tenga en cuenta la prioridad y la asociatividad de cada uno de ellos.

7. El lenguaje reconocido por el siguiente AF se denomina L.

	0	1	
A	B	A	0
B	C	B	1
C	B	A	0

Construya gramáticas lineales por la derecha para:

a) L + λ
b) L . L
c) L*
d) L^+

8. Convierta la siguiente gramática a lineal por la derecha

1. <S> → <S>a
2. <S> → <S>b
3. <S> → a

9. Dadas dos gramáticas G_1 y G_2 que generan los lenguajes L_1 y L_2 respectivamente, defina procedimientos generales para construir gramáticas que generen:

a) $L_1 + L_2$
b) $L_1 . L_2$
c) L_1*

10. Construya gramáticas de la forma especial, sin no terminales extraños, para cada Uno de los siguientes autómatas.

	0	1	
A	A, B		0
B		C	1
C	C	B, C	0
D		C, D	1

(a)

	0	1	
A	B	G	0
B	I	F	1
C	A	B	0
D	F	D	0
E	C	H	0
F	H	F	0
G	A	B	1
H	D	F	0
I	D	A	0

(b)

SOLUCIONES

1a. Construya gramática que genere el siguiente lenguaje:

$0^n1^m2^n$ n, m >= 1

1. <S> → 0<A>2
2. <A> → 0<A>2
3. <A> → 1<B>
4. <B> → 1<B>
5. <B> → λ

1c. Construya gramática que genere el siguiente lenguaje

$0^n1^{n+m}2^m$ n >= 1, m >= 0

1. <S> → 0<A>1<B>
2. <A> → 0<A>1
3. <A> → λ
4. <B> → 1<B>2
5. <B> → λ

1e. Construya gramática que genere el siguiente lenguaje

0^n1^m n > m >= 0

1. <S> → 0<A>
2. <A> → 0<A>
3. <A> → 0<A>1
4. <A> → λ

1g. Construya gramática que genere el siguiente lenguaje

0^n1^m m <= n <= 2m; n, m >= 0

1. <S> → 0<S>1
2. <S> → 00<S>1
3. <S> → λ

1i. Construya gramática que genere el siguiente lenguaje

0^n1^m n <> m; n, m > 0

1. <S> → 0<A>1
2. <A> → 0<A>1
3. <A> → 0<B>
4. <A> → <C>1
5. <B> → 0<B>
6. <B> → λ
7. <C> → <C>1
8. <C> → λ

1k. Construya gramática que genere el siguiente lenguaje

$1^{3n+2}0^n$ n >= 0

1. <S> → 11<A>
2. <A> → 111<A>0
3. <A> → λ

1m. Construya gramática que genere el siguiente lenguaje

0^n1^m n, m > 0; con n par y m impar.

1. <S> → 00<A>1
2. <A> → 00<A>
3. <A> → <A>11
4. <A> → λ

1o. Construya gramática que genere el siguiente lenguaje

$(a^* + b)c^+a^*$

1. <S> → a<A><C>
2. <S> → b<C>
3. <S> → <C>
4. <A> → a<A>
5. <A> → λ
6. <C> → c<C>
7. <C> → c<A>

1q. Construya gramática que genere el siguiente lenguaje

(GO, GOTO, TOO, ON)*

1. <S> → GO<S>
2. <S> → GOTO<S>
3. <S> → TOO<S>
4. <S> → ON<S>
5. <S> → λ

1s. Construya gramática que genere múltiplos de tres, mayores que cero, representados en binario.

Esta gramática se construye con base en un autómata finito que reconozca múltiplos de tres mayores que cero representados en binario.

Para construir este autómata recordemos primero dos cositas: si se tiene una secuencia de ceros y unos, la cual representa un entero **x** mayor que cero, y se le agrega un cero a la derecha, el entero que se obtiene es **2x**; si en vez de agregarle un cero a la derecha, se le agrega un uno, el entero que se obtiene es **2x+1**.

Adicionalmente, un entero mayor que cero es múltiplo de tres si es de la forma **3x**.

Si se tiene un entero de la forma **3x**, representado en binario, y se le agrega un cero a la derecha, se obtiene el entero **6x**, el cual es también de la forma **3x** (6x=3*(2x)); pero si en vez de agregarle un cero se le agrega un uno se obtiene el entero **6x+1**, el cual sería de la forma **3x+1**.

Ahora, si se tiene un entero de la forma **3x+1**, representado en binario y se le agrega un cero a la derecha se obtiene el entero **2*(3x+1)**, el cual es un entero de la forma **3x+2**; pero si se le agrega un uno se obtiene el entero **2*(3x+1)+1**, el cual es un entero de la forma **3x** (6x+2+1 = 6x+3 = 3*(2x+1) = 3x).

Ahora, si se tiene un entero de la forma **3x+2**, representado en binario y se le agrega un cero a la derecha, se obtiene el entero **2*(3x+2)**, el cual es un entero de la forma

3x+1; pero si se le agrega un uno se obtiene el entero **2*(3x+2)+1**, el cual es un entero de la forma **3x+2**.

Con base en lo anterior se construye un autómata finito que reconoce cualquier secuencia de ceros y unos tal que represente un entero múltiplo de tres. Dicho autómata es:

	0	1	
3x	3x	3x+1	1
3x+1	3x+2	3x	0
3x+2	3x+1	3x+2	0

Fíjese que los estados los estados se han llamado 3x, 3x+1 y 3x+2, donde el estado de aceptación es el estado 3x, el cual a su vez es el estado inicial, lo que significa que dicho autómata acepta el cero como múltiplo de tres.

Para evitar esto se agrega un nuevo estado, el cual se define como estado inicial y de rechazo. El autómata finito queda:

Símbolos de entrada = {0, 1}
Estados = {S: hilera vacía o de solo ceros;
A: hilera que representa un entero de la forma 3x;
B: hilera que representa un entero de la forma 3x+1;
C: hilera que representa un entero de la forma 3x+2}
Estado inicial = S
Estado de aceptación = A
Tabla de transiciones:

	0	1	
S	S	B	0
A	A	B	1
B	C	A	0
C	B	C	0

Con base en este autómata y la técnica descrita en 5.8, una gramática que genera múltiplos de tres mayores que cero, representados en binario es:

1. <S> → 1<B>
2. <A> → 0<A>
3. <A> → 1<B>
4. <A> → λ
5. <B> → 0<C>
6. <B> → 1<A>
7. <C> → 0<B>
8. <C> → 1<C>

3. Un lenguaje tiene definidos los siguientes No Terminales:

<dígito> → 0|1|2|3|4|5|6|7|8|9
<letra> → <minúscula>|<mayúscula>
<minúscula> → a|b|c|d|...|x|y|z
<mayúscula> → A|B|C|D|...|X|Y|Z

Si <vares> es un No Terminal para palabras reservadas y se quiere que éstas se escriban en mayúsculas una producción que cumple este propósito es:

a) <vares> → <letra>|<vares><mayúscula>
b) <vares> → <mayúscula>|<vares><letra>
c) <vares> → <mayúscula><minúscula>|<vares><mayúscula>
d) <vares> → <mayúscula>|<vares><mayúscula>
e) <vares> → <mayúscula>|<vares><minúscula>

La opción correcta es la **d),** ya que es la única que no involucra el no terminal <minúscula>. Opciones **a)** y **b)** contienen el no terminal <letra>, el cual está definido como <mayúscula> o <minúscula>, y los lados derechos de las opciones **c)** y **e)** contienen el no terminal <minúscula>.

5. Convierta la siguiente gramática lineal por la derecha a la forma especial y construya el AF que reconoce el lenguaje generado por la gramática.

1. <S> → 01<A>
2. <S> → λ
3. <A> → <B>
4. <A> → 1<S>
5. <A> → 0
6. <B> → 11<A>
7. <B> → λ

Las producciones que no cumplen las condiciones de la forma especial son las producciones 1, 3, 5 y 6. Transformemos cada una de ellas para que cumplan las condiciones de la forma especial.

La producción 1 se reemplaza por:

1a. <S> → 0<1A>
1b. <1A> → 1<A>

La producción 5 se reemplaza por:

5a. <A> → 0<nulo>
5b. <nulo> → λ

La producción 6 se reemplaza por:

6. <B> → 1<1A>

La producción 3 se reemplaza por:

3a. <A> → 1<1A>
3b. <A> → λ

Al efectuar los reemplazos la gramática queda:

1. <S> → 0<1A>
2. <1A> → 1<A>
3. <S> → λ
4. <A> → 1<A>
5. <A> → λ
6. <A> → 1<S>
7. <A> → 0<nulo>
8. <nulo> → λ
9. <B> → 1<1A>
10. <B> → λ

Fíjese que al efectuar los reemplazos, el no terminal <B> quedó inalcanzable, por consiguiente, se elimina de la gramática. La gramática queda:

1. <S> → 0<1A>
2. <S> → λ
3. <1A> → 1<A>
4. <A> → 1<A>
5. <A> → λ
6. <A> → 1<S>
7. <A> → 0<nulo>
8. <nulo> → λ

Con base en lo expuesto en 5.9, el autómata finto que reconoce el lenguaje que genera dicha gramática es:

	0	1	
<S>	<1A>		1
<1A>		<A>	0
<A>	<nulo>	<A>, <S>	1
<nulo>			1

7. El lenguaje reconocido por el siguiente AF se denomina L.

	0	1	
A	B	A	0
B	C	B	1
C	B	A	0

Construya gramáticas lineales por la derecha para:

a) L + λ
b) L . L
c) L*
d) L^+

La gramática original, con base en el autómata finito es:

1. <A> → 0<B>
2. <A> → 1<A>
3. <B> → 0<C>

4. <B> → 1<B>
5. <B> → λ
6. <C> → 0<B>
7. <C> → 1<A>

Resolver para el literal **a)** implica que el estado inicial debe ser de aceptación, es decir, que se debería agregar una producción en la cual se defina el no terminal <A> como la secuencia nula (λ), lo cual sería un error ya que alteraría el lenguaje L, porque cuando esté en el estado **C** y el símbolo de entrada sea un uno hace transición hacia el estado **A**, causando que se acepte una hilera de entrada cuyo último símbolo sea un uno.

Por tanto, para resolver para el literal **a)** se debe agregar un nuevo no terminal, llamémoslo **<S>**, el cual se define como el símbolo inicial de la gramática. La gramática queda:

1. <S> → <A>
2. <S> → λ
3. <A> → 0<B>
4. <A> → 1<A>
5. <B> → 0<C>
6. <B> → 1<B>
7. <B> → λ
8. <C> → 0<B>
9. <C> → 1<A>

Resolver para el literal **b)** implica duplicar la gramática con nuevos no terminales y modificar la producción 5 cambiando el lado derecho por el no terminal equivalente al no terminal <A> de la gramática original. La gramática queda:

1. <A> → 0<B>
2. <A> → 1<A>
3. <B> → 0<C>
4. <B> → 1<B>
5. <B> → <D>
6. <C> → 0<B>
7. <C> → 1<A>
8. <D> → 0<E>
9. <D> → 1<D>
10. <E> → 0<F>
11. <E> → 1<E>
12. <E> → λ
13. <F> → 0<E>
14. <F> → 1<D>

Resolver para el literal **c)** basta con modificar la producción 7 de la gramática obtenida en el literal **a)**, reemplazando el **λ** del lado derecho por el no terminal **<S>**. La gramática queda:

1. <S> → <A>
2. <S> → λ
3. <A> → 0<B>

4. <A> → 1<A>
5. <B> → 0<C>
6. <B> → 1<B>
7. <B> → <S>
8. <C> → 0<B>
9. <C> → 1<A>

Resolver para el literal **d)** consistirá simplemente en agregar una producción para el no terminal <B>, en la cual el lado derecho sea el símbolo inicial de la gramática, es decir el no terminal <A>. La gramática queda:

1. <A> → 0<B>
2. <A> → 1<A>
3. <B> → 0<C>
4. <B> → 1<B>
5. <B> → λ
6. <B> → <A>
7. <C> → 0<B>
8. <C> → 1<A>

9. Dadas dos gramáticas G_1 y G_2 que generan los lenguajes L_1 y L_2 respectivamente, defina procedimientos generales para construir gramáticas que generen:

a) $L_1 + L_2$
b) $L_1 . L_2$
c) L_1^*

Sea <A> el símbolo inicial de la gramática G1 y sea <B> el símbolo inicial de la gramática G2.

Resolver para el literal **a)** consiste en definir un nuevo no terminal, llamémoslo <S>, el cual será el símbolo inicial de la gramática y tendrá dos producciones que son:

1. <S> → <A>
2. <S> → <B>

Resolver para el literal **b)** consiste en definir un nuevo no terminal, llamémoslo <S>, el cual será el símbolo inicial de la gramática y tendrá una producción que es:

1. <S> → <A><B>

Resolver para el literal **c)** consiste en definir un nuevo no terminal, llamémoslo <S>, el cual será el símbolo inicial de la gramática y tendrá tres producciones que son:

1. <S> → <A>
2. <S> → <S><S>
3. <S> → λ

MODULO 6
GRAMATICAS DE TRADUCCIÓN Y GRAMATICAS CON ATRIBUTOS

6.1 INTRODUCCIÓN
Ya hemos definido lo que es una gramática. Una gramática se construye con múltiples propósitos, algunos de ellos son efectuar traducciones y evaluar cierto tipo de lenguajes. Nos ocuparemos en este módulo de ver cómo se logran estos propósitos mediante una gramática.

6.2 OBJETIVOS
1. Conocer lo que son las gramáticas de traducción.
2. Construir gramáticas de traducción.
3. Construir gramáticas con atributos.
4. Reconocer atributos sintéticos y heredados.
5. Manejar atributos de una gramática.

6.3 PREGUNTAS BÁSICAS
39. Qué es una gramática de traducción.
40. Qué es una gramática con atributos.
41. Qué es una secuencia de actividades.
42. Qué son símbolos de acción.
43. Cuándo un atributo es sintético.
44. Cuándo un atributo es heredado.

6.4 Presentación.
Comencemos presentando cómo funciona un proceso de traducción, a medida que se va haciendo un reconocimiento. Consideremos la expresión **a+b*c**, la cual, es una expresión en infijo y la queremos convertir a posfijo.

Vamos a procesar la expresión infijo, y a medida que la vamos procesando la vamos traduciendo a posfijo. Leeremos cada uno de los símbolos de la expresión infijo y ejecutaremos alguna acción según sea el símbolo leído. Como bien se sabe, cualquiera que sea la forma en que esté escrita una expresión (infijo, posfijo o prefijo), el orden relativo de los operandos es el mismo. Lo único que cambia de sitio son los operadores.

Con base en esto, cuando se lea un operando procedemos a escribirlo. Los operadores que se lean sólo se escribirá cuando se haya identificado el segundo operando correspondiente al operador.

Nuestro proceso, expresado en lenguaje natural sería:

Lea(a), escriba(a), lea(+), lea(b), escriba(b), lea(*), lea(c), escriba(c), escriba(*), escriba(+).

Fijémonos que cuando se lee el operando **a**, inmediatamente procedemos a escribirlo, cuando leemos el operador **+**, no lo escribimos inmediatamente, sino que esperamos a reconocer el segundo operando correspondiente al símbolo **+**, el cual es el resultado de multiplicar **b*c**, por consiguiente el **+** lo escribimos cuando se haya reconocido **b*c**. Cuando se lee el operando b, inmediatamente lo escribimos. Cuando se lee el operador *****, esperamos para imprimirlo cuando se haya reconocido el segundo operando correspondiente al *****, es decir la **c**.

Adoptaremos como notación, que la operación de lectura será simplemente leer un elemento (token) de la expresión, y la operación de escritura la escribiremos con el símbolo entre llaves {}. La nomenclatura de escribir algo entre llaves la llamaremos **símbolos de acción**.

Dada esta convención, nuestro proceso queda:

a{a}+b{b}*c{c}{*}{+} **(1)**

Dicha secuencia la llamaremos **secuencia de actividades**.

Una **secuencia de actividades** es entonces una sucesión de símbolos de entrada y símbolos de acción.

Si a una secuencia de actividades le suprimimos los símbolos de entrada nos queda la traducción.

Si a la secuencia de actividades (1) le suprimimos los símbolos de entrada nos queda:

{a}{b}{c}{*}{+}

La cual es la expresión posfijo correspondiente a la expresión infijo a+b*c.

6.5 Traducción desde infijo hacia posfijo.

Nuestro objetivo es entonces lograr, con la gramática que genera el lenguaje, efectuar esta traducción.

Tenga en cuenta que la gramática, además de definir un lenguaje, también la utilizaremos para hacer el proceso de reconocimiento del lenguaje generado por esa gramática y efectuar alguna traducción.

Una de nuestras gramáticas para generar expresiones en infijo es:

12. <E> → <E>+<T>
13. <E> → <E>-<T>
14. <E> → <T>
15. <T> → <T>*<F>
16. <T> → <T>/<F>
17. <T> → <T>%<F>
18. <T> → <F>
19. <F> → <P>^<F>
20. <F> → <P>
21. <P> → (<E>)
22. <P> → I

De aquí en adelante trabajaremos esta gramática en muchas situaciones, pero lo haremos de una manera simplificada. Nuestra gramática simplificada será:

1. <E> → <E>+<T>
2. <E> → <T>
3. <T> → <T>*<F>

4. <T> → <P>
5. <P> → (<E>)
6. <P> → I

Nuestro objetivo es que la gramática traduzca la expresión generada en infijo hacia posfijo.

De acuerdo al proceso que hemos planteado, un operando se escribe tan pronto se genere, por tanto en la producción 6, que es la que genera los operandos colocamos un símbolo de acción para que escriba el operando. La producción 6 quedará: **<P> → I{I}**.

Los operadores se escriben cuando se hayan reconocido los operandos correspondientes a ese operador. Por consiguiente, en las producciones 1 y 3, que son las producciones en las cuales se generan los operadores, colocamos los símbolos de acción correspondientes a dichos operadores. Dichos símbolos de acción los colocamos al final del lado derecho de cada producción, ya que en este sitio ya se han generado los dos operandos correspondientes al operador. Las producciones 1 y 3 quedarán:

<E> → <E>+<T>{+}
<T> → <T>*<P>{*}

Nuestra gramática quedará:

1. <E> → <E>+<T>{+}
2. <E> → <T>
3. <T> → <T>*<P>{*}
4. <T> → <P>
5. <P> → (<E>)
6. <P> → I{I}

Consideremos ahora la otra gramática que tenemos para generar expresiones en infijo.

1. <E> → <T><listaE>
2. <listaE> → +<T><listaE>
3. <listaE> → λ
4. <T> → <P><Lista T>
5. <listaT> → *<P><listaT>
6. <listaT> → λ
7. <P> → (<E>)
8. <P> → I

En esta gramática, los operandos correspondientes a una operación se encuentran en diferentes producciones. Los operandos correspondientes a la operación de suma se hallan definidos en las producciones 1 y 2. En la producción 1 se genera el primer operando y en la producción 2 se genera el segundo operando. Como el símbolo de acción para la suma se debe escribir tan pronto se haya generado el segundo operando correspondiente a la suma, el símbolo de acción se debe colocar a continuación del no terminal <T> de la producción número 2.

Para la operación de multiplicación el razonamiento es exactamente el mismo, por consiguiente el símbolo de acción para la operación de multiplicación debe ir a continuación del no terminal <P> del lado derecho de la producción 5.

Nuestra gramática queda:

1. <E> → <T><listaE>
2. <listaE> → +<T>{+}<listaE>
3. <listaE> → λ
4. <T> → <P><Lista T>
5. <listaT> → *<P>{*}<listaT>
6. <listaT> → λ
7. <P> → (<E>)
8. <P> → I{I}

6.6 Ejemplos gramáticas de traducción

Construir gramática de traducción que genere cualquier secuencia de ceros y unos, incluyendo la secuencia nula (0+1)*, y que produzca como salida 0^m1^n, siendo m y n el número de ceros y unos generados, respectivamente.

Una forma de enfrentar estos problemas de traducción es construir primero la gramática generativa y luego agregarle los símbolo de acción en las producciones apropiadas en los sitios apropiados. Veamos: unas gramáticas que generan (0+1)*es:

1. <S> → 0<S>
2. <S> → 1<S>
3. <S> → λ

1. <S> → <S>0
2. <S> → <S>1
3. <S> → λ

Cualquiera que sea la gramática que utilicemos debemos producir como salida primero los ceros y después los unos. Eso significa que el símbolo de acción para escribir el cero lo debemos ubicar en la producción que genera el cero y antes del símbolo generativo, el cual es el no terminal <S>, y la acción para escribir el uno se debe colocar en la producción que genera el uno y después del símbolo generativo, el cual es el no terminal <S>. Por consiguiente, nuestras gramáticas quedarían:

1. <S> → {0}0<S>
2. <S> → 1<S>{1}
3. <S> → λ

1. <S> → {0}<S>0
2. <S> → <S>1{1}
3. <S> → λ

La clave está en ubicar los símbolos de acción en la gramática generativa.

Sin embargo, no todo tipo de gramáticas son apropiadas para efectuar algunas traducciones. Consideremos el siguiente ejemplo. Construir gramática de traducción que genere 0^m1^n con m y n mayores que cero y que produzca como salida:

0^{m-n} si m > n
1^{n-m} si m < n
λ si m == n

es decir, la traducción debe ser escribir el exceso de ceros si hay más ceros que unos, el exceso de unos si hay más unos que ceros o nada si el número de ceros es igual al número de unos.

Una gramática que genere 0^m1^n con m y n > 0 es:

1. <S> → 0<A>1
2. <A> → 0<A>1
3. <A> → 0<A>
4. <A> → <A>1
5. <A> → λ

Como hemos dicho, el objetivo es escribir el exceso de ceros o de unos dependiendo de cuál de los dos se genera más veces. Desafortunadamente, con esa gramática no estamos en capacidad de determinar cuándo se generan más ceros que unos o más unos que ceros, por consiguiente, la traducción planteada no se puede efectuar con esa gramática.

Para lograr hacer la traducción pedida debemos construir una gramática en la cual podamos identificar cuándo se genera exceso de ceros o de unos. Una gramática con la cual podamos hacer esa identificación es:

1. <S> → 0<A>1
2. <A> → 0<A>1
3. <A> → λ
4. <A> → 0<B>
5. <B> → 0<B>
6. <B> → λ
7. <A> → <C>1
8. <C> → <C>1
9. <C> → λ

En las primeras tres producciones se genera igual cantidad de ceros que de unos.
Si se desea generar más ceros que unos aplicamos la producción 4, en la cual se genera el primer cero de más. De ahí en adelante se debe seguir aplicando la producción 5 o la seis. Cada vez que se aplique la producción 5 se genera un cero de más, hasta que se aplique la producción 6 en la cual se detiene el proceso de generación y se obtiene una hilera final de terminales.

Si lo que se desea es generar más unos que ceros se aplica la producción 7 y a partir de ahí sólo podremos seguir generando exceso de unos aplicando la producción 8 o terminar el proceso aplicando la producción 9.

Como podrá observar, con esta gramática detectamos en qué momento se genera el exceso de ceros o de unos, por tanto, podremos efectuar la traducción ubicando los símbolos de acción en las producciones en las cuales se genera el exceso de un símbolo en particular.

Nuestra gramática de traducción queda:

1. <S> → 0<A>1
2. <A> → 0<A>1
3. <A> → λ

4. <A> → {0}0<B>
5. <B> → 0{0}<B>
6. <B> → λ
7. <A> → {1}<C>1
8. <C> → <C>1{1}
9. <C> → λ

En este ejemplo, no importa el sitio en el cual se ubique el símbolo de acción, lo que importa es la producción en la cual se coloque dicho símbolo. Esto es posible porque sólo hay un símbolo para escribir.

6.7 Gramáticas con atributos

Hasta ahora hemos visto gramáticas generativas (que generan un lenguaje) y gramáticas de traducción (convierten una entrada en otra cosa). Vamos a considerar ahora otro de los usos que se le da a una gramática.

Los símbolos que conforman una gramática, terminales y no terminales, pueden tener asociados datos, los cuales se utilizan para algún propósito. Dichos datos se denominan atributos.

6.8 Atributos sintéticos

Consideraremos, como ejemplo, nuestra gramática para generar expresiones infijo.

Cuando manipulamos expresiones, es de interés evaluar la expresión.

Si tenemos la producción <E> → <E>+<T>, estamos definiendo la sintaxis para la operación de suma, además, nos interesará poder efectuar la suma.

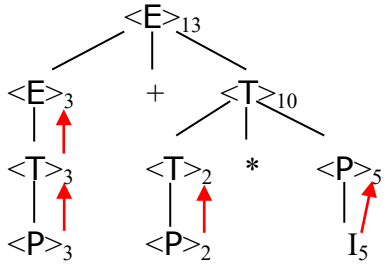
Para poder efectuar la suma los no terminales <E> y <T> del lado derecho de la producción deben tener asociados algún valor. Sumamos estos valores y el resultado lo almacenaremos en el valor asociado al no terminal <E> del lado izquierdo de la producción.

Simbólicamente, nuestra producción queda:

$$<E>_p \rightarrow <E>_q + <T>_r$$
$$p = q + r$$

Es evidente, que para poder efectuar dicha operación en esa producción, en los campos de datos asociados a los no terminales <E> y <T> del lado derecho de la producción deben estar almacenados los valores a sumar.

Para mostrar cómo llegan dichos valores a esos no terminales consideremos el árbol de derivación asociado al proceso de derivación para generar **a+b*c**. Consideraremos además, que los valores de **a**, **b** y **c** son 3, 2 y 5 respectivamente.



Como se puede observar, los valores asociados a los identificadores sólo se conocen en la parte inferior del árbol. Dichos valores se deben desplazar hacia arriba en el árbol de derivación hasta llegar a las producciones en las cuales se efectúan las operaciones de suma y multiplicación. Este proceso de desplazamiento se efectúa producción por producción. Nuestra gramática queda:

1. $<E>_p \rightarrow <E>_q + <T>_r$
   $p = q + r$
2. $<E>_p \rightarrow <T>_q$
   $p = q$
3. $<T>_p \rightarrow <T>_q * <P>_r$
   $p = q * r$
4. $<T>_p \rightarrow <P>_q$
   $p = q$
5. $<P>_p \rightarrow (<E>_q)$
   $p = q$
6. $<P>_p \rightarrow I_q$
   $p = q$

Cuando los atributos se mueven desde abajo hacia arriba en el árbol de derivación se denominan atributos sintéticos.

Entonces, **atributos sintéticos** son aquellos atributos que se mueven desde abajo hacia arriba en el árbol de derivación, o dicho de otra forma, se mueven desde el lado derecho de la producción hacia el lado izquierdo.

6.9 Atributos heredados

Consideraremos ahora una gramática en la cual los datos asociados a los terminales o no terminales de una gramática se mueven desde arriba hacia abajo en el árbol de derivación.

1. <declaración> → TYPE V <listaVariables>
2. <listaVariables> → , V <listaVariables>
3. <listaVariables> → λ

Esta gramática es un ejemplo de la forma como se definen variables en un lenguaje de programación.

V es un token, cuya parte de valor, es decir su atributo, es un apuntador a su correspondiente entrada en la tabla de símbolos.

TYPE es otro token, cuya parte de valor especifica el tipo de datos que se pueden almacenar en la variable V: integer, float, boolean, etc.

En el proceso de definición de variables el trabajo del analizador sintáctico es invocar una acción **setType** la cual actualiza, en la tabla de símbolos, la entrada correspondiente a la variable V.

Esta acción se incluye en la gramática que se ha definido:

1. <declaración> → TYPE V {setType} <listaVariables>
2. <listaVariables> → , V {setType} <listaVariables>
3. <listaVariables> → λ

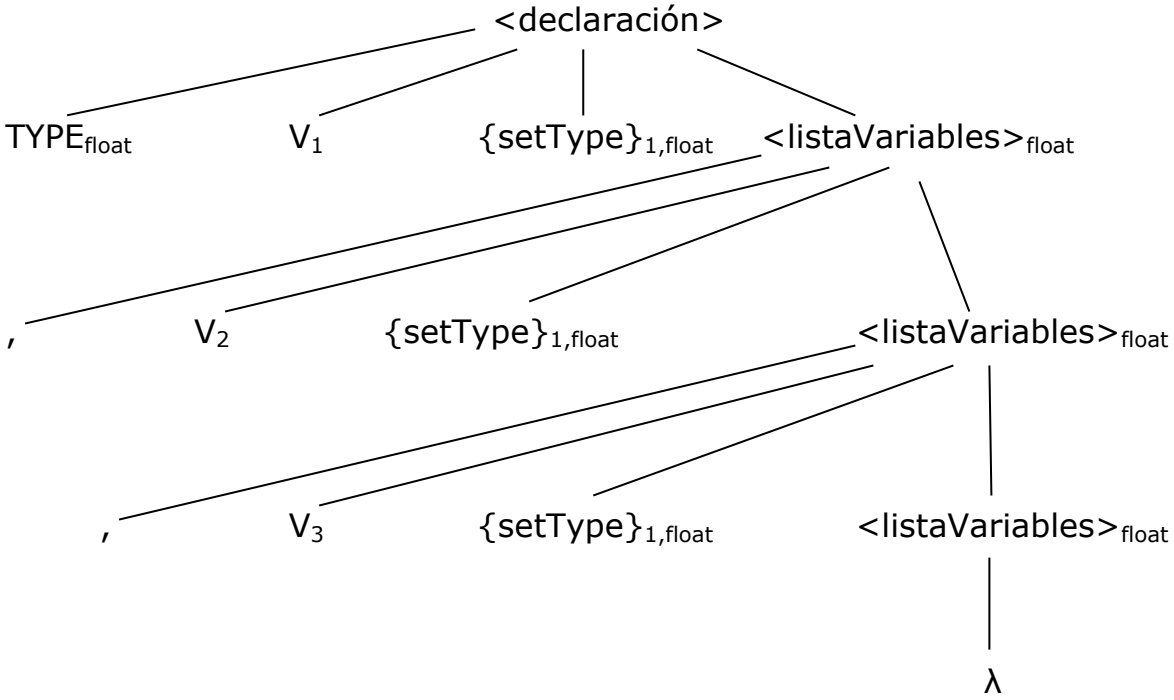
La acción {setType} tendrá dos parámetros: uno, el apuntador correspondiente a la variable V, y otro, el tipo que se le asociará a esa variable.

Nuestra gramática con atributos, y sus correspondientes reglas de evaluación de atributos, queda:

1. <declaración> → $TYPE_t$ V_p $\{setType\}_{p1,t1}$<listaVariables>$_{t2}$
   $(t_1, t_2) = t$; $p_1 = p$
2. <listaVariables>$_t$ → , V_p $\{setType\}_{p1,t1}$<listaVariables>$_{t2}$
   $(t_1, t_2) = t$; $p_1 = p$
3. <listaVariables>$_t$ → λ

Si se tiene la definición: $TYPE_{float}$ V_1, V_2, V_3

El árbol de derivación correspondiente es:



Fíjese que el atributo float del token TYPE se mueve de izquierda a derecha, en un mismo nivel, en el árbol de derivación, y que el atributo float del token <listaVariables> se mueve desde arriba hacia abajo en el árbol de derivación.

Atributos que se mueven de esa forma se denomina **atributos heredados**.

Por último, consideremos nuestra otra gramática para generar expresiones.

1. <E> → <T><listaE>
2. <listaE> → +<T><listaE>
3. <listaE> → λ

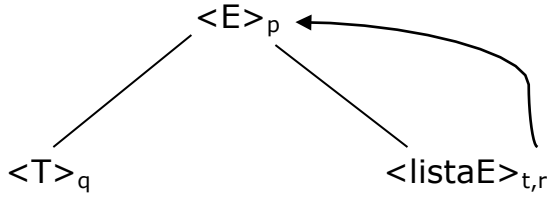
4. <T> → <P><Lista T>
5. <listaT> → *<P><listaT>
6. <listaT> → λ
7. <P> → (<E>)
8. <P> → I

Esta gramática tiene una diferencia muy grande con respecto a la otra. Para efectuar la operación de suma hay que sumar dos operandos que se encuentran en diferentes producciones: el atributo del no terminal <T> de la producción 1 con el atributo del no terminal <T> de la producción 2. Veamos cómo establecer la comunicación entre ambas producciones.

El manejo de atributos en la producción 1 es:

$$<E>_p \rightarrow <T>_q <listaE>_{t,r}$$
$$t = q; \quad p = r$$

Gráficamente, el movimiento de los atributos es así:

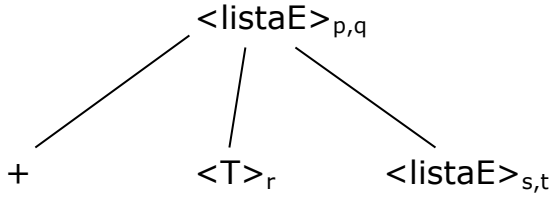


El no terminal <E> tiene un atributo sintético, el cual contendrá el resultado de evaluar la expresión. El no terminal <T> tiene un atributo sintético que contiene el valor del primer sumando. El no terminal <listaE> tiene dos atributos: el primer atributo es un atributo heredado y el segundo atributo es un atributo sintético. Si miramos el no terminal <listaE> como un subprograma, el primer parámetro es un dato de entrada y el segundo parámetro es un dato de retorno.

La ejecución del "subprograma" <listaE> es con las producciones 2 ó 3. Cuando se ejecuta la producción 2 las reglas de evaluación de atributos son:

$$<listaE>_{p,q} \rightarrow +<T>_r <listaE>_{s,t}$$
$$s = p + r; \quad q = t$$

Gráficamente el movimiento de los atributos es:



El atributo p del no terminal <listaE> del lado izquierdo de la producción es el dato de entrada, el cual, fue enviado por la producción 1. El atributo s del no terminal <listaE> del lado derecho de la producción también es un atributo de entrada para volver a ejecutar el "subprograma" <listaE>. En el "subprograma" <listaE> se

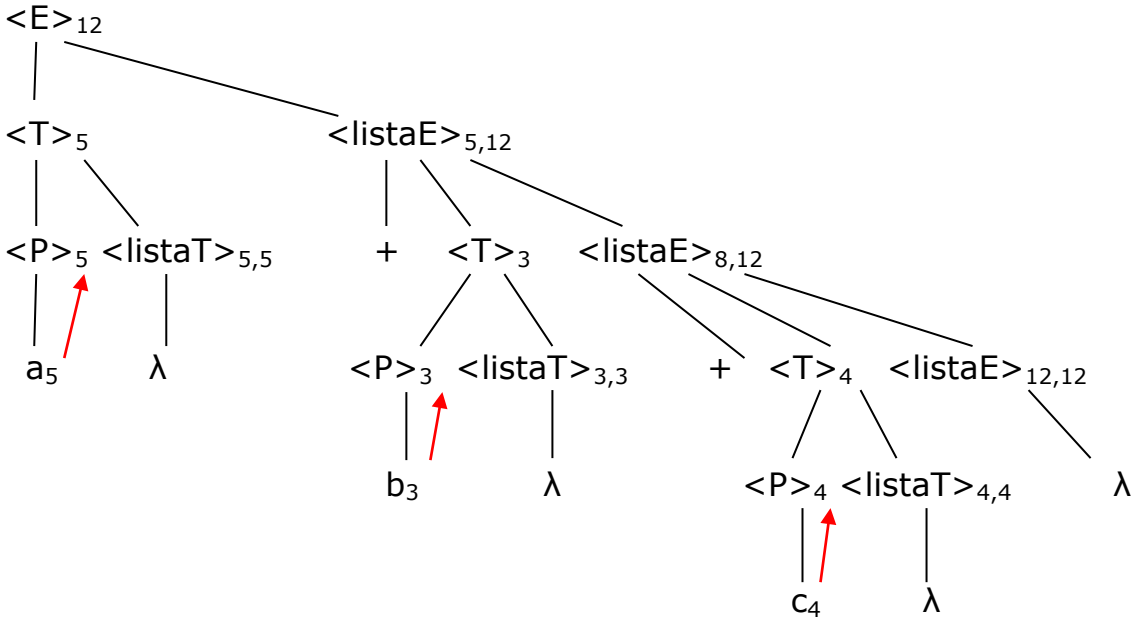
ejecuta la operación de suma y el resultado queda en el segundo atributo, es decir, el atributo t, el cual se retorna al atributo q del no terminal <listaE> del lado izquierdo de la producción para que este valor retorne a la producción 1.

Cuando se ejecuta la producción 3, cuyo lado derecho es la secuencia nula, no hay que efectuar ninguna operación, por lo tanto el dato que llega para hacer una suma, simplemente se retorna. La regla de evaluación de atributos para esta producción es:

$$<listaE>_{p,q} \rightarrow \lambda$$
$$p = q$$

La ejecución de las producciones 4, 5 y 6 es idéntica a la ejecución de las producciones 1, 2 y 3.

Veamos gráficamente cómo es el movimiento de atributos para evaluar la expresión a+b+c, considerando: a = 5; b = 3; c = 4.



1. $<E>_p \rightarrow <T>_q<listaE>_{r,s}$
   $r = q;\ \ p = s$

2. $<listaE>_{p,q} \rightarrow +<T>_r<listaE>_{s,t}$
   $s = p+r;\ \ q = t$

3. $<listaE>_{p,q} \rightarrow \lambda$
   $q = p$

4. $<T>_p \rightarrow <P>_q<listaT>_{r,s}$
   $r = q;\ \ p = s$

5. $<listaT>_{p,q} \rightarrow *<P>_r<listaT>_{s,t}$
   $s = p+r;\ \ q = t$

6. $<listaT>_{p,q} \rightarrow \lambda$
   $q = p$

7. $<P>_p \rightarrow (<E>_q)$
   $p = q$

8. $<P>_p \rightarrow I_q$
   $p = q$

6.10 Gramáticas de traducción con atributos

Ya hemos tratado lo que son gramáticas de traducción y gramáticas con atributos. Veamos ahora lo que son las gramáticas con la unión de ambas características, es decir, de traducción con atributos. Realmente este es el punto culminante para las gramáticas en los lenguajes de programación.

En los ejemplos que tratamos de gramáticas con atributos consideramos que el atributo es el valor de la variable. En realidad en los atributos se maneja es la dirección de memoria en la cual se halla almacenado el valor de la variable. Con base en esto, vamos a sintetizar con un ejemplo, el proceso de un compilador: el reconocimiento y la traducción con atributos. Consideremos la siguiente expresión en un lenguaje de alto nivel:

$$(a+b)*(a+c).$$

Cuando se definen las variables en algún lenguaje de programación, el lenguaje asigna posiciones de memoria relativas a dichas variables. En nuestro ejemplo, digamos que a la variable **a** se le asignó la posición 1, a la variable **b** la posición 2 y a la variable **c** la posición 3.

En el proceso de análisis lexicográfico el compilador reconoce los elementos de la expresión y los escribe de una manera más apropiada para el análisis sintáctico. Al ejecutar el análisis de léxico a esta expresión se obtiene:

$$(I_1+I_2)*(I_1+I_3)$$

donde I significa identificador y los números como subíndices son los atributos que indican la posición de memoria en la cual se hallan almacenados los valores correspondientes a dichos identificadores.

El analizador sintáctico, reconoce que la expresión esté bien formada de acuerdo a la gramática de expresiones y la traduce a una forma entendible por la máquina para que ésta ejecute las operaciones. La salida producida por el analizador sintáctico es:

$$\{Sume\}_{1,2,4}\{Sume\}_{1,3,5}\{Multiplique\}_{4,5,6}$$

Donde el primer símbolo de acción ($\{Sume\}_{1,2,4}$) especifica que debe sumar los contenidos de las posiciones de memoria 1 y 2, y dejar el resultado en una nueva posición de memoria, la cual estamos llamando 4; el segundo símbolo de acción ($\{Sume\}_{1,3,5}$) especifica que se deben sumar los contenidos de las posiciones de memoria 1 y 3 y dejar el resultado en una nueva posición de memoria, la cual estamos llamando 5, y el tercer símbolo de acción ($\{Multiplique\}_{4,5,6}$) le indica al ala máquina que debe multiplicar el contenido de las posiciones de memoria 4 y 5 y dejar el resultado en una nueva posición de memoria que hemos llamado 6.

Dado el anterior ejemplo veamos entonces cómo sería una gramática de traducción con atributos para expresiones.

1. $<E>_p \rightarrow <E>_q + <T>_r\{Sume\}_{s,t,u}$
   s = q; t = r; (p, u) = nuevo registro
2. $<E>_p \rightarrow <T>_q$
   p = q
3. $<T>_p \rightarrow <T>_q * <P>_r\{Multiplique\}_{s,t,u}$
   s = q; t = r; (p, u) = nuevo registro
4. $<T>_p \rightarrow <P>_q$
   p = q
5. $<P>_p \rightarrow (<E>_q)$
   p = q
6. $<P>_p \rightarrow I_q$
   p = q

EJERCICIOS PROPUESTOS

1. Construya gramática de traducción que genere 1^n0^m con n>0 y m>0 y que produzca como salida 1^n0^{2n}

2. Construya gramática de traducción que genere $1^n0^m1^m0^n$ con m y n > 0 y que produzca como salida 1^m0^{n+m}

3. Construya gramática de traducción que genere (0+1)*, lo cual representa un entero i cualquiera en binario y que produzca como salida la representación en binario, en reversa, del entero i+1

4. Construya gramática de traducción que genere (0+1)* y que produzca como salida la hilera generada en reversa.

5. Construya gramática de traducción que genere (0+1)* y que produzca como salida la misma hilera generada.

6. Construya gramática de traducción que genere (0+1)* y que produzca como salida 0^m1^n siendo m el número de ceros generados y n el número de unos generados.

7. Construya gramática de traducción que genere el lenguaje aceptado por el siguiente AF y que produzca como salida la secuencia de estados de la hilera generada. Por ejemplo, si la hilera generada es 1010 la salida es ABBCD.

	0	1	
A	A	B	0
B	B	C	1
C	D	A	0
D	A	D	1

8. Construya gramática de traducción que genere expresiones en infijo y las convierta a prefijo.

9. Construya gramática de traducción que genere expresiones en prefijo y que las convierta a posfijo.

10. Agregue atributos a la siguiente gramática

    1. <entero> → dígito <másDígitos>
    2. <másDígitos> → dígito<másDígitos>
    3. <másDígitos> → λ

    Tal que el no terminal <entero> tenga un atributo sintético que sea el valor del entero generado. Asuma que el símbolo "dígito" tiene un atributo, el cual es número entre 0 y 9.

11. Agregue atributos a la siguiente gramática

    1. <declaración> → INTEGER I <listaVariables>
    2. <listaVariables> → , I <listaVariables>
    3. <listaVariables> → λ

    Tal que el no terminal <declaración> tenga un atributo sintético, el cual contendrá el número de variables definidas a continuación de la palabra INTEGER.

12. Considere la siguiente gramática con atributos

    1. $<S>_{p4,q1} \rightarrow a_{p1}<S>_{t1,p2}b_{s1}\{c\}_{q2,p3}<A>_{r1,q3}\{d\}_{r2,s2,t2}$
       $(p_2, p_3, p_4) = p_1$; $(q_2, q_3) = q_1$; $s_2 = s_1$; $t_2 = t_1$; $r_2 = r_1$
    2. $<S>_{p2,p1} \rightarrow \lambda$
       $p_2 = p_1$
    3. $<A>_{p3,t1} \rightarrow <S>_{p1,t2}b_{q1}<A>_{r1,q2}<A>_{s1,q3}\{f\}_{r2,s2,p2}$
       $(p_2, p_3) = p_1$; $(q_2, q_3) = q_1$ $r_2 = r_1$; $s_2 = s_1$; $t_2 = t_1$
    4. $<A>_{p2,q1} \rightarrow \{e\}_{q2}c_{p1}$

       $<S>_{p,q}$ p es sintético; q es heredado
       $<A>_{p,q}$ p es sintético; q es heredado
       Los atributos de los símbolos de acción son heredados
       El valor inicial del atributo heredado de <S> es 1
       Dibuje el árbol de derivación completo, con movimiento de atributos para la secuencia: $a_1b_2a_3b_4c_5b_2c_6c_7$

SOLUCIONES

1. Construya gramática de traducción que genere 1^n0^m con n y m mayores que cero y que produzca como salida 1^n0^{2n}.

1. <S> → {1}1<A>0{00}
2. <A> → {1}1<A>{00}
3. <A> → <A>0
4. <A> → λ

3. Construya gramática de traducción que genere (0+1)*, lo cual representa un entero **i** cualquiera en binario y que produzca como salida la representación en binario, en reversa, del entero **i+1**.

1. <S> → {0}<A>1
2. <S> → {1}<B>0
3. <A> → {0}<A>1
4. <A> → {1}<B>0
5. <A> → {1}
6. <B> → {1}<B>1
7. <B> → {0}<B>0
8. <B> → {1}<C>1
9. <C> → λ

5. Construya gramática de traducción que genere (0+1)* y que produzca como salida la misma hilera generada.

1. <S> → {0}0<S>
2. <S> → {1}1<S>
3. <S> → λ

7. Construya gramática de traducción que genere el lenguaje aceptado por el siguiente AF y que produzca como salida la secuencia de estados de la hilera generada. Por ejemplo, si la hilera generada es 1010 la salida es ABBCD.

	0	1	
A	A	B	0
B	B	C	1
C	D	A	0
D	A	D	1

Construyamos primero la gramática correspondiente a dicho autómata:

1. <A> → 0<A>
2. <A> → 1<B>
3. <B> → 0<B>
4. <B> → 1<C>
5. <B> → λ
6. <C> → 0<D>
7. <C> → 1<A>
8. <D> → 0<A>
9. <D> → 1<D>

10. <D> → λ

Ahora hagamos el proceso de derivación para generar 1010:

<A>
↑ producción 2
1<B>
↑ producción 3
10<B>
↑ producción 4
101<C>
↑ producción 6
1010<D>
↑ producción 10
1010

Observe que lo que debe imprimir es el no terminal a partir del cual efectuó el proceso de derivación, en el orden que se derivó. Por tanto, se debe colocar un símbolo de acción que escriba el símbolo del lado izquierdo de la producción usada para derivar. La gramática queda:

1. <A> → {A}0<A>
2. <A> → {A}1<B>
3. <B> → {B}0<B>
4. <B> → {B}1<C>
5. <B> → {B}
6. <C> → {C}0<D>
7. <C> → {C}1<A>
8. <D> → {D}0<A>
9. <D> → {D}1<D>
10. <D> → {D}

9. Construya gramática de traducción que genere expresiones en prefijo y las convierta a posfijo.

1. <E> → +<E><E>{+}
2. <E> → -<E><E>{-}
3. <E> → *<E><E>{*}
4. <E> → /<E><E>{/}
5. <E> → I{I}

11. Agregue atributos a la siguiente gramática

1. <declaración> → INTEGER I <listaVariables>
2. <listaVariables> → , I <listaVariable>
3. <listaVariables> → λ

Tal que el no terminal <declaración> tenga un atributo sintético, el cual contendrá el número de variables definidas a continuación de la palabra INTEGER.

El atributo del no terminal <declaración> lo llamaremos **p.**

El no terminal <listaVariables> tendrá dos atributos: el primero es un atributo heredado y el segundo es un atributo sintético. En la producción uno el atributo heredado del no terminal <listaVariables> tendrá como valor 1.

La gramática con atributos y sus correspondientes reglas de evaluación de atributos queda:

1. $<declaración>_p \rightarrow$ INTEGER I $<listaVariables>_{1,r}$
   p = r
2. $<listaVariables>_{p,q} \rightarrow$, I $<listaVariable>_{r,s}$
   r = p+1; q = s
3. $<listaVariables>_{p,q} \rightarrow \lambda$
   q = p

MODULO 7
RECONOCIMIENTO DESCENDENTE. CONCEPTOS BÁSICOS

7.1 INTRODUCCIÓN
Las técnicas de reconocimiento de hileras se han desarrollado con base en las gramáticas que generan un lenguaje. Con el fin de aplicar estas técnicas es necesario construir una serie de conjuntos que facilitan la construcción de los reconocedores. Veremos en este módulo dichos conjuntos y la forma de construirlos.

7.2 OBJETIVOS
1. Conocer cómo funciona el proceso de reconocimiento descendente del lenguaje que genera una gramática.

2. Aprender a construir los conjuntos necesarios para poder elaborar el reconocedor descendente del lenguaje que genera una gramática.

7.3 PREGUNTAS BÁSICAS
1. Qué es un no terminal anulable.
2. Cuándo una producción es anulable.
3. Qué es el conjunto de primeros de un no terminal.
4. Qué es el conjunto de primeros de una producción.
5. Qué es el conjunto de siguientes de un no terminal.
6. Qué es el conjunto de selección de una producción.

7.4 Técnica descendente
Comencemos presentando la forma como funciona el reconocimiento descendente. La técnica consiste en procesar la hilera de entrada, símbolo por símbolo, e ir identificando cuál producción fue la que se aplicó para generar ese símbolo. Ilustremos esta técnica con un ejemplo.

Consideremos la siguiente gramática

1. <A> → a<B><C>b
2. <A> → b<A><C>
3. <B> → b<C>a
4. <B> → c<A>
5. <C> → c
6. <C> → a<B>

y esta hilera de entrada: **abcacb**┤

Nos interesa determinar si dicha hilera pudo haber sido generada por la gramática dada, es decir, si dicha hilera pertenece al lenguaje que genera esa gramática.

Para hacer este proceso utilizaremos un autómata de pila, en el cual, la configuración inicial de la pila es ▼<A>. En la figura siguiente presentamos la forma como va evolucionando la pila de acuerdo al símbolo de entrada que estemos procesando.

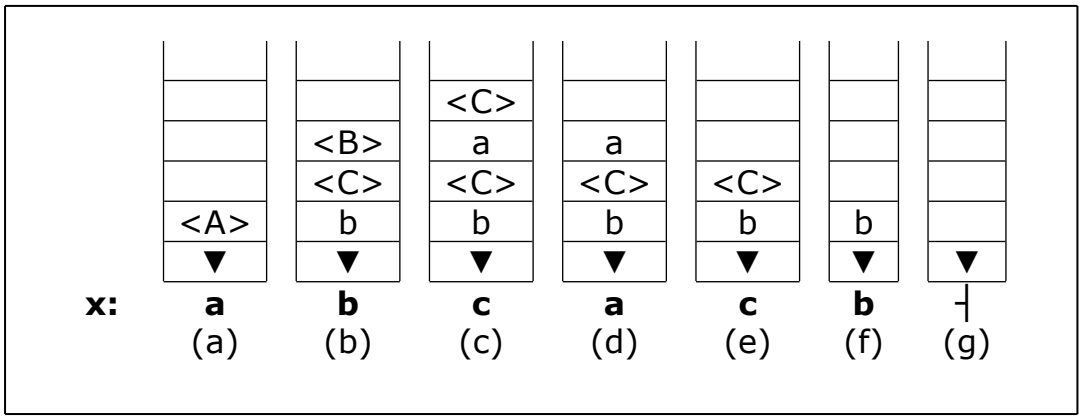


Figura 7.1

Como hemos dicho, la configuración inicial de la pila es ▼<A>, debido a que el símbolo inicial de la gramática es el no terminal <A>.

La configuración inicial de la pila siempre será el símbolo de pila vacía con el símbolo inicial de la gramática.

Estando la pila en su configuración inicial leemos el primer símbolo de la hilera a reconocer, es decir la **a**.

Esta situación se presenta en el caso **(a)** de la figura 7.1.

En el tope de la pila está el no terminal **<A>** y el símbolo de entrada es el terminal **a**.

El razonamiento aquí es: existe alguna producción cuyo símbolo del lado izquierdo sea el no terminal **<A>** y que pueda generar el terminal **a**?

La respuesta es si: la producción 1. Hemos detectado que el terminal **a** se obtuvo aplicando la producción 1, por consiguiente, debemos continuar nuestro proceso de reconocimiento con los símbolos del lado derecho de la producción 1. Para lograr esto reemplazamos el símbolo en el tope de la pila (el no terminal <A>) por el resto del lado derecho de la producción 1 en reversa, y leemos el siguiente símbolo.

Estamos en el caso **(b)** de la figura 7.1: en el tope de la pila está el no terminal <B> y el símbolo de entrada es el terminal **b**.

Nuestro razonamiento aquí será el mismo: existe alguna producción cuyo símbolo del lado izquierdo sea el no terminal **<B>** y que pueda generar el terminal **b**?

La respuesta es nuevamente si: la producción 3. Hemos detectado que el terminal **b** se obtuvo aplicando la producción 3, por consiguiente, debemos continuar nuestro proceso de reconocimiento con los símbolos del lado derecho de la producción 3. Para lograr esto reemplazamos el símbolo en el tope de la pila (el no terminal <B>) por el resto del lado derecho de la producción 3 en reversa, y leemos el siguiente símbolo.

Estamos en el caso **(c)** de la figura 7.1: en el tope de la pila está el no terminal <C> y el símbolo de entrada es el terminal **c**.

Nuestro razonamiento aquí es nuevamente el mismo: existe alguna producción cuyo símbolo del lado izquierdo sea el no terminal **<C>** y que pueda generar el terminal **c**?

La respuesta es nuevamente si: la producción 5. Hemos detectado que el terminal **c** se obtuvo aplicando la producción 5, por consiguiente, debemos continuar nuestro proceso de reconocimiento con los símbolos del lado derecho de la producción 5. Para lograr esto reemplazamos el símbolo en el tope de la pila (el no terminal <C>) por el resto del lado derecho de la producción 5 en reversa, y leemos el siguiente símbolo.

Fijémonos que como el resto del lado derecho de la producción 5 es vacío, es decir, es la secuencia nula, nuestro proceso de reemplazo es equivalente a desapilar el símbolo en el tope de la pila.

Quedamos en el caso **(d)** de la figura 7.1: en el tope de la pila hay un terminal, el terminal **a**.

Cuando en el tope de la pila hay un terminal fue porque llegó como consecuencia de una operación de reemplazo. El símbolo de entrada en ese momento debe ser exactamente ese mismo terminal para confirmar que efectivamente se aplicó esa producción (en nuestro ejemplo, la producción 3). La acción a tomar es simplemente desapilar y leer el siguiente símbolo.

Estamos ya en el caso **(e)** de la figura 7.1: en el tope de la pila está nuevamente el no terminal <C> y el símbolo de entrada es el terminal **c**.

Estamos en una situación idéntica al caso **(c)**, lo cual implica que se aplicó nuevamente la producción 5, por consiguiente, desapilamos y leemos el siguiente símbolo.

Quedamos en el caso **(f)** de la figura 7.1: en el tope de la pila hay un terminal, el terminal **b**.

Es una situación idéntica al caso **(d)**. Se está confirmando que se aplicó la producción 1, por tanto, la acción a tomar es desapile y avance.

Llegamos al caso **(g)** de la figura 7.1: en el tope de la pila está el símbolo de pila vacía (▼) y el símbolo de entrada es el símbolo de fin de secuencia (┤).

Cuando llegamos a esta situación significa que la hilera procesada sí puede ser generada por esa gramática, por tanto la acción a tomar es **Acepte**.

Como podrá haber observado, la técnica de reconocimiento descendente consiste en determinar, con base en el símbolo en el tope de la pila y el símbolo de entrada, cuál producción fue la que generó el símbolo de entrada, con el fin de llevar a la pila el resto, o el total, del lado derecho de la producción identificada para continuar con el reconocimiento.

Para efectuar ese proceso de detectar cuál producción fue la que se aplicó es necesario definir una serie de conjuntos, los cuales veremos a continuación.

7.5 Conjuntos necesarios

Convención: Sea x un terminal cualquiera.
$\alpha = (T + N)^*$

i el número de una producción.

1. **No terminales anulables:** son no terminales a partir de los cuales se puede obtener la secuencia nula **(λ)** mediante algún proceso de derivación. A este conjunto lo denominaremos $N_{anulables}$.

2. **Producciones anulables:** son producciones cuyo lado derecho es la secuencia nula **(λ)** o es anulable, es decir, su lado derecho se puede convertir en la secuencia **(λ)** mediante algún proceso de derivación. Dicho de otra forma, su lado derecho está conformado por sólo no terminales, los cuales son todos anulables.

3. **Conjunto de primeros de un no terminal (N).** Es el conjunto de terminales **(T)** con los cuales puede comenzar una hilera derivada a partir de ese no terminal **(N)**.

   3.1 Si se tiene una producción de la forma

   <S> → xα

   Entonces: primeros(<S>) = {x}

   3.2 Si se tiene una producción de la forma

   <S> → <A>xα

   Entonces:
   si <A> no es anulable primeros(<S>) = {primeros(<A>)}
   si <A> es anulable primeros(<S>) = {primeros(<A>) + x}

   3.3 Si se tiene una producción de la forma

   <S> → <A><B>xα

   Entonces:
   si <A> no es anulable
   primeros(<S>) = {primeros<A>}
   si <A> es anulable y <B> no es anulable
   primeros(<S>) = {primeros(<A>) + primeros(<B>)}
   si <A> y <B> son anulables
   primeros(<S>) = {primeros(<A>) + primeros(<B>) + x}.

4. **Conjunto de primeros de una producción i.** Es el conjunto de **T** con los cuales puede comenzar una hilera derivada a partir de esa producción.

   4.1 Si se tiene una producción de la forma

   i. <S> → xα

   Entonces:
   Primeros(i) = {x}

   4.2 Si se tiene una producción de la forma

i. <S> → <A>xα

Entonces:

Si <A> no es anulable primeros(i) = {primeros(<A>)}

Si <A> es anulable primeros(i) = {primeros(<A>) + x}

4.3 Si se tiene una producción de la forma

i. <S> → <A><B>xα

Entonces:

Si <A> no es anulable primeros(i) = {primeros(<A>)}

Si <A> es anulable y <B> no es anulable

Primeros(i) = {primeros(<A>) + primeros(<B>)}

Si <A> y <B> son anulables

Primeros(i) = {primeros(<A>) + primeros(<B>) + x}

5. Conjunto de siguientes de un N. Es el conjunto de **T** que pueden aparecer inmediatamente después de ese N mediante algún proceso de derivación.

5.1 Si se tiene una producción de la forma

<S> → α

y <S> es el símbolo inicial de la gramática entonces:

siguientes(<S>) = {┤}

5.2 Si se tiene una producción de la forma

<S> → α<A>

Entonces siguientes(<A>) = {siguientes(<S>)}

5.3 Si se tiene una subhilera <A>x en el lado derecho de alguna producción

Entonces siguientes(<A>) = {x}

5.4 Si se tiene una subhilera <A><B>x en el lado derecho de alguna producción

Entonces:

Si <B> no es anulable siguientes(<A>) = {primeros(<B>)

Si <B> es anulable siguientes(<A>) = {primeros(<B>) + x}

6. Conjunto de selección de una producción i. Es el conjunto de **T** que permite identificar cuándo se aplicó la producción **i**, cuando se está haciendo un proceso de reconocimiento descendente.

6.1 Si la producción **i** no es anulable

selección(i) = {primeros(i)}

6.2 Si la producción **i** es anulable

selección(i) = {primeros(i) + siguientes(<S>)}

Siendo <S> el símbolo del lado izquierdo de la producción **i**.

Es bueno resaltar que cuando el lado derecho de una producción es la secuencia nula **(λ)** el conjunto de selección de esa producción es sòlamente los siguientes del símbolo del lado izquierdo, ya que el conjunto de primeros de dicha producción es vacío.

7.6 Ejemplo.

Para ilustrar cómo identificar no terminales anulables, producciones anulables y construir los conjuntos definidos anteriormente, en una gramática, vamos a considerar el siguiente ejemplo.

1. <A> → a<B><C>
2. <A> → <D>b<A>
3. <B> → λ
4. <B> → b<A><B>
5. <C> → c<C>
6. <C> → <D>d<B>
7. <D> → λ
8. <D> → e<E>
9. <E> → <B><D>
10. <E> → f

Comencemos construyendo el conjunto de no terminales anulables:

1. $\mathbf{N_{anulables}}$ = {<B>, <D>, <E>}

Los no terminales **<B>** y **<D>** son anulables en virtud de las producciones 3 y 7, en las cuales, el lado derecho es la secuencia nula **(λ)**; el no terminal **<E>** es anulable en virtud de la producción 9, cuyos símbolos del lado derecho son todos anulables.

2. Producciones anulables = {3, 7, 9}

Las producciones 3 y 7 son anulables porque el lado derecho de dichas producciones es la secuencia nula **(λ)**, y la producción 9 es anulable porque todos los símbolos del derecho de dicha producción son anulables.

3. Conjunto de primeros de cada no terminal (N).

Primeros(<A>) = {a, b, e}
Primeros(<B>) = {b}
Primeros(<C>) = {c, d, e}
Primeros(<D>) = {e}
Primeros(<E>) = {b, e, f}

El conjunto de primeros del no terminal <A> es: el terminal **a** por la producción 1, y los **primeros de <D>** y el terminal **b** por la producción 2. El terminal **b** entra por la producción 2 porque el no terminal <D> es anulable. Por consiguiente, los primeros(<A>) son **a**, **b**, y **e**.

El conjunto de primeros del no terminal <B> es sólo el terminal **b** porque el lado derecho de la producción 4 comienza con ese terminal. La producción 3 no aporta símbolos porque su lado derecho es la secuencia nula **(λ)**.

El conjunto de primeros del no terminal <C> es: el terminal **c** por la producción 5, y los **primeros de <D>** y el terminal **d** por la producción 6. El terminal **d** entra por la producción 6 porque el no terminal <D> es anulable. Por consiguiente, los primeros(<C>) son **c**, **d** y **e**.

El conjunto de primeros del no terminal <D> es sólo el terminal **e** porque el lado derecho de la producción 8 comienza con ese terminal. La producción 7 no aporta símbolos porque su lado derecho es la secuencia nula **(λ)**.

El conjunto de primeros del no terminal <E> es: el terminal **f** por la producción 10, y los **primeros de <B>** más los **primeros de <D>** por la producción 9. Los primeros de <D> entran por la producción 9 porque el no terminal <B> es anulable. Por consiguiente, los primeros(<E>) son **b**, **e** y **f**.

Es supremamente importante que entienda las implicaciones que tienen los no terminales anulables.

4. Conjunto de primeros de cada producción:

1. <A> → a<B><C>	primeros(1)	= {a}
2. <A> → <D>b<A>	primeros(2)	= {b, e}
3. <B> → λ	primeros(3)	= {}
4. <B> → b<A><B>	primeros(4)	= {b}
5. <C> → c<C>	primeros(5)	= {c}
6. <C> → <D>d<B>	primeros(6)	= {d, e}
7. <D> → λ	primeros(7)	= {}
8. <D> → e<E>	primeros(8)	= {e}
9. <E> → <B><D>	primeros(9)	= {b, e}
10. <E> → f	primeros(10)	= {f}

El conjunto de primeros de las producciones 1, 4, 5, 8 y 10 es simple de construir: cuando el lado derecho de una producción comienza con un terminal, ese terminal es el conjunto de selección de esa producción. Las producciones 1, 4, 5, 8 y 10, su lado derecho comienza con un terminal.

El conjunto de primeros de las producciones 3 y 7 es vacío, puesto que el lado derecho de estas producciones es la secuencia nula **(λ)**.

El conjunto de primeros de la producción 2 es: los **primeros(<D>)** más el terminal **b**, puesto que el no terminal <D> es anulable. Los primeros de <D> es el terminal **e**, por consiguiente, los primeros de la producción 2 son **b** y **e**.

El conjunto de primeros de la producción 6 es: los **primeros(<D>)** más el terminal **d**, puesto que el no terminal <D> es anulable. Los primeros de <D> es el terminal **e**, por consiguiente, los primeros de la producción 2 son **d** y **e**.

El conjunto de primeros de la producción 9 es: los **primeros(<B>)** más los **primeros(<D>)**: los primeros de <B> porque el lado derecho comienza con un no terminal, y los primeros de <D> porque el no terminal <B> es anulable. Por tanto, los primeros de la producción 9 son los terminales **b** y **e**.

Es supremamente importante que entienda las implicaciones que tienen los no terminales anulables.

5. Conjunto de siguientes de cada no terminal (N).

Siguientes(<A>) = {b, c, d, e, ┤}
Siguientes(<B>) = {b, c, d, e, ┤}
Siguientes(<C>) = {b, c, d, e, ┤}
Siguientes(<D>) = {b, d}
Siguientes(<E>) = {b, d}

Siguientes(<A>). El conjunto de siguientes del no terminal <A> es: el símbolo de fin de secuencia **(┤)**, puesto que el no terminal <A> es el símbolo inicial de la gramática (Recuerde que el símbolo de fin de secuencia, por derecho propio, pertenece al conjunto de siguientes del símbolo inicial de la gramática); luego analizamos el lado derecho de cada producción de la gramática buscando en dónde se halla el no terminal <A>. Este se halla en el lado derecho de la producción 2 y en el lado derecho de la producción 4. Como se encuentra de último en el lado derecho de la producción 2, los siguientes de <A> serán los siguientes del símbolo del lado izquierdo de dicha producción. El símbolo del lado izquierdo de la producción 2 es el mismo no terminal <A>, por consiguiente, los siguientes de <A>, por la producción 2, es el mismo símbolo fin de secuencia **(┤)**; en la producción 4, el símbolo a continuación del no terminal <A> es el no terminal <B>, por tanto los siguientes de <A>, por la producción 4, son los primeros de <B>, es decir, el terminal **b**. **Pero**, el no terminal <B> es anulable, por tanto, el no terminal <A> puede quedar de último en el lado derecho de esta producción y entonces los siguientes de <A> serán también los siguientes del símbolo del lado izquierdo de esta producción, es decir, los siguientes del no terminal <B>, los cuales, como veremos más adelante son b, c, d, e y ┤. Por consiguiente, los siguientes de <A> son **b**, **c**, **d**, **e** y ┤.

Es supremamente importante que entienda las implicaciones que tienen los no terminales anulables.

Siguientes(<B>). Para construir el conjunto de siguientes del no terminal <B> debemos analizar el lado derecho de cada producción, buscando en dónde se halla el no terminal <B>. El no terminal <B> lo encontramos en el lado derecho de las producciones 1, 4, 6 y 9. En la producción 1, el símbolo a continuación del no terminal <B> es el no terminal <C>, por consiguiente los siguientes de <B>, por la producción 1, son los primeros de <C>, es decir, **c**, **d** y **e**; en la producción 4 el no terminal <B> se halla de último en el lado derecho de dicha producción, por tanto los siguientes de <B>, por la producción 4 son los siguientes del símbolo del lado izquierdo de dicha producción, o sea, los siguientes de <B>, los cuales son los mismos que se encuentren en las otras

producciones; en la producción 6, el no terminal <B> se halla también de último en el lado derecho, por consiguiente los siguientes de <B>, por la producción 6, son los siguientes de <C>, que como veremos a continuación son b, c, d, e y ┤; en la producción 9, los siguientes de <B> son los primeros de <D>, el cual es el símbolo a continuación del no terminal <B>, más los siguientes de <E>, ya que como el no terminal <D> es anulable entonces el no terminal <B> queda ocupando también el último sitio en la producción 9 y sus siguientes son el conjunto de siguientes del símbolo del lado izquierdo de la producción.

Siguientes(<C>). Al igual que con el no terminal <B>, debemos buscar el no terminal <C> en el lado derecho de cada producción. El no terminal <C> se halla en el lado derecho de las producciones 1 y 5. Por la producción 1, los siguientes de <C> son los siguientes del símbolo del lado izquierdo, es decir, los siguientes de <A> (b, c, d, e, ┤); por la producción 5 los siguientes de <C> son los mismos siguientes de <C>, ya que el no terminal <C> se halla de último en el lado derecho de dicha producción y el símbolo del lado izquierdo de la misma producción es el mismo no terminal <C>.

Siguientes(<D>). Nuevamente buscamos el no terminal <D> en el lado derecho de cada producción. El no terminal <D> lo encontramos en el lado derecho de las producciones 2, 6 y 9. Por la producción 2, el símbolo a continuación es el terminal b; por la producción 6 el símbolo a continuación es el terminal d, y por la producción 9, los siguientes de <D> son los siguientes de <E>, que como veremos a continuación los siguientes de <E> son **b** y **d**.

Siguientes(<E>). Al buscar el no terminal <E> en el lado derecho de cada producción, éste sólo se halla en el lado derecho de la producción 8, por consiguiente los siguientes de <E> serán los siguientes del no terminal <D>, los cuales son **b** y **d**.

6. Conjunto de selección de cada producción:

1. <A> → a<B><C> selección(1) = {a}
2. <A> → <D>b<A> selección(2) = {b, e}
3. <B> → λ selección(3) = {b, c, d, e, ┤}
4. <B> → b<A><B> selección(4) = {b}
5. <C> → c<C> selección(5) = {c}
6. <C> → <D>d<B> selección(6) = {d, e}
7. <D> → λ selección(7) = {b, d}
8. <D> → e<E> selección(8) = {e}
9. <E> → <B><D> selección(9) = {b, d, e}
10. <E> → f selección(10) = {f}

Como dijimos, la construcción del conjunto de selección de cada producción depende de si la producción es anulable o no. Si la producción no es anulable, el conjunto de selección es el conjunto de primeros de dicha producción. Si la producción es anulable el conjunto de selección es el conjunto de primeros más el conjunto de siguientes del símbolo del lado izquierdo de dicha producción.

En nuestro ejemplo, las producciones 1, 2, 4, 5, 6, 8 y 10 no son anulables, por consiguiente, el conjunto de selección de cada una de ellas es el conjunto de primeros que construimos en 4.

El conjunto de selección de las producciones 3 y 7 es el conjunto de siguientes del símbolo del lado izquierdo, puesto que el lado derecho de dichas producciones es la secuencia nula **(λ)**. Por tanto, el conjunto de selección de la producción 3 son los siguientes de <B>, es decir, b, c, d, e y ┤, y el conjunto de selección de la producción 7 son los siguientes de <D>, es decir b y d.

El conjunto de selección de la producción 9 son los primeros(9) más los siguientes de <E>, es decir, **b**, **d** y **e**.

Asegúrese de haber entendido bien este ejemplo y los conceptos desarrollados en él. Es supremamente importante para los capítulos siguientes.

EJERCICIOS PROPUESTOS

Construya conjunto de primeros y de siguientes para cada uno de los no terminales, y conjunto de primeros y de selección para cada una de las producciones de las siguientes gramáticas

1.
   1. <S> → <S>c
   2. <S> → c<A>
   3. <S> → λ
   4. <A> → a<A>
   5. <A> → a

2.
   1. <S> → b<A><D>c
   2. <D> → <G>i
   3. <A> → a<G>s
   4. <G> → λ

3.
   1. <S> → <A><S><B>
   2. <S> → a
   3. <A> → a<A>
   4. <A> → λ
   5. <B> → b<B>
   6. <B> → λ

4.
   1. <S> → a<B><D>
   2. <S> → <A><B>
   3. <S> → <D><A><C>
   4. <S> → b
   5. <A> → <S><C><B>
   6. <A> → <S><A><B><C>
   7. <A> → <C>b<D>
   8. <A> → c
   9. <A> → λ
   10. <B> → c<D>
   11. <B> → d
   12. <B> → λ

13. <C> → <A><D><C>
14. <C> → c
15. <D> → <S>a<C>
16. <D> → <S><C>
17. <D> → fg

5.

1. <S> → <A><B><C><D>
2. <A> → a<A>p
3. <A> → λ
4. <B> → <B>q
5. <B> → b
6. <C> → <A><F>
7. <D> → d<D>r
8. <D> → λ
9. <F> → f<F>g
10. <F> → λ

6.

1. <A> → <A>c<B>
2. <A> → c<C>
3. <A> → <C>
4. <B> → b<B>
5. <B> → i
6. <C> → <C>a<B>
7. <C> → <B>b<B>
8. <C> → <B>

SOLUCIONES

1. Construir los conjuntos vistos para la siguiente gramática

1. <S> → <S>c
2. <S> → c<A>
3. <S> → λ
4. <A> → a<A>
5. <A> → a

No terminales anulables = {<S>}
Producciones anulables = {3}

Primeros(<S>) = {c}
Primeros(<A>) = {a}

Primeros(1) = {c}
Primeros(2) = {c}
Primeros(3) = {}
Primeros(4) = {a}
Primeros(5) = {a}

Siguientes(<S>) = {c, ┤}
Siguientes(<A>) = {c, ┤}

Selección(1) = {c}
Selección(2) = {c}
Selección(3) = {c, ┤}
Selección(4) = {a}
Selección(5) = {a}

3. Construya conjuntos vistos para la siguiente gramática

1. <S> → <A><S><B>
2. <S> → a
3. <A> → a<A>
4. <A> → λ
5. <B> → b<B>
6. <B> → λ

No terminales anulables = {<A>, <B>}
Producciones anulables = {4, 6}

Primeros(<S>) = {a}
Primeros(<A>) = {a}
Primeros(<B>) = {b}

Primeros(1) = {a}
Primeros(2) = {a}
Primeros(3) = {a}
Primeros(4) = {}

Primeros(5) = {b}
Primeros(6) = {}

Siguientes(<S>) = {b, ┤}
Siguientes(<A>) = {a}
Siguientes(<B>) = {b, ┤}

Selección(1) = {a}
Selección(2) = {a}
Selección(3) = {a}
Selección(4) = {a}
Selección(5) = {b}
Selección(6) = {b, ┤}

5. Construya conjuntos vistos para la siguiente gramática

1. <S> → <A><B><C><D>
2. <A> → a<A>p
3. <A> → λ
4. <B> → <B>q
5. <B> → b
6. <C> → <A><F>
7. <D> → d<D>r
8. <D> → λ
9. <F> → f<F>g
10. <F> → λ

No terminales anulables = {<A>, <C>, <D>, <F>}
Producciones anulables = {3, 6, 8, 10}

Primeros(<S>) = {a, b}
Primeros(<A>) = {a}
Primeros(<B>) = {b}
Primeros(<C>) = {a, f}
Primeros(<D>) = {d}
Primeros(<F>) = {f}

Primeros(1) = {a, b}
Primeros(2) = {a}
Primeros(3) = {}
Primeros(4) = {b}
Primeros(5) = {b}
Primeros(6) = {a, f}
Primeros(7) = {d}
Primeros(8) = {}
Primeros(9) = {f}
Primeros(10) = {}

Siguientes(<S>) = {┤}
Siguientes(<A>) = {b, d, f, p, ┤}
Siguientes(<B>) = {a, d, f, q, ┤}

Siguientes(<C>) = {d, ┤}
Siguientes(<D>) = {r, ┤}
Siguientes(<F>) = {d, g, ┤}

Selección(1) = {a, b}
Selección(2) = {a}
Selección(3) = {b, d, f, p, ┤}
Selección(4) = {b}
Selección(5) = {b}
Selección(6) = {a, d, f, ┤}
Selección(7) = {d}
Selección(8) = {r, ┤}
Selección(9) = {f}
Selección(10) = {d, g, ┤}

MODULO 8
GRAMATICAS S, Q y LL(1)

8.1 INTRODUCCIÓN
Presentamos en el módulo anterior los conjuntos necesarios para construir un reconocedor descendente del lenguaje que genera una gramática. Veremos en este módulo cómo se clasifican algunas gramáticas según las propiedades encontradas en ellas, con base en los conjuntos construidos, y la forma como se construye el reconocedor de acuerdo a esa clasificación.

8.2 OBJETIVOS
1. Reconocer cuándo una gramática es una gramática S.
2. Reconocer cuándo una gramática es una gramática Q.
3. Reconocer cuándo una gramática es una gramática LL(1).
4. Construir el reconocedor descendente de pila de una gramática S, Q ó LL(1).

8.3 PREGUNTAS BÁSICAS
7. Qué es un parser.
8. Qué es una gramática S.
9. Qué es una gramática Q.
10. Qué es una gramática LL(1).

8.4 Gramáticas S.
Son gramáticas en las cuales el lado derecho de cada producción comienza con un terminal **(T)**, y producciones cuyo símbolo del lado izquierdo es el mismo, los terminales con los cuales comienzan sus lados derechos son diferentes.

Simbólicamente, cada producción es de la forma:

$$\mathbf{<N> \rightarrow T\alpha}$$

Siendo α una hilera cualquiera de terminales **(T)** y no terminales **(N)**, la cual puede ser la secuencia nula (**λ**).

Simbólicamente, α pertenece a $\mathbf{(T + N)^*}$

Veamos cómo construir el autómata de pila para reconocer el lenguaje generado por una gramática **S**.

Definamos una convención general para desarrollar nuestro método:

Sea:
A: símbolo en el tope de la pila.
x: símbolo de entrada.

1. Símbolos de entrada: Los terminales **(T)** de la gramática más el símbolo de fin de secuencia **(┤)**.
2. Símbolos en la pila: los no terminales **(N)** de la gramática, todo terminal **(T)** que pertenezca a alguna hilera α y el símbolo de pila vacía (▼).
3. Configuración inicial de la pila: **▼S**, siendo **S** el símbolo inicial de la gramática.
4. Transiciones:

4.1 Existe una producción de la forma:

A → xα

La transición es: **Replace(α^r), Avance**
Si **α** es la secuencia nula (**λ**) la transición queda: **Desapile, Avance**.

4.2 **A** es un terminal **(T)**, entonces en la intersección de **A** con **x** (**A == x**) la transición es:

Desapile, Avance.

4.3 **A** es ▼, y **x** es ┤, entonces la transición es: **Acepte**.

4.4 Cualquier otra situación es: **Rechace**.

Consideremos la siguiente gramática:

1. <B> → b<C>d
2. <B> → a<B>
3. <C> → b<C>d
4. <C> → a

Es una gramática de tipo **S**: El lado derecho de cada producción comienza con un terminal **(T)**, y producciones cuyo símbolo del lado izquierdo es el mismo (producciones 1 y 2, y producciones 3 y 4), los terminales con los cuales comienzan sus lados derechos son diferentes.

Para construir el autómata de pila con el cual se reconoce el lenguaje que genera dicha gramática se procede así:

1. Símbolos de entrada = {a, b, d, ┤}
2. Símbolos en la pila = {<B>, <C>, d, ▼}
3. Configuración inicial de la pila = ▼<B>
4. Tabla de transiciones:

	a	b	d	┤
<B>	#2	#1		
<C>	#4	#3		
d			#0	
▼				**A**

#0: Desapile, avance
#1: Replace(d<C>), avance
#2: Replace(<B>), avance
#3: Replace(d<C>), avance
#4: Desapile, avance
A: Acepte

Los cajones en blanco son situación de rechazo.

Las gramáticas **S** tienen dos restricciones grandes:

1. No admiten producciones cuyo lado derecho sea la secuencia nula (**λ**).
2. No admiten producciones cuyo lado derecho comience con un no terminal **(N)**.

8.5 Gramáticas Q

Si quitamos la primera restricción, es decir, admitimos producciones cuyo lado derecho sea la secuencia nula (**λ**), las gramáticas se denominan gramáticas **Q**, siempre y cuando el conjunto de selección de esa producción sea disyunto con los terminales con los cuales comienzan otras producciones con el mismo símbolo en el lado izquierdo y, obviamente, producciones cuyo símbolo del lado izquierdo sea el mismo y su lado derecho comience con un terminal, estos terminales son diferentes.

Recuerde que el conjunto de selección de una producción, cuyo lado derecho sea la secuencia nula **(λ),** es el conjunto de siguientes del símbolo del lado izquierdo.

Cuando la gramática sea una gramática **Q**, la construcción del autómata de pila que reconoce el lenguaje que genera dicha gramática, es así:

Sea:

A: símbolo en el tope de la pila.
x: símbolo de entrada.

1. Símbolos de entrada: Los terminales **(T)** de la gramática más el símbolo de fin de secuencia **(┤)**.
2. Símbolos en la pila: los no terminales de la gramática **(N)**, todo terminal **(T)** que pertenezca a alguna hilera **α** y el símbolo de pila vacía (▼).
3. Configuración inicial de la pila: **▼S**, siendo **S** el símbolo inicial de la gramática.
4. Transiciones:

   4.1 Existe una producción de la forma:

   A → xα

   La transición es: **Replace(α^r), Avance**
   Si **α** es la secuencia nula (**λ**) la transición queda: **Desapile, Avance**.

   4.2 Existe una producción de la forma:

   A → λ

   y **x** pertenece al conjunto de selección de esa producción, la transición es:

   Desapile, retenga

   4.3 **A** es un terminal **(T)**, entonces en la intersección de **A** con **x** (**A == x**) la transición es:

   Desapile, Avance.

   4.4 **A** es ▼, y **x** es ┤, entonces la transición es: **Acepte**.

4.5 Cualquier otra situación es: **Rechace**.

Consideremos como ejemplo la siguiente gramática:

1. <A> → a<C>
2. <A> → b<B>c
3. <B> → a<A>
4. <B> → b<B>
5. <C> → a<C>c
6. <C> → b<B>cc
7. <C> → λ

Fijémonos que el lado derecho de cada producción comienza con un terminal, excepto el lado derecho de la producción 7, el cual es la secuencia nula **(λ)**.

Producciones 1 y 2, cuyo símbolo del lado izquierdo es el mismo, los terminales con los cuales comienzan sus lados derechos son diferentes: **a** y **b**.

Producciones 3 y 4, cuyo símbolo del lado izquierdo es el mismo, los terminales con los cuales comienzan sus lados derechos son diferentes: **a** y **b**.

El conjunto de selección de la producción 7 está conformado por los símbolos **c** y ┤.

Entonces, el conjunto de selección de la producción 5 es el terminal **a**, el conjunto de selección de la producción 6 es el terminal **b**, y el conjunto de selección de la producción 7 son los símbolos **c** y ┤. Por consiguiente, los conjuntos de selección de las producciones 5, 6 y 7 son disyuntos.

Nuestra gramática es una gramática Q.

El autómata de pila que reconoce el lenguaje generado por esa gramática es:

1. Símbolos de entrada = {a, b, c, ┤}
2. Símbolos en la pila = {<A>, <B>, <C>, c, ▼}
3. Configuración inicial de la pila = ▼<A>
4. Tabla de transiciones:

	a	b	c	┤
<A>	#1	#2		
<B>	#3	#4		
<C>	#5	#6	#7	#7
c			#0	
▼				**A**

#0: Desapile, avance
#1: Replace(<C>), avance
#2: Replace(c<B>), avance
#3: Replace(<A>), avance
#4: Replace(<B>), avance
#5: Replace(c<C>), avance
#6: Replace(cc<B>), avance

#7: Desapile, retenga
A: Acepte

Los cajones en blanco son situación de rechazo.

Para ilustrar cómo funciona dicho autómata consideremos la siguiente hilera:

abaacc┤

En la figura 8.1 mostramos el desarrollo del proceso de reconocimiento.

El caso **(a)** representa el estado inicial del autómata de pila y el primer símbolo de entrada.

En el tope de la pila se halla el no terminal **<A>** y el símbolo de entrada es el terminal **a**. En este caso se aplica la transición correspondiente a la producción 1 y se lee el siguiente símbolo, quedando la pila como en el caso (b) de la figura 8.1.

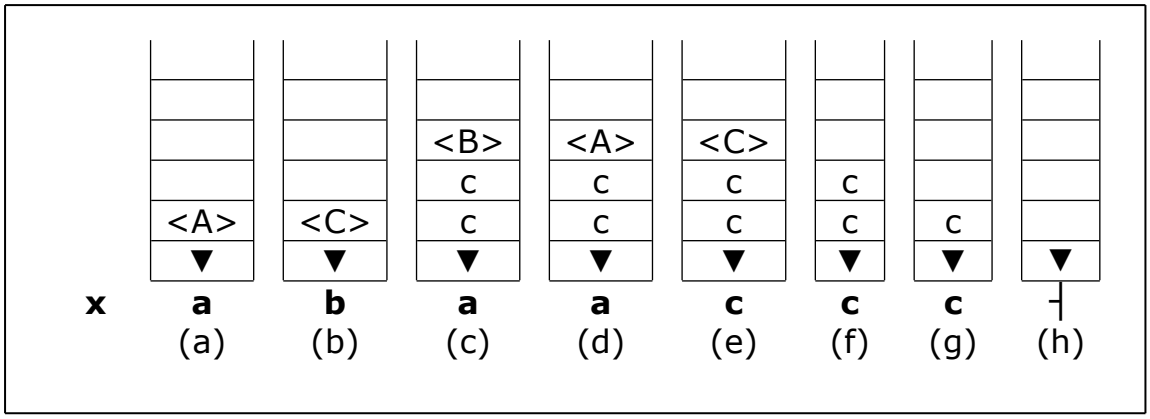


Figura 8.1

Estamos en el caso **(b)** de la figura 8.1: en el tope de la pila está el no terminal <C> y el símbolo de entrada es el terminal b. En esta situación se aplica la transición correspondiente a la producción 6 y se lee el siguiente símbolo, quedando la pila como en el caso (c) de la figura 8.1.

Estamos en el caso **(c)** de la figura 8.1: en el tope de la pila está el no terminal <B> y el símbolo de entrada es el terminal a. En esta situación se aplica la transición correspondiente a la producción 3 y se lee el siguiente símbolo, quedando la pila como en el caso (d) de la figura 8.1.

Estamos en el caso **(d)** de la figura 8.1: en el tope de la pila está el no terminal <A> y el símbolo de entrada es el terminal a. En esta situación se aplica la transición correspondiente a la producción 1 y se lee el siguiente símbolo, quedando la pila como en el caso (e) de la figura 8.1.

Estamos en el caso **(e)** de la figura 8.1: en el tope de la pila está el no terminal <C> y el símbolo de entrada es el terminal c. En esta situación se aplica la transición correspondiente a la producción 7, la cual sólo desapila el símbolo en el tope de la pila y retiene el símbolo de entrada, quedando la pila como en el caso (f) de la figura 8.1.

Estamos en el caso **(f)** de la figura 8.1: en el tope de la pila está el terminal c y el símbolo de entrada es el terminal c. En esta situación se aplica la transición correspondiente a cuando en el tope de la pila se halla un terminal, es decir, desapila y lee el siguiente símbolo, quedando la pila como en el caso (g) de la figura 8.1.

Estamos en el caso **(g)** de la figura 8.1: en el tope de la pila está el terminal c y el símbolo de entrada es el terminal c. Es la misma situación que el caso (g), por tanto, la transición es la misma, se desapila y lee el siguiente símbolo, quedando la pila como en el caso (h) de la figura 8.1.

En el caso **(h)** de la figura 8.1 el símbolo en el tope de la pila es el símbolo de pila vacía **(▼)** y el símbolo de entrada es el símbolo de fin de secuencia **(┤)**, por consiguiente, se acepta la hilera de entrada.

8.6 Gramáticas LL(1)

Son gramáticas en las cuales, producciones cuyo símbolo del lado izquierdo es el mismo, tienen conjuntos de selección disyuntos.
En general, las producciones pueden ser de las siguientes formas:

1. <N> → T**α**
2. <N> → **β**
3. <N> → **λ**

Siendo:
T un terminal, **α** una hilera cualquiera de terminales y no terminales que puede ser la secuencia nula, es decir, **α** pertenece a (N+T)*, **β** una hilera cualquiera de terminales y no terminales que comienza con un no terminal y **λ** la secuencia nula.

Cuando la gramática es LL(1) se podrá construir el reconocedor descendente de pila para el lenguaje que ella genera. Los pasos para construir este autómata de pila seguimos los siguientes pasos:
Sea:

A: símbolo en el tope de la pila.
x: símbolo de entrada.

1. Símbolos de entrada: Los terminales **(T)** de la gramática más el símbolo de fin de secuencia **(┤)**.
2. Símbolos en la pila: los no terminales de la gramática **(N)**, todo terminal **(T)** que pertenezca a alguna hilera **α** ó **β,** y el símbolo de pila vacía (▼).
3. Configuración inicial de la pila: **▼S**, siendo **S** el símbolo inicial de la gramática.
4. Transiciones:

   4.1 Existe una producción de la forma:

   A → xα

   La transición es: **Replace(α^r), avance**
   Si **α** es la secuencia nula (**λ**) la transición queda: **Desapile, Avance**.

   4.2 Existe una producción de la forma:

A → β

Y **x** pertenece al conjunto de selección de esa producción, la transición es:

Replace(β^r), retenga

4.3 Existe una producción de la forma:

A → λ

y **x** pertenece al conjunto de selección de esa producción, la transición es:

Desapile, retenga

4.4 **A** es un terminal **(T)**, entonces en la intersección de **A** con **x** (**A == x**) la transición es:

Desapile, Avance.

4.5 **A** es ▼, y **x** es ┤, entonces la transición es: **Acepte**.

4.6 Cualquier otra situación es: **Rechace**.

Consideremos como ejemplo la siguiente gramática:

1. <A> → <B>c<D>
2. <A> → a<E>
3. <B> → b<A>c
4. <B> → λ
5. <D> → d<B><D>c
6. <D> → λ
7. <E> → a
8. <E> → <B><D>

Primero debemos determinar si es LL(1) o no. Para determinar si es LL(1) debemos construir el conjunto de selección de cada producción y comprobar que cumple la condición de que producciones con el mismo símbolo en el lado izquierdo tengan conjuntos de selección disyuntos.

Para construir el conjunto de selección de cada producción seguimos el proceso definido en el módulo 7.

1. No terminales anulables = {<B>, <D>, <E>}
   Los no terminales <B> y <D> son anulables en virtud de las producciones 4 y 6, cuyo lado derecho es la secuencia nula **{λ}**, el no terminal <E> es anulable en virtud de la producción 8, ya que su lado derecho está conformado exclusivamente por no terminales anulables.

2. Producciones anulables = {4, 6, 8}

3. Conjunto de primeros de cada no terminal:

Primeros(<A>) = {a, b, c}
Primeros(<B>) = {b}
Primeros(<D>) = {d}
Primeros(<E>) = {a, b, d}

Los primeros(<B>) y los primeros(<D>) son bastante simples: por las producciones 4 y 5 obtenemos sólo terminal b para el no terminal <B>, y por las producciones 5 y 6 obtenemos sólo el terminal d para el no terminal <D>.

Los primeros(<A>): por la producción 1 son los primeros de <B> más el terminal c, ya que el no terminal <B> es anulable, y por la producción 2 obtenemos el terminal a.

Los primeros(<E>): por la producción 7 obtenemos el terminal a, y por la producción 8 serán los primeros(<B>) más los primeros(<D>) ya que el no terminal <B> es anulable. Si hubiera más símbolos a continuación del no terminal <D> habría que agregar dicho símbolo al conjunto de primeros de <E>, en virtud de la producción 8.

4. Conjunto de primeros de cada producción:
   Primeros(1) = {b, c}
   Primeros(2) = {a}
   Primeros(3) = {b}
   Primeros(4) = {}
   Primeros(5) = {d}
   Primeros(6) = {}
   Primeros(7) = {a}
   Primeros(8) = {b, d}

   Los primeros(1): son los primeros de <B> más el terminal c, ya que el no terminal <B> es anulable.

   Los primeros de las producciones 2, 3, 5 y 7 son simplemente el terminal con el que comienza el lado derecho de cada una de esas producciones.

   Los primeros de las producciones 4 y 6 son vacío, ya que el lado derecho de dichas producciones es la secuencia nula **{λ}**.

   Los primeros(8): son los primeros(<B>) más los primeros(<D>), ya que el no terminal <B> es anulable.

5. Conjunto de siguientes de cada no terminal:
   Siguientes(<A>) = {c, ┤}
   Siguientes(<B>) = {c, d, ┤}
   Siguientes(<D>) = {c, ┤}
   Siguientes(<E>) = {c, ┤}

6. Conjunto de selección de cada producción:
   Selección(1) = {b, c}
   Selección(2) = {a}
   Selección(3) = {b}

Selección(4) = {c, d, ┤}
Selección(5) = {d}
Selección(6) = {c, ┤}
Selección(7) = {a}
Selección(8) = {b, c, d, ┤}

Producciones 1, 2, 3, 5 y 7 son producciones no anulables, por consiguiente, el conjunto de selección de ellas es el conjunto de primeros correspondiente.

Producciones 4 y 6, el lado derecho es la secuencia nula, por tanto, el conjunto de selección de ellas es el conjunto de siguientes del símbolo del lado izquierdo.

Producción 8 es una producción anulable, por tanto, el conjunto de selección es el conjunto de primeros de esa producción más el conjunto de siguientes del símbolo del lado izquierdo.

7. Determinar si la gramática es LL(1):
   Ahora, para determinar si la gramática es LL(1) fijémonos cómo son los conjuntos de selección de producciones cuyo símbolo del lado izquierdo es el mismo: producciones 1 y 2, el símbolo del lado izquierdo es el no terminal <A> y sus conjuntos de selección son disyuntos, cumple la condición; producciones 3 y 4, el símbolo del lado izquierdo es el no terminal <B> y sus conjuntos de selección son disyuntos; producciones 5 y 6, el símbolo del lado izquierdo es el no terminal <D> y sus conjuntos de selección son disyuntos; producciones 7 y 8, el símbolo del lado izquierdo es el no terminal <E> y sus conjuntos de selección son disyuntos. Por consiguiente, dicha gramática cumple las condiciones de la gramática LL(1). Dicha gramática es LL(1) y podremos construir el reconocedor descendente de pila para el lenguaje que genera.

8. Construcción del autómata de pila.
   Símbolos de entrada = {a, b, c, d, ┤}
   Símbolos en la pila = {<A>, <B>, <D>, <E>, c, ▼}
   Configuración inicial de la pila: ▼<A>
   Tabla de transiciones:

	a	b	c	d	┤
<A>	#2	#1	#1		
<B>		#3	#4	#4	#4
<D>			#6	#5	#6
<E>	#7	#8	#8	#8	#8
c			#0		
▼					**A**

#0: Desapile, avance.
#1: Replace(<D>c<B>), retenga.
#2: Replace(<E>), avance.
#3: Replace(c<A>), avance.
#4: Desapile, retenga.
#5: Replace(c<D><B>), avance.
#6: Desapile, retenga.
#7: Desapile, avance.
#8: Replace(<D><B>), retenga.

EJERCICIOS PROPUESTOS

1. Construya gramática **S** y su correspondiente reconocedor descendente de pila para cada uno de los siguientes lenguajes.

   a) 1^n20^n n >= 0.
   b) $w2w^r$ con w en (0+1)*.
   c) $1^n20^n + 21^n20^{2n}$ n > 0.
   d) $1^n20^n1^p20^p$ n > 0, p >= 0.
   e) 1^n0^n n > 0.

2. Construya gramática **Q** y su correspondiente reconocedor descendente de pila para cada uno de los siguientes lenguajes.

   a) 1^n n > 0.
   b) 1^n0^n n >= 0.
   c) 1^n0^m 0 < n <= m.

3. El siguiente autómata de pila se construyó usando el método definido en este módulo.

	a	b	c	d	┤
<S>	#1	#2	#2	#2	#2
<A>	#2	#3	#2	#2	#2
<B>	#4	#4	#5	#6	#4
▼	#4	#4	#4	#4	#7

   #1: Replace(<B><A>), avance.
   #2: Desapile, retenga.
   #3: Replace(<B><A><S>), avance.
   #4: Rechace.
   #5: Replace(<B>), avance.
   #6: Desapile, avance.
   #7: Acepte

   Construya la gramática utilizada en la construcción de dicho autómata y determine cuáles otras transiciones pueden ser también Rechace, sin afectar el lenguaje que está reconociendo.

4. Construya reconocedor descendente de pila para las siguientes gramáticas.

   1. <S> → a<A><B>
   2. <S> → b<B><S>
   3. <S> → λ
   4. <A> → c<B><S>
   5. <A> → λ
   6. <B> → d<B>
   7. <B> → e

1. <E> → <T><listaE>
2. <listaE> → +<T><listaE>
3. <listaE> → -<T><listaE>
4. <listaE> → λ
5. <T> → <P><listaT>
6. <listaT> → *<P><listaT>
7. <listaT> → /<P><listaT>
8. <listaT> → %<P><listaT>
9. <listaT> → λ
10. <P> → (<E>)
11. <P> → i

SOLUCIONES

1a. Construir gramática **S** y su correspondiente reconocedor descendente de pila para el siguiente lenguaje: 1^n20^n con n >= 0

1. <S> → 1<S>0
2. <S> → 2

Símbolos de entrada = {0, 1, 2, ┤}
Símbolos en la pila = {▼, <S>, 0}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	0	1	2	┤
<S>		#1	#2	
0	#3			
▼				A

#1: Replace(0<S>), avance.
#2: Desapile, avance.
#3: Desapile, avance.

1c. Construya gramática **S** y su correspondiente reconocedor descendente de pila para el siguiente lenguaje: $1^n20^n + 21^n20^{2n}$ con n > 0

1. <S> → 1<A>0
2. <S> → 21<B>00
3. <A> → 1<A>0
4. <A> → 2
5. <B> → 1<B>00
6. <B> → 2

Símbolos de entrada = {0, 1, 2, ┤}
Símbolos en la pila = {▼, <S>, <A>, <B>, 1, 0}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	0	1	2	┤
<S>		#1	#2	
<A>		#3	#4	
<B>		#5	#6	
0	#7			
1		#7		
▼				A

#1: Replace(0<A>), avance.
#2: Replace(00<B>1), avance.
#3: Replace(0<A>), avance.
#4: Desapile, avance.
#5: Replace(00<B>), avance.
#6: Desapile, avance.
#7: Desapile, avance.

1e. Construya gramática S y su correspondiente reconocedor descendente de pila para el siguiente lenguaje: 1^n0^n con $n > 0$

1. <S> → 1<A>
2. <A> → 1<A>0
3. <A> → 0

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {▼, <S>, <B>, 0}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	0	1	┤
<S>		#1	
<A>		#2	
<B>	#3		
0	#4		
▼			A

#1: Replace(<A>), avance.
#2: Replace(0<A>), avance.
#3: Desapile, avance.
#4: Desapile, avance.

2a. Construya gramática **Q** y su correspondiente reconocedor descendente de pila para el siguiente lenguaje: 1^n con $n > 0$

1. <S> → 1<A>
2. <A> → 1<A>
3. <A> → λ

Primeros(<S>) = {1}
Primeros(<A>) = {1}

Primeros(1) = {1}
Primeros(2) = {1}
Primeros(3) = {}

Siguientes(<S>) = {┤}
Siguientes(<A>) = {┤}

Selección(1) = {1}
Selección(2) = {1}
Selección(3) = {┤}

Símbolos de entrada = {1, ┤}
Símbolos en la pila = {▼, <S>, <A>}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	0	1	┤
<S>		#1	
<A>		#2	#3
▼			A

#1: Replace(<A>), avance.
#2: Avance.
#3: Desapile, retenga.

2c. Construya gramática **Q** y su correspondiente reconocedor descendente de pila para el siguiente lenguaje: 1^n0^m con m >= n > 0

1. <S> → 1<A>0<B>
2. <A> → 1<A>0
3. <A> → λ
4. <B> → 0<B>
5. <B> → λ

Primeros(<S>) = {1}
Primeros(<A>) = {1}
Primeros(<B>) = {0}

Primeros(1) = {1}
Primeros(2) = {1}
Primeros(3) = {}
Primeros(4) = {0}
Primeros(5) = {}

Siguientes(<S>) = {┤}
Siguientes(<A>) = {0}
Siguientes(<B>) = {┤}

Selección(1) = {1}
Selección(2) = {1}
Selección(3) = {0}
Selección(4) = {0}
Selección(5) = {┤}

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {<S>, <A>, <B>, 0, ▼}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	0	1	┤
<S>		#1	
<A>	#3	#2	
<B>	#4		#5
0	#0		
▼			A

#0: Desapile, avance.
#1: Replace(<B>0<A>), avance.

#2: Replace(0<A>), avance.
#3: Desapile, retenga.
#4: Avance.
#5: Desapile, retenga.

3. El siguiente autómata de pila se construyó usando el método definido en este módulo.

	a	b	c	d	┤
<S>	#1	#2	#2	#2	#2
<A>	#2	#3	#2	#2	#2
<B>	#4	#4	#5	#6	#4
▼	#4	#4	#4	#4	#7

#1: Replace(<B><A>), avance.
#2: Desapile, retenga.
#3: Replace(<B><A><S>), avance.
#4: Rechace.
#5: Replace(<B>), avance.
#6: Desapile, avance.
#7: Acepte

Construya la gramática utilizada en la construcción de dicho autómata y determine cuáles otras transiciones pueden ser también Rechace, sin afectar el lenguaje que está reconociendo.

1. <S> → a<A><B>
2. <S> → λ
3. <A> → b<S><A><B>
4. <A> → λ
5. <B> → c<B>
6. <B> → d

Para determinar cuáles transiciones pueden ser rechace sin afectar el reconocedor construimos el conjunto de selección de cada producción:

Selección(1) = {a}
Selección(2) = {b, c, d, ┤}
Selección(3) = {b}
Selección(4) = {c, d}
Selección(5) = {c}
Selección(6) = {d}

	A	b	c	d	┤
<S>	#1	#2	#2	#2	#2
<A>	#2	#3	#2	#2	#2
<B>	#4	#4	#5	#6	#4
▼	#4	#4	#4	#4	#7

Las transiciones resaltadas en amarillo pueden ser también RECHACE y no se afecta el reconocedor.

MODULO 9

GRAMATICAS DE TRADUCCION, MANEJO DE ERRORES Y RECONOCIMIENTO DESCENDENTE RECURSIVO

9.1 INTRODUCCIÓN

En el módulo anterior vimos lo que es el reconocimiento descendente y las clases de gramáticas sobre las cuales se aplica esta técnica. Consideraremos en este módulo el hecho de que se tengan gramáticas de traducción, el tratamiento que se le da a las diferentes situaciones de error que se puedan presentar y una técnica alterna para construir los programas que efectúan el proceso de reconocimiento.

9.2 OBJETIVOS

1. Reconocer cuándo una gramática de traducción es una gramática S, Q ó LL(1).
2. Determinar las transiciones cuando el símbolo en el tope de la pila sea un símbolo de acción.
3. Reconocer cuándo una gramática es una gramática LL(1).
4. Determinar las acciones a tomar y los mensajes a producir cuando se presentan situaciones de error durante el proceso de reconocimiento.
5. Construir los programas que efectúan el reconocimiento del lenguaje que genera una gramática usando técnica recursiva.

9.3 PREGUNTAS BÁSICAS

1. Qué implica los símbolos de acción en el tope de la pila.
2. Qué opciones tiene el constructor de un reconocedor cuando encuentra una situación de error.
3. Qué pasos se siguen para construir reconocedor descendente recursivo de una gramática.

9.4 Convención. En el módulo anterior definimos que las producciones podían ser de tres formas:

1. $<N> \rightarrow T\alpha$ con α en $(T+N)^*$.
2. $<N> \rightarrow \beta$ con β en $(T+N)^+$ tal que comience con un no terminal.
3. $<N> \rightarrow \lambda$

Como en este módulo trataremos el proceso de reconocimiento descendente de gramáticas de traducción vamos a redefinir las tres diferentes formas de las producciones, considerando los símbolos de acción. Las tres formas quedarán:

1. $<N> \rightarrow \gamma_1 T \gamma_2 \alpha$ con α en $(T+N)^*$.
2. $<N> \rightarrow \gamma\beta$ con β en $(T+N)^+$ tal que comience con un no terminal.
3. $<N> \rightarrow \gamma$

Siendo los γ hileras cualquiera de símbolos de acción, que pueden ser la secuencia nula.

Para construir el reconocedor ascendente de pila de gramáticas de traducción, la gramática debe cumplir exactamente las mismas condiciones que en el módulo anterior. Es decir, debe ser una gramática S, Q ó LL(1).

Para determinar si la gramática es S, Q ó LL(1) se sigue el mismo proceso que en el módulo anterior, es decir construir el conjunto de selección de cada producción si es necesario. Para construir dichos conjuntos debe hacer caso omiso de los símbolos de acción.

Tenga en cuenta: **los símbolos de acción no son símbolos de entrada, son tareas que se ejecutan cuando se está haciendo el proceso de reconocimiento.**

9.2 Construcción de autómata de pila para reconocimiento descendente de gramáticas de traducción.

A continuación presentamos los pasos para construir este autómata de pila. Presentamos el caso más general, cuando la gramática tenga producciones de las tres formas descritas.

Sea:

A: símbolo en el tope de la pila.
x: símbolo de entrada.

5. Símbolos de entrada: Los terminales **(T)** de la gramática más el símbolo de fin de secuencia **(┤)**.
6. Símbolos en la pila: los no terminales de la gramática **(N)**, todo terminal **(T)** y símbolo de acción que pertenezca a alguna hilera **α** ó **β,** y el símbolo de pila vacía (▼).
7. Configuración inicial de la pila: **▼S**, siendo **S** el símbolo inicial de la gramática.
8. Transiciones:

4.1 Existe una producción de la forma:

$$\mathbf{A \rightarrow \gamma_1 x \gamma_2 \alpha}$$

La transición es: **out($\gamma_1\gamma_2$), Replace(α^r), avance**

Si **α** es la secuencia nula (**λ**) la transición queda:
Out($\gamma_1\gamma_2$), Desapile, Avance.

4.2 Existe una producción de la forma:

$$\mathbf{A \rightarrow \gamma\beta}$$

y **x** pertenece al conjunto de selección de esa producción, la transición es:

Out(γ), Replace(β^r), retenga

4.3 Existe una producción de la forma:

$$\mathbf{A \rightarrow \gamma}$$

y **x** pertenece al conjunto de selección de esa producción, la transición

es:

out(γ), Desapile, retenga

4.4 **A** es un terminal **(T)**, entonces en la intersección de **A** con **x** (**A == x**) la transición es:

Desapile, Avance.

4.5 **A** es un símbolo de acción. Cualquiera que sea el símbolo de entrada la transición es:

out(γ), desapile, retenga.

4.6 **A** es ▼, y **x** es ┤, entonces la transición es: **Acepte**.

4.7 Cualquier otra situación es: **Rechace**.

9.5 Ejemplo

Consideremos la siguiente gramática con el fin de ilustrar el proceso dado.

1. <A> → {x}{y}a{z}<B>a<C>
2. <A> → b{y}<A>c{z}
3. <B> → {x}
4. <B> → {x}<C>a{y}<B>a
5. <C> → c
6. <C> → b<A>{y}c

Nuestro primer paso es determinar qué clase de gramática es: no es una gramática S ni Q porque tiene producciones cuyo lado derecho es la secuencia nula (producción 3) y porque tiene producciones cuyo lado derecho comienza con un no terminal (producción 4).

Por consiguiente, debemos determinar si la gramática es LL(1). Para ello debemos construir el conjunto de selección de cada producción.

Tenga en cuenta que si los símbolos de acción lo enredan en el proceso de construcción de el conjunto de selección de cada producción usted puede eliminar temporalmente los símbolos de acción de la gramática y trabajar con la gramática resultante con el fin de construir los conjuntos de selección. Si hacemos este "truco", la gramática es:

1. <A> → a<B>a<C>
2. <A> → b<A>c
3. <B> → λ
4. <B> → <C>a<B>a
5. <C> → c
6. <C> → b<A>c

y quizá, sea más sencillo trabajar con ella

Teniendo esta gramática, los conjuntos de selección son:

Selección(1) = {a}
Selección(2) = {b}
Selección(3) = {a}
Selección(4) = {b, c}
Selección(5) = {c}
Selección(6) = {b}

Por tanto, nuestra gramática es una gramática LL(1) y el autómata de pila que reconoce el lenguaje generado por ella y efectúa la traducción es:

1. Símbolos de entrada = {a, b, c, ┤}
2. Símbolos en la pila = {<A>, <B>, <C>, a, c, {y}, {z}}
3. Configuración inicial de la pila: ▼<A>
4. Transiciones: A continuación presentamos la tabla de transiciones correspondiente.

	a	b	c	┤
<A>	#1	#2		
<B>	#3	#4	#4	
<C>		#6	#5	
a	#0			
c			#0	
▼				**A**
{y}	out(y), desapile, retenga			
{z}	out(z), desapile, retenga			

#0: Desapile, avance.
#1: Out(xyz), Replace(<C>a<B>), avance.
#2: Out(y), Replace({z}c<A>), avance.
#3: Out(x), desapile, retenga.
#4: Out(x), Replace(a<B>{y}a<C>), retenga.
#5: Desapile, avance
#6: Replace(c{y}<A>), avance
A: Acepte

9.6 Manejo de errores en reconocimiento descendente.

Hasta ahora hemos construido autómatas de pila que reconocen el lenguaje que genera una gramática. Hemos dejado en blanco, en las tablas de transiciones, las situaciones en las cuales se rechaza una hilera cuando se está haciendo un proceso de reconocimiento. Vamos a ocuparnos de estos casos.

Cuando se encuentra un error en una hilera que se está reconociendo debemos tomar algunas acciones: primero, producir un mensaje que oriente al usuario acerca del error que cometió, con el fin de que lo pueda arreglar fácilmente y segundo, optar por detener el proceso de reconocimiento o continuar el procesamiento como si no hubiera ocurrido ningún error.

Para producir un mensaje de error apropiado es supremamente importante el conocimiento del lenguaje que se está generando.

Veamos ahora algo correspondiente a la segunda acción.

Si opta por detener el proceso de reconocimiento, implica que el usuario debe hacer la corrección y luego volver a ejecutar el proceso de reconocimiento. Un lenguaje como el Pascal, funcionaba de esta forma: los mensajes de error eran claros, pero el proceso de dejar a punto un programa para comenzar a probarlo era dispendioso e ineficiente.

Si opta por lo segundo, el reconocedor deberá hacer algún reacomodo en la pila para continuar el reconocimiento como si no hubiese ocurrido ningún error. A pesar de que hay algunas técnicas para decidir cuál reacomodo es el mejor, esto del reacomodo es criterio del constructor del reconocedor. Cuando se hace un reacomodo en la pila, para continuar el proceso de reconocimiento como si no hubiese ocurrido ningún error, puede suceder que el reacomodo no sea el más apropiado para el error que ocurrió, lo cual implica que más adelante se presente alguna situación de error que realmente no existe. El lenguaje C++ funciona de esta manera. Para los que han programado en C++ es muy factible que les haya sucedido que en la línea 16 del programa les reporte un error, y que luego, por allá en la línea 80 produzca otro mensaje de error. El programador corrige el error de la línea 16, pero no encuentra el error de la línea 80. Sin embargo, el programador vuelve a compilar su programa habiendo corregido el error de la línea 16, y el error de la línea 80 desapareció como por arte de magia. El error que apareció en la línea 80 se produjo porque el reacomodo que se hizo cuando encontró el error de la línea 16 no fue el más apropiado.

Consideremos la siguiente gramática para ilustrar el manejo de errores en una gramática.

1. <A> → a
2. <A> → (<A><B>
3. <B> → ,<A><B>
4. <B> →)

Como se puede observar esta es una gramática S: el lado derecho de cada producción comienza con un terminal, y producciones cuyo símbolo del lado izquierdo es el mismo (producciones 1y 2, y producciones 3 y 4) sus conjuntos de selección son disyuntos.

El autómata de pila con el cual se efectúa el reconocimiento descendente del lenguaje que genera dicha gramática es:

1. Símbolos de entrada = {(, a, ",",), ┤}
2. Símbolos en la pila = {<A>, <B>, ▼}
3. Configuración inicial de la pila: ▼<A>
4. Tabla de transiciones:

	(	a	,	)	┤
<A>	#2	#1	R_a	R_b	R_c
<B>	R_d	R_e	#3	#4	R_f
▼	R_g	R_h	R_i	R_j	**A**

#1: Desapile, avance.
#2: Replace(<B><A>), avance.
#3: Replace(<B><A>), avance.
#4: Desapile, avance.
A: Acepte.

Las situaciones de error las hemos denominado R_a, R_b, R_c, R_d, R_e, R_f, R_g, R_h, R_i, y R_j.

Podremos optar por tratar cada situación de error de una manera individual, sin embargo hay ciertas situaciones de error que se pueden agrupar en una sola, debido a que el mensaje de error a producir es básicamente el mismo y que ambas situaciones de error se resuelven utilizando un mismo símbolo en el tope de la pila.

En nuestro ejemplo podremos agrupar R_a con R_b, R_c con R_f, R_d con R_e, R_g con R_h y R_i con R_j.

Recordemos, que con el propósito de producir buenos mensajes de error, es importante conocer bien el lenguaje que se está reconociendo, es decir, el lenguaje que genera la gramática cuyo reconocedor se está construyendo.

Ejemplos de hileras generadas por nuestra gramática son:

a
(a)
(a,a,a,a)
((a,a),a,a,(a,a,a,(a,a),a),a,a)

Hileras de ese estilo se conocen como listas generalizadas, o expresiones S en LISP.

Comencemos considerando las situaciones R_a y R_b. Agrupamos estas dos situaciones en una sola, la cual llamaremos R_{ab}. En el tope de la pila se halla el no terminal <A>, lo cual implica que el autómata de pila está esperando que el símbolo de entrada sea una "a" o un abre paréntesis, pero resulta que el símbolo de entrada fue una coma o un cierre paréntesis. El mensaje a producir, perfectamente puede ser: **write**("se esperaba una a o un abre paréntesis, y el símbolo que llegó fue", símbolo), siendo símbolo una variable en la cual se halla el símbolo leído. Ahora, para continuar el proceso como si no hubiera ocurrido ningún error debemos hacer algún reacomodo en la pila. Las opciones son:

Avance: ignora el símbolo leído, con la esperanza de que el siguiente símbolo se pueda reconocer con el no terminal <A>;
Apilar(<A>), retenga: lleva a la pila el no terminal <A> y retiene el símbolo de entrada, o,
Desapile, retenga: elimina el símbolo en el tope de la pila y retiene el símbolo de entrada, con la esperanza de que en el tope de la pila quede un no terminal <A>, el cual es el símbolo que reconoce tanto a la coma como al cierre paréntesis.

Las situaciones R_c y R_f las agrupamos en una sola, la cual llamamos R_{cf}. Aquí lo único a efectuar es producir un mensaje lo más apropiado posible y terminar el proceso de reconocimiento puesto que llegó el símbolo de fin de secuencia indicando que no hay más símbolos a reconocer. Un mensaje apropiado podría ser: "Expresión S vacía o incompleta".

Las situaciones R_d y R_e las agrupamos en una sola, la cual llamamos R_{de}. En el tope de la pila se halla el no terminal <B>, lo cual significa que el autómata de pila está esperando una coma o un cierre paréntesis, pero lo que apareció fue una "a" o un abre paréntesis. Ejemplos de estas situaciones son:

(a,a,**a a**,a,a)
(a,**a (**a,a),a)

En ambos casos, es evidente que falta una coma, por consiguiente El mensaje apropiado es: write("falta coma"). En cuanto al reacomodo en la pila, tenemos tres opciones:

Avance: ignora el símbolo leído, con la esperanza de que el siguiente símbolo se pueda reconocer con el no terminal <B>;

Apilar(<B>), retenga: lleva a la pila el no terminal <B> y retiene el símbolo de entrada, o,

Desapile, retenga: elimina el símbolo en el tope de la pila y retiene el símbolo de entrada, con la esperanza de que en el tope de la pila quede un no terminal <A>, el cual es el símbolo que reconoce tanto a la "a" como al abre paréntesis.

En las situaciones R_g, R_h, R_i y R_j, en el tope de la pila se halla el símbolo de pila vacía **(▼)**, lo cual significa que se ha efectuado un proceso de reconocimiento exitoso, por tanto el autómata de pila está esperando que el símbolo de entrada sea el símbolo de fin de secuencia **(┤)**, pero apareció un símbolo diferente. En cualquiera de las cuatro situaciones el mensaje perfectamente puede ser: **write**("más símbolos después de una expresión S", símbolo).

Si la situación fuera R_{gh}, el reacomodo en la pila podría ser:

Avance: ignora el símbolo leído, con la esperanza de que el siguiente símbolo sea el símbolo de fin de secuencia, o,

Apilar(<A>), retenga: lleva a la pila el no terminal <A> y retiene el símbolo de entrada.

Si la situación fuera R_{ij}, el reacomodo en la pila puede ser:

Avance: ignora el símbolo leído, con la esperanza de que el siguiente símbolo sea el símbolo de fin de secuencia, o,

Apilar(<B>), retenga: lleva a la pila el no terminal <B> y retiene el símbolo de entrada.

9.7 Reconocimiento descendente recursivo.

Recordemos que nuestro proceso de reconocimiento consiste en construir un autómata de pila, con base en el cual, elaboramos un programa de computador que reconocerá el lenguaje correspondiente a una gramática dada.

Es posible elaborar directamente los programas que reconocen el lenguaje que genera una gramática, con base en la gramática, siempre y cuando, ésta cumpla las características de que sea LL(1).

Para ilustrar cómo hacer esta tarea, consideremos nuevamente la gramática del numeral 3 del módulo anterior.

9. <A> → <B>c<D> Selección(1) = {b, c}

10. <A> → a<E> Selección(2) = {a}
11. <B> → b<A>c Selección(3) = {b}
12. <B> → λ Selección(4) = {c, d, ┤}
13. <D> → d<B><D>c Selección(5) = {d}
14. <D> → λ Selección(6) = {c, ┤}
15. <E> → a Selección(7) = {a}
16. <E> → <B><D> Selección(8) = {b, c, d, ┤}

Nuestro objetivo es construir directamente los programas con los cuales se haga el reconocimiento del lenguaje que genera dicha gramática.

La forma de hacer esto es: Definir un programa principal, utilizar una variable global con la cual leemos la hilera a reconocer y definir un subprograma, tipo void, por cada no terminal que tenga la gramática. Cada subprograma debe dejar un símbolo leído.

La forma general del programa principal es:

```
lea(símbolo)
nta()
if  símbolo == "┤"  then
        Acepte
else
        Rechace
end(if)
```

El primer símbolo que lea el programa principal tuvo que haber sido generado por la aplicación de alguna de las producciones que definen el símbolo inicial de la gramática, por consiguiente, invocamos el subprograma correspondiente a dicho no terminal (en nuestro ejemplo **nta**). El subprograma nta() deja un símbolo leído, el cual, debe ser el símbolo de fin de secuencia **(┤)** para que la secuencia sea válida.

Veamos cómo sería cada uno de los subprogramas.

```
void  nta()
        casos de símbolo
                "b", "c":
                        ntb()
                        if  símbolo == "c"  then
                                lea(símbolo)
                                ntd()
                                return
                        end(if)
                        Rechace
                "a":
                        lea(símbolo)
                        nte()
                        return
                "else":
                        Rechace
        fin(casos)
fin(nta)
```

El subprograma nta() corresponde a reconocer las producciones que definen el no terminal <A>. En nuestro ejemplo hay dos producciones que definen al no terminal <A>, dichas producciones son la 1 y la 2. Cuando tenemos el autómata de pila sabemos que cuando en el tope de la pila se halle el no terminal <A> fue porque se aplicó la producción 1 ó 2. Con base en los conjuntos de selección que hemos construido, determinamos que cuando el símbolo de entrada sea una "b" o un "c", la que se aplicó fue la producción 1, y que cuando el símbolo de entrada sea una "a" la que se aplicó fue la producción 2, cualquier otro que sea el símbolo de entrada es una situación de error.
Estas situaciones las plasmamos en nuestro algoritmo con la instrucción "casos".

Cuando el símbolo leído sea una "b" o una "c" ejecutamos lo correspondiente al lado derecho de la producción 1: invocamos el subprograma correspondiente al no terminal <B>, el cual deja un símbolo leído que debe ser el terminal "c"; controlamos que el símbolo leído sea efectivamente una "c", en cuyo caso leemos el siguiente símbolo e invocamos el subprograma ntd(), el cual debe reconocer el nuevo símbolo leído. En este punto hemos llegado al final del lado derecho de la producción 1, por tanto, regresamos a continuar con el proceso de reconocimiento al programa llamante.

Cuando el símbolo leído sea una "a" significa que se aplicó fue la producción 2, por tanto, ejecutaremos las instrucciones correspondientes al lado derecho de la producción 2: leemos otro símbolo e invocamos el subprograma correspondiente al no terminal <E>, es decir, nte().

Si el símbolo leído es diferente a "a", "b" ó "c" es una situación de error, por tanto nuestro subprograma nta() lo rechaza.

```
void  ntb()
      casos de símbolo
            "b":
                  lea(símbolo)
                  nta()
                  if  símbolo == "c"  then
                        lea(símbolo)
                        return
                  end(if)
                  Rechace
            "c", "d", "┤":
                  return
            "else":
                  Rechace
      fin(casos)
fin(ntb)
```

El subprograma ntb() es correspondiente a cuando en el tope de la pila se halle el no terminal <B>, cuando se está reconociendo con un autómata de pila. Dichas situaciones se reflejan en nuestro algoritmo.

Si el símbolo de entrada es una "b" significa que se aplicó la producción 3, por consiguiente las instrucciones correspondientes a esa producción son: leer un nuevo símbolo, invocar el subprograma nta() y comprobar que el símbolo que dejó leído el

subprograma nta() sea una "c", para proceder a leer otro símbolo y retornar al programa llamante.

Si el símbolo de entrada es una "c" o una "d", es decir, alguno de los símbolos correspondientes al conjunto de selección de la producción 4, significa que se aplicó la producción 4, y por tanto, efectuamos las operaciones correspondientes al lado derecho de esa producción, que como es la secuencia nula, simplemente es la instrucción return.

```
void  ntd()
      casos de símbolo
            "d":
                  lea(símbolo)
                  ntb()
                  ntd()
                  if  simbolo == "c"  then
                        lea(símbolo)
                        return
                  end(if)
                  Rechace
            "c", "┤":
                  return
            "else":
                  Rechace
      fin(casos)
fin(ntd)

void  nte()
      casos de símbolo
            "a":
                  lea(símbolo)
                  return
            "b", "c", "d", "┤":
                  ntb()
                  ntd()
                  return
            "else":
                  Rechace
      fin(casos)
fin(nte)
```

Dejamos al estudiante para que compruebe la correctitud de los subprogramas ntd() y nte().

En general, el lado derecho de cada producción puede ser de cada una de las siguientes formas: comenzar con un terminal, comenzar con un no terminal o ser la secuencia nula.

Una producción de la forma a<B><C>d<E>, es decir, cuyo lado derecho comienza con un terminal, las instrucciones correspondientes a dicho lado derecho son:

```
lea(símbolo)
ntb()
```

```
ntc()
if  simbolo == "d"  then
        lea(símbolo)
        nte()
end(if)
```

Una producción, cuyo lado derecho sea de la forma <A><B>c<D>, es decir, cuyo lado derecho comience con un no terminal, las instrucciones correspondientes a reconocer ese lado derecho son:

```
nta()
ntb()
if  símbolo == "c"  then
        lea(símbolo)
        ntd()
end(if)
```

Una producción cuyo lado derecho sea la secuencia nula, la instrucción correspondiente es simplemente return.

EJERCICIOS PROPUESTOS

1. Construya reconocedor descendente recursivo para las siguientes gramáticas:

a)

8. <S> → a<A><B>
9. <S> → b<B><S>
10. <S> → λ
11. <A> → c<B><S>
12. <A> → λ
13. <B> → d<B>
14. <B> → e

b)

12. <E> → <T><listaE>
13. <listaE> → +<T><listaE>
14. <listaE> → λ
15. <T> → <P><listaT>
16. <listaT> → *<P><listaT>
17. <listaT> → λ
18. <P> → (<E>)
19. <P> → i

2. Construya reconocedor descendente de pila para las siguientes gramáticas:

a)

1. <S> → a{x}<B>{h}{a}c
2. <S> → {a}b{a}
3. <B> → c{x}{y}<B><B>{e}
4. <B> → a{b}{c}d{e}f{y}

b)

1. <S> → 3{tres}<ER><S>{void}
2. <S> → {y}{z}4
3. <ER> → λ
4. <ER> → 2{dos}<S><ER>

3. Construya reconocedor descendente de pila para la gramática b del ejercicio 1. Incluya mensajes de error y tratamiento a cada una de las situaciones de error que se presenten.

SOLUCIONES

1a. Construya reconocedor descendente recursivo para las siguientes gramáticas:

1. <S> → a<A><B>
2. <S> → b<B><S>
3. <S> → λ
4. <A> → c<B><S>
5. <A> → λ
6. <B> → d<B>
7. <B> → e

Solución

El primer paso es construir el conjunto de selección de cada producción. El conjunto de selección para las producciones 1, 2, 4, 6 y 7 es simplemente el terminal con el cual comienza su lado derecho. El conjunto de selección de la producción 5 son los siguientes de <A>, los cuales son los primeros de <B> (d, e), en virtud de la producción 1. El conjunto de selección de la producción 3 son los siguientes de <S>, los cuales son los siguientes de <A> (por producción 4) y el símbolo de fin de secuencia, por ser <S> el símbolo inicial de la gramática.

Sintetizando tenemos:

Selección(1) = {a}
Selección(2) = {b}
Selección(3) = {d, e, ┤}
Selección(4) = {c}
Selección(5) = {d, e}
Selección(6) = {d}
Selección(7) = {e}

Como se puede observar, la gramática es LL(1). El conjunto de programas con los que se efectúa el reconocimiento descendente recursivo son:

Programa principal

```
        lea(símbolo)
        NTS()
        if   símbolo  ==  "┤"  then
                Acepte
        else
                Rechace
        end(if)
fin(programa principal)
```

void NTS()

```
        casos de símbolo
                "a":   lea(símbolo)
                       NTA()
                       NTB()
                       return
                "b":   lea(símbolo)
                       NTB()
```

```
                    NTS()
                    return
            "d", "e", "┤": return
            "else":  Rechace
        fin(casos)
fin(NTS)

void  NTA()
        casos de símbolo
            "c":   lea(símbolo)
                    NTB()
                    NTS()
                    return
            "d", "e":  return
            "else":  Rechace
        fin(casos)
fin(NTA)

void  NTB()
        casos de símbolo
            "d":   lea(símbolo)
                    NTB()
                    return
            "e":   lea(símbolo)
                    return
            "else":  Rechace
        fin(casos)
fin(NTB)
```

2a. Construya reconocedor descendente de pila para la siguiente gramática:

1. <S> → a{x}<B>{h}{a}c
2. <S> → {a}b{a}
3. <B> → c{x}{y}<B><B>{e}
4. <B> → a{b}{c}d{e}f{y}

Solución:

Como se puede observar la gramática es S. Por consiguiente, el conjunto de selección de cada producción es el terminal con el cual comienza su lado derecho: Selección(1) = {a}, Selección(2) = {b}, Selección(3) = {c} y Selección(4) = {a}. Recuerde que los símbolos de acción no son símbolos de entrada para la gramática, son tareas que se ejecutan cuando se hace un proceso de reconocimiento.

Símbolos de entrada = {a, b, c, d, f, ┤}
Símbolos en la pila = {<S>, <B>, c, d, f, ▼, {a}, {e}, {h}, {y}}
Configuración inicial de la pila = ▼<S>
Tabla de transiciones:

	a	b	c	d	f	┤
<S>	#1	#2				
<B>	#4		#3			
c			#0			
d				#0		
f					#0	
▼						**A**
{a}	out(a), desapile, retenga					
{e}	out(e), desapile, retenga					
{h}	out(h), desapile, retenga					
{y}	out(y), desapile, retenga					

#0: desapile, avance.
#1: out(x), replace(c{a}{h}<B>), avance.
#2: out(aa), desapile, avance.
#3: out(xy), replace(c<B><B>), avance.
#4: out(bc), replace({y}f{e}d), avance.

3. Construya reconocedor descendente de pila para la gramática b del ejercicio 1. Incluya mensajes de error y tratamiento a cada una de las situaciones de error que se presenten.

1. <E> → <T><listaE>
2. <listaE> → +<T><listaE>
3. <listaE> → λ
4. <T> → <P><listaT>
5. <listaT> → *<P><listaT>
6. <listaT> → λ
7. <P> → (<E>)
8. <P> → i

Solución:

Construimos primero el conjunto de selección de cada producción.

Selección(1) = {(, i}
Selección(2) = {+}
Selección(3) = {), ┤}
Selección(4) = {(, i}
Selección(5) = {*}
Selección(6) = {+,), ┤}
Selección(7) = {(}
Selección(8) = {i}

La gramática es LL(1).

Símbolos de entrada = {+, *, (, i,), ┤}
Símbolos en la pila = {<E>, <listaE>, <T>, listaT>, <P>,), ▼}
Configuración inicial de la pila = ▼<E>
Transiciones:

	+	*	(	i	)	┤
<E>	R_a	R_b	#1	#1	R_c	R_d
<listaE>	#2	R_e	R_f	R_g	#3	#3
<T>	R_h	R_i	#4	#4	R_j	R_k
<listaT>	#6	#5	R_l	R_m	#6	#6
<P>	R_n	R_o	#7	#8	R_p	R_q
)	R_s	R_t	R_u	R_v	#0	R_w
▼	R_x	R_y	R_z	R_0	R_1	**A**

#0: desapile, avance.
#1: replace(<listaE><T>), retenga
#2: replace(<listaE><T>), avance.
#3: desapile, retenga.
#4: replace(<listaT><P>), retenga.
#5: replace(<listaT><P>), avance.
#6: desapile, retenga.
#7: replace()<E>), avance.
#8: desapile, avance.
R_a: write("Símbolo + no permitido"), replace(<listaE><T>), retenga.
R_b: write("Símbolo * no permitido"), replace(<listaT><P>), retenga.
R_{cjp}: write("Cierre paréntesis inválido"), avance.
R_d: write("Expresión nula o incomplete"), termine.
R_e: write("Símbolo * no permitido"), replace(<listaT><P>), retenga.
R_{fg}: write("Abre paréntesis o identificador no permitido"), replace(<E>), retenga.
R_{hns}: write("Operador + no permitido"), replace(<listaE>), retenga.
R_{iot}: write("Operador * no permitido"), replace(<listaT><P>), retenga.
R_{kqw}: write("Expresión incompleta"), termine.
R_{lmuv}: write("Abre paréntesis o identificador no permitido"), replace(<listaT><P>), retenga.
R_{xyz01}: write("Más símbolos después de expresión válida"), avance.

MODULO 10
PROCESAMIENTO DESCENDENTE DE GRAMÁTICAS CON ATRIBUTOS

10.1 INTRODUCCIÓN
Hemos estado tratando el proceso de reconocimiento descendente sólo en gramáticas generativas y de traducción. Veremos en este módulo cómo es el proceso cuando la gramática tiene atributos.

10.2 OBJETIVOS
1. Construir reconocedores descendentes considerando la parte de atributos.
2. Construir gramáticas de traducción con atributos tal que se les pueda procesar descendentemente.

10.3 PREGUNTAS BÁSICAS
1. Qué es una gramática L.
2. Qué restricción tiene la evaluación de atributos sintéticos.
3. Qué restricción tiene la evaluación de atributos heredados.
4. Qué restricción tiene la evaluación de atributos sintéticos de símbolos de acción.
5. Qué condiciones debe cumplir el conjunto de reglas de evaluación de atributos de una producción.

10.1 Gramáticas L.
Para poder efectuar procesamiento descendente de gramática con atributos se requiere que la gramática cumpla ciertas condiciones: que la gramática sea LL(1), que sea L en cuanto a evaluación de atributos y que las reglas de evaluación de atributos estén en la forma de asignación simple. Ya hemos visto cómo identificar si una gramática es LL(1), veremos ahora cómo determinar si una gramática es L y si las reglas de evaluación de atributos están en la forma de **asignación simple**.

Para que una gramática sea L, en cuanto a la evaluación de atributos, debe cumplir las siguientes condiciones:

1. Toda regla de evaluación de atributos heredados de símbolos del lado derecho de una producción debe estar en función de dos tipos de atributos: atributos heredados del símbolo del lado izquierdo de la producción y/o de cualquier atributo de símbolos del lado derecho que estén antes del símbolo cuyo atributo se está evaluando.

2. Toda regla de evaluación de atributos sintéticos del símbolo del lado izquierdo de una producción puede estar en función de cualquier atributo de símbolos del lado derecho y de atributos heredados del mismo símbolo del lado izquierdo.

3. Toda regla de evaluación de atributos sintéticos de símbolos de acción debe estar en función de atributos heredados del mismo símbolo de acción.

Veamos un ejemplo para ilustrar dichas condiciones. Sea la producción:

$$<A>_{p,q,r} \rightarrow <B>_{s,t}<C>_{u,v}<D>_{w,x}<E>_{m,z}\{f\}_{a,b,c}$$

Donde **p**, **q**, **s**, **u**, **w**, **m** y **c** son atributos sintéticos y

r, **t**, **v**, **x**, **z**, **a** y **b** son atributos heredados.

La regla de evaluación del atributo **x**, el cual es un atributo del símbolo <D>, puede estar en función solamente de **r**, **s**, **t**, **u** y **v**, en virtud de la primera condición.

Simbólicamente: x = f(r,s,t,u,v)

La regla de evaluación del atributo **x** no puede estar en función de **p** y **q** puesto que éstos son atributos sintéticos del símbolo del lado izquierdo; no puede estar en función de los atributos **m**, **z**, **a**, **b** y **c** puesto que éstos son atributos de símbolos del lado derecho de símbolos que está a continuación de <D> en el lado derecho de la producción, y no puede estar en función de **w** puesto que éste es un atributo sintético del mismo símbolo <D>.

La regla de evaluación del atributo **p**, el cual es un atributo sintético del símbolo del lado izquierdo puede estar en función de cualquier atributo, excepto de **q**, ya que éste es otro atributo sintético del mismo símbolo del lado izquierdo. Lo anterior en virtud de la condición 2.
La regla de evaluación del atributo c sólo puede estar en función de los atributos a y b, ya que éstos son los atributos heredados del mismo símbolo de acción. Lo anterior en virtud de la condición 3.

Veamos ahora qué significa que las reglas de evaluación de atributos estén en la forma de **asignación simple**.

Para que el conjunto de reglas de evaluación de atributos de una producción esté en la forma de asignación simple se deben cumplir dos condiciones: una, que las reglas sean **reglas de copiado** y dos, que el conjunto de reglas de evaluación de atributos de una producción sea **independiente**.

En una regla de evaluación de atributos el lado izquierdo se denomina sumidero y el lado derecho de denomina fuente. Si tenemos la regla **a = b+c**, **a** es el **sumidero** y **b+c** es la **fuente**.

Una regla es de copiado si la fuente es sólo una variable. Por consiguiente, la regla a = b+c no es una regla de copiado. Las únicas reglas, que se admite que no sean de copiado, son las reglas de evaluación de atributos sintéticos de símbolos de acción.

En general, una regla es de copiado si es de alguna de las siguientes formas:

$$x = y \quad \text{ó} \quad (x_1, x_2, \ldots, x_n) = y$$

El conjunto de reglas de una producción es independiente, sí y sólo sí, la fuente de cada regla no aparece en ninguna otra parte en cualquiera de las otras reglas del conjunto.

Si se tienen las reglas: **(x, y) = z** y **(a, b) = y**, éstas se pueden consolidar en una sola regla así: **(x, y, a, b) = z**

Dada una producción que tenga reglas de evaluación de atributos en las cuales haya alguna que no sea de copiado, la podemos transformar de tal manera que cumpla la condición de que las reglas sean de copiado. Consideremos este ejemplo:

$$<A> \rightarrow a_r<B>_s<C>_i$$
$$i = f(r, s)$$

siendo **s** un atributo sintético y **r** e **i** atributos heredados.

La anterior regla de evaluación de atributos no es una regla de copiado, pero podemos convertir la producción de tal manera que las reglas sean de copiado. Para ello, incluimos un símbolo de acción, el cual tendrá tres atributos: a, b y c. Los atributos a y b serán datos de entrada al símbolo de acción (atributos heredados) y c es el atributo (atributo sintético) en el cual se retornará el resultado de evaluar la función. Teniendo incluido este símbolo de acción ($\mathbf{\{f\}_{a,b,c}}$) las reglas de evaluación de atributos serán: a = r; b = s; i = c, las cuales son todas reglas de copiado. Lo que se sigue, es determinar en cuál sitio del lado derecho de la producción se debe colocar el símbolo de acción. Consideremos las diferentes posibilidades.

$$<A> \rightarrow \mathbf{\{f\}_{a,b,c}} a_r<B>_s<C>_i \quad (1)$$
$$<A> \rightarrow a_r\mathbf{\{f\}_{a,b,c}}<B>_s<C>_i \quad (2)$$
$$<A> \rightarrow a_r<B>_s\mathbf{\{f\}_{a,b,c}}<C>_i \quad (3)$$
$$<A> \rightarrow a_r<B>_s<C>_i\mathbf{\{f\}_{a,b,c}} \quad (4)$$

Recuerde, que cualquiera que sea el sitio en el ubiquemos el símbolo de acción, las reglas de evaluación de atributos son:

$$a = r; \quad b = s; \quad i = c$$

Si elegimos la opción (1), estaríamos violando la primera condición para que la gramática sea L, ya que la regla a=r evalúa un atributo heredado (el atributo a) en función de un atributo que pertenece a un símbolo que está después del símbolo {f} en el lado derecho de la producción, por tanto, la opción (1) no es aceptable.

La opción (2) también viola la primera condición para que la gramática sea L, en virtud de la regla b=s.

La opción (4) también viola la primera condición para que la gramática sea L debido a la regla i=c.

Por consiguiente, la única opción posible es la (3). Nuestra producción queda:

$$<A> \rightarrow a_r<B>_s\mathbf{\{f\}_{a,b,c}}<C>_i$$
$$a = r; \quad b = s; \quad i = c$$

En nuestro ejemplo solo hubo un sitio apropiado para ubicar el símbolo de acción, sin embargo, habrá situaciones en las cuales el símbolo de acción se puede colocar en más de un sitio, de tal manera que cumpla las propiedades para que la gramática sea L en cuanto a evaluación de atributos. En estos casos cualquiera que sea el sitio en el que se ubique el símbolo de acción es válido.

10.2 Un ejemplo

Teniendo definidas las condiciones para poder efectuar procesamiento descendente presentemos un ejemplo con el cual ilustraremos lo anteriormente descrito.

Consideremos la siguiente gramática con la cual se generan, traducen y evalúan expresiones en prefijo.

1. <S> → $<E>_p\{Resultado\}_q$
2. $<E>_p$ → $+<E>_q<E>_r\{Sume\}_{s,t,u}$
3. $<E>_p$ → $*<E>_q<E>_r\{Multiplique\}_{s,t,u}$
4. $<E>_p$ → I_q

Como se podrá observar la gramática es una gramática S. Los conjuntos de selección son: de la producción 1 son los símbolos +, *, I; de la producción 2 es el símbolo +; de la producción 3 es el símbolo *, y de la producción 4 es el símbolo I.

El autómata de pila es el siguiente:
Símbolos de entrada = {+, *, I, ┤}.
Símbolos en la pila = {<S>, <E>, {Resultado}, {Sume}, {Multiplique}, ▼}.
Configuración inicial de la pila: ▼<S>.
Transiciones:

	+	*	I	┤
<S>	#1	#1	#1	
<E>	#2	#3	#4	
▼				A
{Sume}	#5			
{Multiplique}	#6			
{Resultado}	#7			

#1: Reemplace en la pila como en la figura 1, retenga.
#2: Reemplace en la pila como en la figura 2, avance.
#3: Reemplace en la pila como en la figura 3, avance.
#4: Lleve el valor del atributo del terminal I al sitio indicado por el atributo del no terminal <E> que se halla en el tope de la pila, desapile y avance.
#5: Sume el contenido de los dos primeros atributos y lleve el resultado al sitio indicado por el tercer atributo, desapile y retenga.
#6: Multiplique el contenido de los dos primeros atributos y lleve el resultado al sitio indicado por el tercer atributo, desapile y retenga.
#7: Muestre el resultado que se halla en el atributo del símbolo de acción, desapile y retenga.

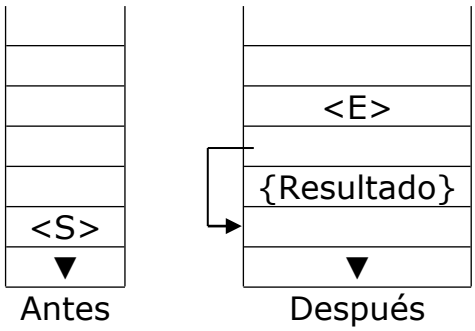
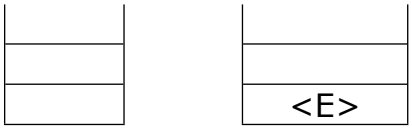


Figura 1



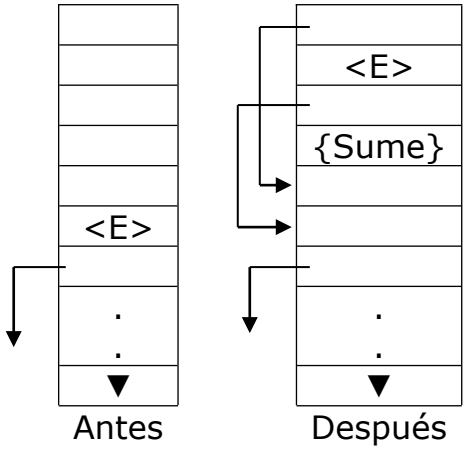


Figura 2

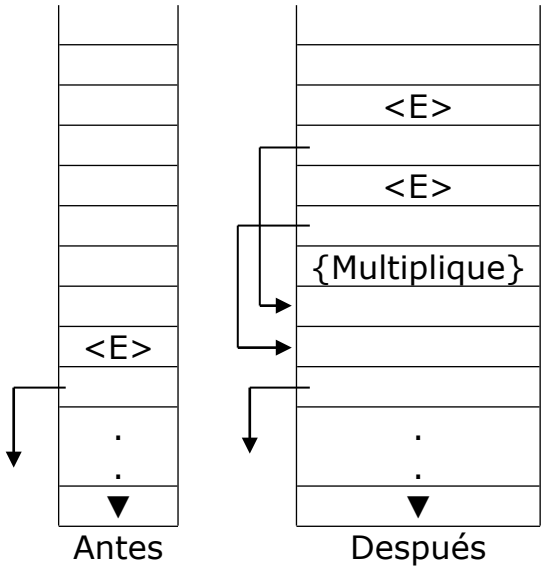


Figura 3

Mostremos ahora, cómo sería el proceso cuando se tenga la expresión **+*abc**┤ sabiendo que **a** = 5, **b =** 7 y **c =** 3.

Cuando se va a reconocer la expresión **+*abc**┤ lo primero que hace es tokenizar la expresión, convirtiéndola en una forma apropiada para que el analizador sintáctico la traduzca y la evalúe. La expresión queda:

$$+*I_5I_7I_3\dashv$$

Habiendo efectuado este proceso, continúa con el autómata de pila, el cual procede así: inicialmente la pila está como en el caso (a) de la figura 4, el símbolo de entrada es un más (+), por consiguiente, aplica la transición 1 definida en el autómata de pila, quedando la pila como en el caso (b) de la figura 4, y el símbolo de entrada se retiene; en el caso (b) de la figura 4, en el tope de la pila quedó el no terminal <E> y el símbolo de entrada es el más (+), por consiguiente, aplicamos la transición 2 definida en el autómata de pila quedando ésta como en el caso (c) de la figura 4 y se lee el siguiente símbolo de entrada; en el caso (c) de la figura 4 el símbolo en el tope de la pila es el no terminal <E> y el símbolo de entrada es el asterisco (*), por consiguiente, aplicamos la transición 3 del autómata de pila quedando la pila como en el caso (d) de la figura 4 y leemos el siguiente símbolo de entrada; en el caso (d) de la figura 4 el símbolo en el tope de la pila es el no terminal <E> y el símbolo de entrada es un identificador cuyo valor es 5, por consiguiente, aplicamos la transición 4 del autómata de pila, quedando la pila como

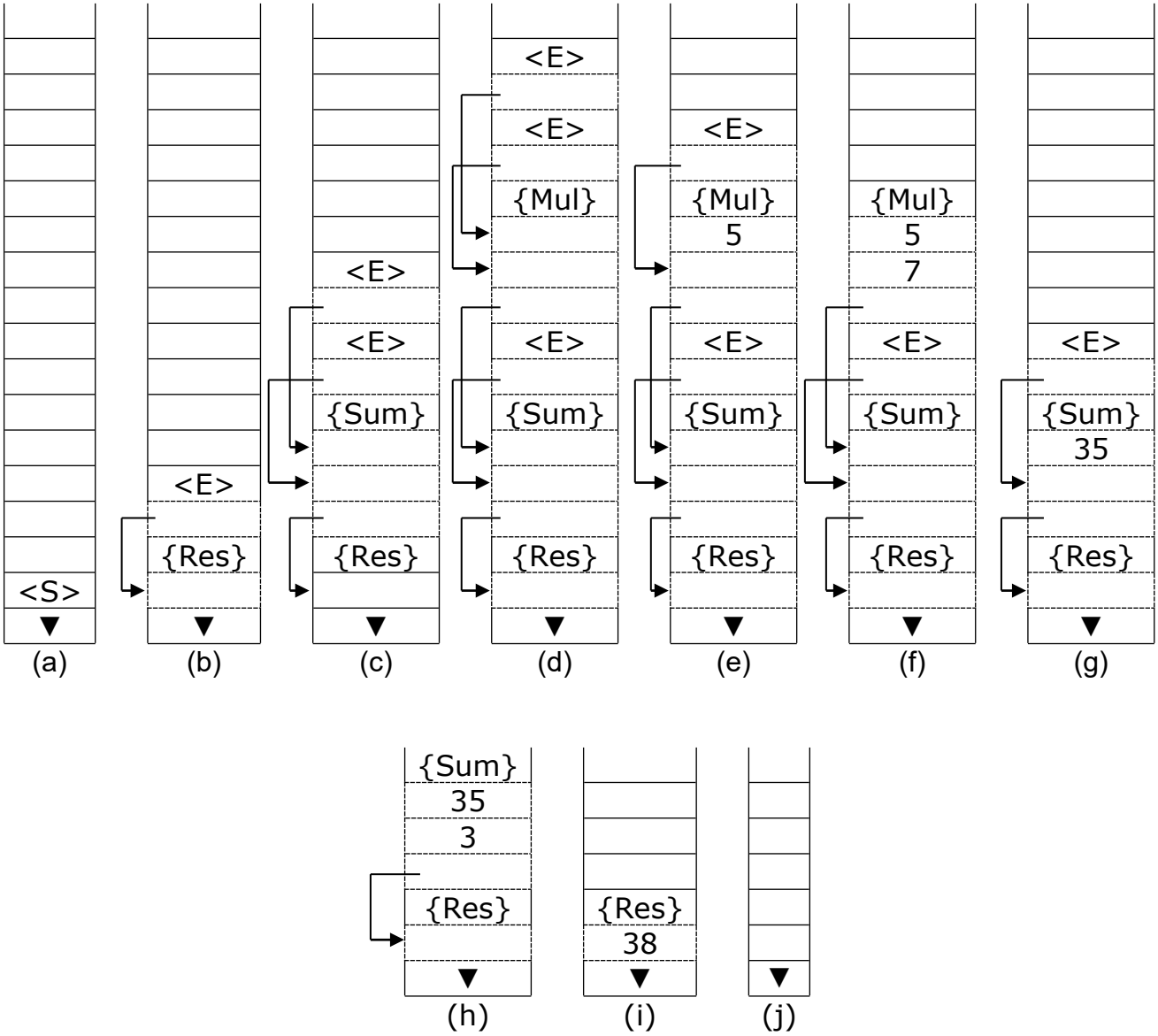


Figura 4

en el caso (e) y leemos el siguiente símbolo; en el caso (e) de la figura 4 el símbolo en el tope de la pila es el no terminal <E> y el símbolo de entrada es un identificador cuyo atributo tiene el valor de 7, por consiguiente, aplicamos nuevamente la transición 4 del autómata de pila quedando ésta como en el caso (f) de la figura 4; en el caso (f) de la figura 4 en el tope de la pila está el símbolo de acción de la multiplicación, por tanto, aplicamos la transición 6 del autómata de pila quedando la pila como en el caso (g), el símbolo de entrada se retiene; en el caso (g) de la figura 4 tenemos en el tope de la pila el no terminal <E> y el símbolo de entrada es un identificador cuyo atributo tiene el valor de 3, por tanto, aplicamos nuevamente la transición 4 del autómata de pila, quedando ésta como en el caso (h) de la figura 4; en el caso (h) el símbolo den el tope de la pila es el símbolo de acción correspondiente a la suma, por consiguiente, aplicamos la transición 5 del autómata de pila, quedando ésta como en el caso (i) de la figura 4, el símbolo de entrada se retiene; en el caso (i) de la figura 4 está el símbolo de acción del resultado en el tope de la pila, por consiguiente aplicamos la transición 7 del autómata de pila, quedando ésta como en el caso (j) de la figura 4; en el caso (j) el símbolo en el tope de la pila es el símbolo de pila vacía y el símbolo de entrada es el símbolo de fin de secuencia, por tanto se termina el proceso exitosamente.

EJERCICIOS PROPUESTOS

1. Las siguientes producciones son parte de una gramática con atributos. En cada producción los atributos p, q y r son heredados y s y t son sintéticos. Para cada producción establezca con cuáles atributos se pueden evaluar los atributos p y r para que la gramática sea L.

   1. $\langle S\rangle_{s,q} \rightarrow a_u \langle S\rangle_{t,p} \langle A\rangle_r$
   2. $\langle S\rangle_{s,q} \rightarrow \{c_p\} b_u \langle S\rangle_{t,r}$
   3. $\langle S\rangle_{s,q} \rightarrow c_u \langle A\rangle_r \langle S\rangle_{t,p}$

2. Reescriba cada una de las siguientes producciones tal que las reglas de evaluación de atributos queden en la forma de asignación simple. Los atributos que comienzan con H son heredados y los que comienzan con S son sintéticos.

   1. $\langle S\rangle_{S1,H1} \rightarrow \langle A\rangle_{S2} \langle B\rangle_{S3,H2} \langle C\rangle_{S4,H3} \langle D\rangle_{S5,H4}$
      $H2 = S2*H1;\ H3 = H1^2;\ H4 = H2*H3;\ S1 = S2^2$
   2. $\langle S\rangle_{S1,H1} \rightarrow \lambda$
      $S1 = SENO(H1)$
   3. $\langle S\rangle_{S1,S2} \rightarrow \langle A\rangle_{S3} \langle B\rangle_{H1} \{JUAN\}_{H2,H3} \langle C\rangle_{H4}$
      $H1 = 3*S3;\ H2 = H1^2;\ (S2, H3) = H2;\ H4 = H3;\ S1 = H4+2$

3. Convierta las reglas de evaluación de atributos de la siguiente gramática a la forma de asignación simple y construya reconocedor descendente de pila, mostrando la configuración antes y después de la pila correspondiente a cada transición, para la siguiente gramática:

   1. $\langle S\rangle_{a,b} \rightarrow a_c \langle S\rangle_{d,e} b_f \{c\}_{g,h} \langle A\rangle_{i,j} \{d\}_{m,n,p}$
      $e=c;\ g=b;\ h=f;\ j=b-c;\ m=i;\ n=j+3;\ p=d;\ a=c+n$
   2. $\langle S\rangle_{p,q} \rightarrow \lambda$
      $p=q$
   3. $\langle S\rangle_{a,b} \rightarrow c_s \{c\}_{u,v} \langle S\rangle_{p,q} \{c\}_{d,e} b_t \{d\}_{f,g,h}$
      $u=b;\ v=s;\ q=b-v;\ e=q;\ (a,f,g)=t;\ (d,h)=p$
   4. $\langle A\rangle_{p,q} \rightarrow d \langle S\rangle_{a,b} b_v \langle A\rangle_{r,s} \langle A\rangle_{c,d} \{g\}_t \{f\}_{m,n,u}$
      $b=q;\ m=r;\ (s,d,p)=v;\ n=c;\ u=t$
   5. $\langle A\rangle_{p,q} \rightarrow \{c\}_{r,t} c_s$
      $r=q;\ p=s;\ t=q+3$

      Los atributos son:
      $\langle S\rangle_{s,h}$ s: sintético; h: heredado
      El valor inicial de h es 1
      $\langle A\rangle_{s,h}$ s: sintético; h: heredado
      $\{g\}_s$ s:sintético
      t: Nuevo registro
      los atributos de {c}, {d} y {f} son heredados

4. Convierta las reglas de evaluación de atributos de la siguiente gramática a la forma de asignación simple y construya reconocedor descendente de pila, mostrando la configuración antes y después de la pila correspondiente a cada transición, para la siguiente gramática:

1. $<S>_{q} \rightarrow a_{x1}<A>_{x,y,z}<S>_{q1}\{b\}_{w}$
   $w=y*z+q1;\ x=x1;\ q=q1$
2. $<S>_{r1} \rightarrow ba_{y}\{c\}_{w}<A>_{u,r,x}\{b\}_{v}$
   $w=y+3;\ r1=r;\ v=r*x-w;\ u=w+2$
3. $<A>_{x,y,z1} \rightarrow a_{q}\{c\}_{v}<A>_{x1,z,u}<A>_{u,t,y1}$
   $v=x-q;\ x1=x;\ y=y1;\ z1=z;\ u=3$
4. $<A>_{t,s,s1} \rightarrow ba_{q}\{b\}_{t1}a_{s2}$
   $t1=t;\ (s,s1)=s2$

SOLUCIONES

1. Las siguientes producciones son parte de una gramática con atributos. En cada producción los atributos p, q y r son heredados y s y t son sintéticos. Para cada producción establezca con cuáles atributos se pueden evaluar los atributos p y r para que la gramática sea L.

1. $<S>_{s,q} \rightarrow a_u <S>_{t,p} <A>_r$
2. $<S>_{s,q} \rightarrow \{c_p\} b_u <S>_{t,r}$
3. $<S>_{s,q} \rightarrow c_u <A>_r <S>_{t,p}$

Solución:

En producción 1:

p = f(q, u)
r = f(p, q, t, u)

En producción 2:

p = f(q)
r = f(p, q, u)

En producción 3:

p = f(q, r, u)
r = f(q, u)

3. Convierta las reglas de evaluación de atributos de la siguiente gramática a la forma de asignación simple y construya reconocedor descendente de pila, mostrando la configuración antes y después de la pila correspondiente a cada transición, para la siguiente gramática:

1. $<S>_{a,b} \rightarrow a_c <S>_{d,e} z_f \{c\}_{g,h} <A>_{i,j} \{d\}_{m,n,p}$
   e=c; g=b; h=f; j=b-c; m=i; n=j+3; p=d; a=c+n
2. $<S>_{p,q} \rightarrow \lambda$
   p=q
3. $<S>_{a,b} \rightarrow c_s \{c\}_{u,v} <S>_{p,q} \{c\}_{d,e} z_t \{d\}_{f,g,h}$
   u=b; v=s; q=b-v; e=q; (a,f,g)=t; (d,h)=p
4. $<A>_{p,q} \rightarrow d <S>_{a,b} z_v <A>_{r,s} <A>_{c,d} \{g\}_t \{f\}_{m,n,u}$
   b=q; m=r; (s,d,p)=v; n=c; u=t
5. $<A>_{p,q} \rightarrow \{c\}_{r,t} c_s$
   r=q; p=s; t=q+3

Los atributos son:

$<S>_{s,h}$ s: sintético; h: heredado
El valor inicial de h es 1
$<A>_{s,h}$ s: sintético; h: heredado
$\{g\}_s$ s:sintético
t: Nuevo registro
los atributos de {c}, {d} y {f} son heredados

Solución:

En la producción 1 debemos incluir tres símbolos de acción: uno para el cálculo del atributo j, otro para el cálculo del atributo n y otro para el cálculo del atributo a.

En la producción 3 se debe incluir un símbolo de acción para el cálculo del atributo q, además se debe cambiar de sitio el símbolo de acción {c} para que cumpla las condiciones L.

En la producción 5 también se debe incluir un símbolo de acción para el cálculo del atributo t.

En todos los casos se debe tener cuidado con las condiciones de asignación simple, para poder efectuar el procesamiento descendente.

La gramática queda así:

1. $<S>_{a,b} \rightarrow a_c <S>_{d,e} b_f \{c\}_{g,h} \{fj\}_{b1,c1,j0} <A>_{i,j} \{fn\}_{j2,n0} \{d\}_{m,n,p} \{fa\}_{c2,n1,a0}$
   $(g, b_1)=b;\ (e,c_1,c_2)=c;\ h=f;\ m=i;\ p=d;\ (j, j_2)=j_0;\ a=a_0;\ (n, n_1)=n_0$
2. $<S>_{p,q} \rightarrow \lambda$
   $p=q$
3. $<S>_{a,b} \rightarrow c_s \{c\}_{u,v} \{fq\}_{b1,v1,q0} <S>_{p,q} b_t \{c\}_{d,e} \{d\}_{h,f,g}$
   $(b_1, u)=b;\ (v, v_1)=s;\ (q, e)=q_0;\ (a,f,g)=t;\ (d,h)=p$
4. $<A>_{p,q} \rightarrow d <S>_{a,b} b_v <A>_{r,s} <A>_{c,d} \{g\}_t \{f\}_{m,n,u}$
   $b=q;\ m=r;\ (s, d, p)=v;\ n=c;\ u=t$
5. $<A>_{p,q} \rightarrow c_s \{ft\}_{q1,t0} \{c\}_{r,t}$
   $r=q;\ p=s;\ q_1=q;\ t=t_0$

Los atributos son:

$<S>_{s,h}$ s: sintético; h: heredado
El valor inicial de h es 1

$<A>_{s,h}$ s: sintético; h: heredado

$\{g\}_s$ s:sintético

t: Nuevo registro.

Los atributos de {c}, {d} y {f} son heredados

Los atributos de {fa} son: los dos primeros heredados, el tercero sintético.

Los atributos de {fj} son: los dos primeros heredados, el tercero sintético.

Los atributos de {fn} son: el primero heredado, el segundo sintético.

Los atributos de {fq} son: los dos primeros heredados, el tercero sintético.

Los atributos de {ft} son: el primero heredado, el segundo sintético.

Ahora, para construir el autómata de pila construimos el conjunto de selección de cada producción.

Selección(1) = {a}
Selección(2) = {b}
Selección(3) = {c}
Selección(4) = {d}
Selección(5) = {c}

Símbolos de entrada = {a, b, c, d, ┤}
Símbolos en la pila = {<S>, <A>, b, {c}, {d}, {f}, {fa}, {fj}, {fn}, {fq}, {ft}, {g}, ▼}

Recuerde que cada símbolo en la pila va con sus respectivos atributos.

Configuración inicial de la pila: ▼<S> (ver figura 5)

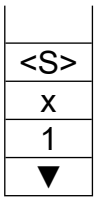


Figura 5

Transiciones:

	a	b	c	d	⊣
<S>	#1	#2	#3		
<A>			#5	#4	
b		#0			
▼					A
{c}	#6				
{d}	#7				
{f}	#8				
{fa}	#9				
{fj}	#10				
{fn}	#11				
{fg}	#12				
{ft}	#13				
{g}	#14				

Figura 6

#0: Lleve el dato entrado en b al sitio indicado por su atributo en la pila, desapile, avance.
#1: Reemplace como en la figura 7 (c es el atributo del terminal a), avance.
#2: Lleve el valor de q al sitio indicado por p, desapile, retenga.
#3: Reemplace como en la figura 8, avance.
#4: Reemplace como en la figura 9, avance.
#5: Reemplace como en la figura 10, avance.
#6: Ejecute la acción, desapile, retenga.
#7: Ejecute la acción, desapile, retenga.
#8:
#9:
#10:
#11:
#12:
#13:
#14:

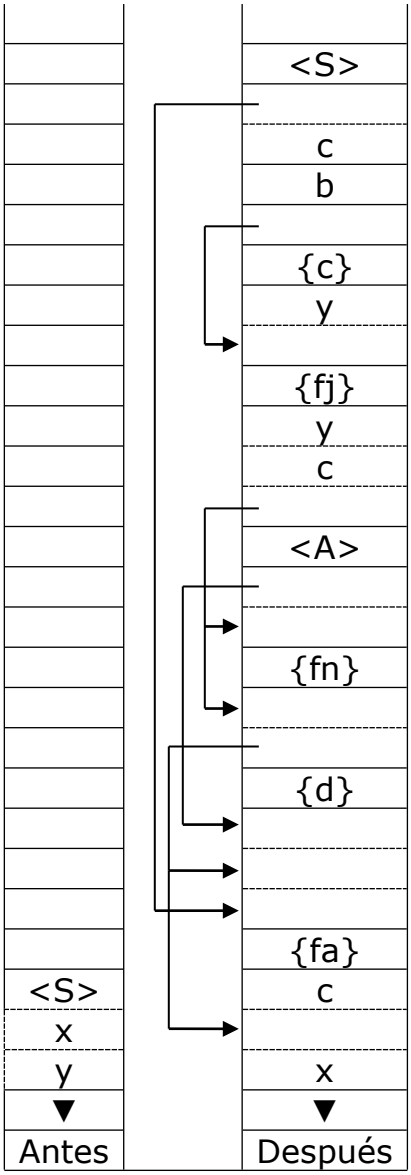


Figura 7

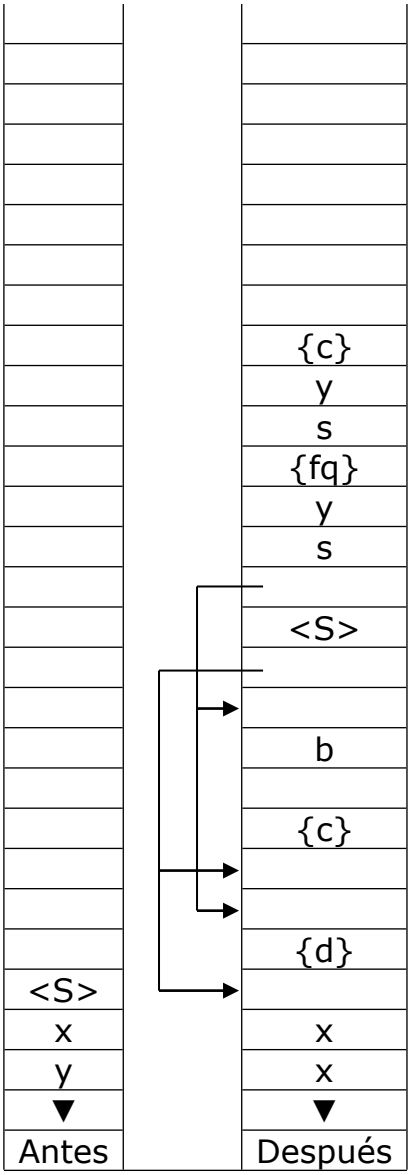


Figura 8

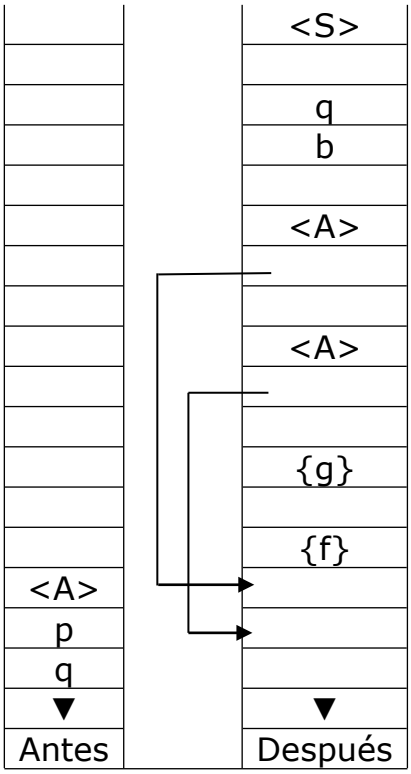


Figura 9

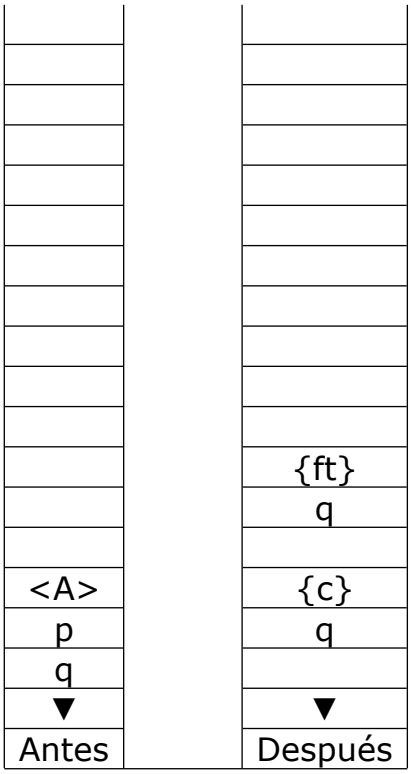


Figura 10

4. $\langle A\rangle_{p,q} \rightarrow d\langle S\rangle_{a,b} b_v \langle A\rangle_{r,s} \langle A\rangle_{c,d} \{g\}_t \{f\}_{m,n,u}$
   $b=q;\ m=r;\ (s, d, p)=v;\ n=c;\ u=t$
5. $\langle A\rangle_{p,q} \rightarrow c_s \{ft\}_{q1,t0} \{c\}_{r,t}$
   $r=q;\ p=s;\ q_1=q;\ t=t_0$

MODULO 11
TRANSFORMACIONES GRAMATICALES

11.1 INTRODUCCIÓN
Presentamos en los módulos anteriores la forma de reconocer el lenguaje que genera una gramática usando una técnica que se llama reconocimiento descendente. Para poder emplear esta técnica la gramática debe cumplir ciertas características. Veremos aquí algunas técnicas para convertir gramáticas que no cumplen estas características, de tal manera que se les pueda aplicar esta forma de construir el reconocedor.

11.2 OBJETIVOS
1. Convertir gramáticas a LL(1).

11.3 PREGUNTAS BÁSICAS
1. Qué es una transformación gramatical.
2. Cuándo una producción es una producción con esquina.
3. Cuándo un no terminal es recursivo por la izquierda.
4. Qué es una sustitución única.
5. Cuáles son los objetivos de las transformaciones gramaticales.

11.4 Factorización por la izquierda.
Es una transformación tendiente a convertir una gramática que no cumple las condiciones de LL(1) a LL(1). Consideremos una gramática en la cual aparecen las siguientes dos producciones:

1. <decisión> → if<condición>then<instrucciones>
2. <decisión> → if<condición>then<instrucciones>else<instrucciones>

Dicha gramática no es LL(1) puesto que el lado izquierdo de ambas producciones es el mismo (el no terminal <decisión>) y el conjunto de selección de ambas es el mismo también (el terminal if).

Cuando decimos factorización **por la izquierda** nos estamos refiriendo a los símbolos más a la izquierda del lado derecho de las producciones, y cuando hablamos de factorización, es el caso más simple de los procesos de factorización: factor común.

Las dos producciones mostradas las podremos reescribir así:

1. <decisión> → if<condición>then<instrucciones><restoDecisión>
2. <restoDecisión> → λ
3. <restoDecisión> → else<instrucciones>

La nueva producción 1 la hemos construido sacando el factor común de los símbolos más a la izquierda del lado derecho de las producciones 1 y 2, y agregamos un nuevo no terminal, el cual llamamos <restoDecisión>. Este factor común es **if <condición> then <instrucciones>**. Para el nuevo no terminal <restoDecisión>, definimos dos nuevas producciones, cada una de ellas con el resto de lado derecho de las producciones 1 y 2.

Las tres producciones obtenidas ya cumplen las condiciones LL(1).

Simbólicamente, esta transformación la definimos así: si tenemos un conjunto de n producciones de la forma: α β

1. <A> → $\boldsymbol{\alpha}\,\boldsymbol{\beta_1}$
2. <A> → $\boldsymbol{\alpha}\,\boldsymbol{\beta_2}$
.
.
.
n. <A> → $\boldsymbol{\alpha}\,\boldsymbol{\beta_n}$

con $\boldsymbol{\alpha}$ en $(T+N)^+$ y $\boldsymbol{\beta}$ en $(T+N)^*$, este conjunto de producciones no cumple las condiciones para que sea LL(1), puesto que el símbolo del lado izquierdo es el mismo y los conjuntos de selección de todas ellas es también el mismo.

Podemos tratar de convertir esta gramática a LL(1) aplicando la técnica de factorización por la izquierda así:

1. <A> → $\boldsymbol{\alpha}$<restoA>
2. <restoA> → $\boldsymbol{\beta_1}$
3. <restoA> → $\boldsymbol{\beta_2}$
.
.
.
n. <restoA> → $\boldsymbol{\beta_n}$

Cuando se aplica la transformación, factorización por la izquierda, no se garantiza que la gramática resultante sea LL(1).

11.5 Sustitución de la esquina.

Definamos primero lo que es una producción con esquina. Una producción con esquina es una producción cuyo lado derecho comienza con un no terminal. La esquina de la producción es precisamente el no terminal con el cual comienza su lado derecho. Consideremos la siguiente gramática:

1. <A> → a
2. <A> → <B>c
3. <B> → a<A>
4. <B> → b<B>

La producción 2 es una producción con esquina, puesto que su lado derecho comienza con un no terminal. La esquina de la producción 2 es el no terminal <B>.

Si quisiéramos construir el reconocedor descendente de pila para dicha gramática lo primero que debemos hacer es determinar si es LL(1) o no. Para determinar si es LL(1) o no construimos el conjunto de selección de cada producción. Los conjuntos de selección de las producciones de dicha gramática son:

Selección(1) = {a}
Selección(2) = {a, b}
Selección(3) = {a}
Selección(4) = {b}

Dicha gramática no es LL(1) debido a que los conjuntos de selección de las producciones 1 y 2, cuyo símbolo del lado izquierdo es el mismo, no son disyuntos. Para tratar de convertir dicha gramática a LL(1) lo primero que hacemos es aplicar la sustitución de la esquina de la producción 2. El no terminal <B> de la producción 2 lo reemplazamos por el lado derecho de las producciones 3 y 4 que son las que definen dicho no terminal. Nuestra gramática queda:

1. <A> → a
2. <A> → a<A>c
3. <A> → b<B>c
4. <B> → a<A>
5. <B> → b<B>

Como puede observar, nuestra gramática aún no es LL(1), ya que el conjunto de selección de las producciones 1 y 2 es el mismo: el terminal a. Sin embargo, en este punto podemos aplicar la transformación de factorización por la izquierda en dichas producciones. Nuestra gramática queda:

1. <A> → a<restoA>
2. <restoA> → λ
3. <restoA> → <A>c
4. <A> → b<B>c
5. <B> → a<A>
6. <B> → b<B>

La cual, al organizarla por los símbolos del lado izquierdo queda:

1. <A> → a<restoA>
2. <A> → b<B>c
3. <restoA> → λ
4. <restoA> → <A>c
5. <B> → a<A>
6. <B> → b<B>

Los conjuntos de selección de esta nueva gramática son:

Selección(1) = {a}
Selección(2) = {b}
Selección(3) = {c, ┤}
Selección(4) = {a, b}
Selección(5) = {a}
Selección(6) = {b}

Que como puede observar, cumple las condiciones para que una gramática sea LL(1). Esta gramática genera exactamente el mismo lenguaje que la gramática original y es LL(1), por tanto, podemos construir el reconocedor descendente de pila que reconoce el lenguaje que ella genera.

11.6 Sustitución única.
Esta es una transformación cuyo objetivo es construir gramáticas con menos producciones. Se efectúa cuando existe algún no terminal que sólo tiene una producción que lo define. Consideremos la siguiente gramática:

1. <A> → a<B><B>c
2. <A> → b<B><C>
3. <B> → b<C>a
4. <C> → c<A>b
5. <C> → λ

Observe que el no terminal <B> sólo tiene una producción que lo define, la cual es la producción 3. Podemos sustituir el no terminal <B> en el lado derecho de las producciones donde aparezca, por el lado derecho de la producción 3, y eliminar la producción 3. Nuestra gramática queda:

1. <A> → a b<C>ab<C>ac
2. <A> → b b<C>a<C>
3. <C> → c<A>b
4. <C> → λ

Podrá parecer una bobada, sin embargo, esta transformación es útil en algunas situaciones.

11.7 Eliminación de producciones anulables.
Recordemos que producciones anulables son aquellas a partir de las cuales se puede obtener la secuencia nula. Para que existan producciones anulables tiene que haber producciones cuyo lado derecho sea la secuencia nula **(λ)**. Si eliminamos estas producciones habremos eliminado las producciones anulables. Veamos entonces cómo eliminar producciones cuyo lado derecho es la secuencia nula **(λ)**.

Consideremos la siguiente gramática:

1. <A> → a<B><B>c<C>
2. <A> → b<B>a
3. <B> → c<B>a
4. <B> → b<A>
5. <B> → λ
6. <C> → a<A>b
7. <C> → c

En esta gramática existe una producción cuyo lado derecho es la secuencia nula. La producción 5. Veamos cómo eliminar esta producción.

Cada producción en la cual aparezca el no terminal <B> en el lado derecho, vamos a reemplazarla por dos producciones: una en la cual colocamos el no terminal <B> como $<B>_{si}$ y otra en la cual colocamos el no terminal <B> como $<B>_{no}$. Cada ocurrencia del no terminal <B> en el lado derecho de una producción implicará dos producciones. La producción que defina el no terminal <B> como la secuencia nula la llamaremos $<B>_{no}$ y las otras producciones que definan el no terminal <B> las llamaremos $<B>_{si}$.

Al hacer estos reemplazos en la gramática dada, ésta queda así:

1. <A> → a<B>$_{si}$<B>$_{si}$c<C>
2. <A> → a<B>$_{si}$<B>$_{no}$c<C>
3. <A> → a<B>$_{no}$<B>$_{si}$c<C>
4. <A> → a<B>$_{no}$<B>$_{no}$c<C>
5. <A> → b<B>$_{si}$a
6. <A> → b<B>$_{no}$a
7. <B>$_{si}$ → c<B>$_{si}$a
8. <B>$_{si}$ → c<B>$_{no}$a
9. <B>$_{si}$ → b<A>
10. <B>$_{no}$ → λ
11. <C> → a<A>b
12. <C> → c

Al tener la gramática de esta forma estamos en la situación en la cual hay un no terminal que sólo tiene una producción que lo define, por consiguiente, podemos aplicar la transformación de sustitución única. Fíjese que el no terminal que tiene una sola producción es el no terminal <B>$_{no}$, el cual es el único cuyo lado derecho es la secuencia nula. Al hacer esta sustitución, la gramática queda sin producciones cuyo lado derecho sea la secuencia nula. Nuestra gramática queda así:

1. <A> → a<B><B>c<C>
2. <A> → a<B>c<C>
3. <A> → a<B>c<C>
4. <A> → ac<C>
5. <A> → b<B>a
6. <A> → ba
7. <B> → c<B>a
8. <B> → ca
9. <B> → b<A>
10. <C> → a<A>b
11. <C> → c

Como podrá observarlas producciones 2 y 3 son idénticas, por consiguiente podemos eliminar una de ellas. La gramática queda así:

1. <A> → a<B><B>c<C>
2. <A> → a<B>c<C>
3. <A> → ac<C>
4. <A> → b<B>a
5. <A> → ba
6. <B> → c<B>a
7. <B> → ca
8. <B> → b<A>
9. <C> → a<A>b
10. <C> → c

Esta gramática no tiene producciones anulables y genera exactamente el mismo lenguaje que la gramática original.

Es importante hacer notar que esta transformación es aplicable sólo si el símbolo inicial de la gramática no es anulable.

Si el símbolo inicial de la gramática es anulable esta transformación no es viable puesto que modificaría la gramática.

11.8 Transformación de gramáticas con producciones recursivas a sí mismas por la izquierda.

Para definir esta transformación, comencemos definiendo lo que es un no terminal recursivo por la izquierda. Un no terminal <X> es recursivo por la izquierda si a partir de él es posible obtener una hilera que comience con <X> mediante algún proceso de derivación. Consideremos la siguiente gramática:

1. <A> → <B>c<D>a
2. <A> → a<B>b
3. <B> → b<A><C>
4. <B> → λ
5. <B> → <C><D>a
6. <C> → <D>a<B>
7. <C> → c
8. <D> → d
9. <D> → <A><B>c

Si hacemos un proceso de derivación como el siguiente tendremos:

<A>
↑ producción 1
<A> → <B>c<D>a
↑ producción 5
<A> → <C><D>ac<D>a
↑ producción 6
<A> → <D>a<B><D>ac<D>a
↑ producción 9
<A> → <A><B>ca<B><D>ac<D>a

O sea, a partir del no terminal <A> hemos obtenido una hilera que comienza con el no terminal <A>. Cuando se pueda hacer un proceso de derivación, a partir de un no terminal cualquiera, y se obtenga una hilera que comience con ese mismo no terminal, se dice que ese no terminal es recursivo por la izquierda.

En nuestro ejemplo, el no terminal <A> es recursivo por la izquierda.

Si se tiene una gramática que tenga algún no terminal recursivo por la izquierda, es condición suficiente para afirmar que dicha gramática no es LL(1).

Ahora, si se tiene una producción en la cual el símbolo del lado izquierdo es el mismo con el cual comienza su lado derecho, esa producción es recursiva a sí misma por la izquierda. Es decir, una producción de la forma:

$$<S> \rightarrow <S>\alpha$$

Con α en $(T+N)^+$ es una producción recursiva a sí misma por la izquierda.

Si una gramática tiene producciones recursivas a sí mismas por la izquierda, es condición suficiente para afirmar que esa gramática no es LL(1).

Veremos en esta transformación, cómo, cuando se tiene una gramática con producciones recursivas a sí mismas por la izquierda, podemos tratar de convertirla a LL(1).

Consideremos el caso general de una manera simbólica. Sea una gramática con producciones de la siguiente forma:

1. <A> → <A>α_1
2. <A> → <A>α_2
.
.
.
n. <A> → <A>α_n
n+1. <A> → β_1
n+2. <A> → β_2
.
.
.
n+m. <A> → β_m

Las producciones 1 a n son recursivas a sí mismas por la izquierda. Las producciones n+1 a m tienen el mismo lado izquierdo que las producciones 1 a n y no son recursivas a sí mismas por la izquierda. Los alfas y los betas representan cualquier hilera de símbolos, terminales y no terminales. Los beta (β) pueden ser la secuencia nula, los alfa (α) no. Simbólicamente, α pertenece a $(T+N)^+$ y β pertenece a $(T+N)^*$.

La forma como tratamos de convertir esta gramática a LL(1) es con la siguiente transformación:

1. <A> → β_1<restoA>
2. <A> → β_2<restoA>
.
.
.
m. <A> → β_m<restoA>
m+1. <restoA> → α_1<restoA>
m+2. <restoA> → α_2<restoA>
.
.
.
n+m. <restoA> → α_n<restoA>
n+m+1. <restoA> → λ

Fíjese en lo que consiste la transformación: hemos creado un nuevo no terminal, el cual, hemos llamado <restoA>; hemos redefinido las producciones recursivas a sí

mismas por la izquierda, comenzando con las hileras **β** seguidas del nuevo no terminal <restoA>; el nuevo no terminal <restoA>, lo hemos definido como los **α** seguidos del nuevo no terminal <restoA>, y hemos agregado una producción para el nuevo no terminal <restoA> con lado derecho la secuencia nula (**λ**).

Presentemos algunos ejemplos.

Si nos piden construir una gramática que genere ba^n con n >= 0, una gramática obvia es:

1. <B> → b<A>
2. <A> → a<A>
3. <A> → λ

Sin embargo, podemos construir una gramática como esta:

1. <B> → <B>a
2. <B> → b

la cual, genera exactamente el mismo lenguaje.

La gramática obvia, la primera, es una gramática LL(1), mientras que la segunda gramática no es LL(1).

La segunda gramática es una gramática que tiene una producción recursiva a sí misma por la izquierda: la producción 1. Si a esta gramática le aplicamos la transformación que estamos planteando tendremos: el terminal **a** de la producción 1 es un alfa (**α**) y el terminal **b** de la producción 2 es un beta (**β**). La gramática quedará:

1. <B> → b<restoB>
2. <restoB> → a<restoB>
3. <restoB> → λ

Donde el no terminal <restoB> es equivalente al no terminal <A> que tenemos en la gramática inicialmente escrita.

EJERCICIOS PROPUESTOS

Determine si las siguientes gramáticas son LL(1). En caso de no serlas, trate de convertirlas a LL(1).

1.
    1. <S> → <S>c
    2. <S> → c<A>
    3. <S> → λ
    4. <A> → a<A>
    5. <A> → a

2.
    7. <S> → <A><S><B>
    8. <S> → a

9. <A> → a<A>
10. <A> → λ
11. <B> → b<B>
12. <B> → λ

3.

1. <S> → <A><B><C><D>
2. <A> → a<A>p
3. <A> → λ
4. <B> → <B>q
5. <B> → b
6. <C> → <A><F>
7. <C> → <C>
8. <D> → d<D>r
9. <D> → λ
10. <F> → f<F>g
11. <F> → λ

4.

1. <S> → a<A><B>b<C><D>
2. <S> → λ
3. <A> → <A><S>d
4. <A> → λ
5. <B> → <S><A>c
6. <B> → e<C>
7. <B> → λ
8. <C> → <S>f
9. <C> → <C>g
10. <C> → λ
11. <D> → a<B><D>
12. <D> → λ

5.

9. <A> → <A>c<B>
10. <A> → c<C>
11. <A> → <C>
12. <B> → b<B>
13. <B> → i
14. <C> → <C>a<B>
15. <C> → <B>b<B>
16. <C> → <B>

SOLUCIONES

1. Determine si la siguiente gramática es LL(1). En caso de no serla, trate de convertirla a LL(1).

1. <S> → <S>c
2. <S> → c<A>
3. <S> → λ
4. <A> → a<A>
5. <A> → a

Solución:

La gramática no es LL(1). El mero hecho de que la producción 1 es recursiva a sí misma por la izquierda es suficiente para asegurar que la gramática no es LL(1). Además, producciones 4 y 5 cuyo símbolo del lado izquierdo es el mismo, el lado derecho de ambas producciones comienza con el mismo terminal.

Para tratar de convertirla a LL(1) aplicamos la técnica descrita en el numeral 11.8 a las producciones 1 a 3, y la técnica descrita en 11.4 a las producciones 4 y 5. La gramática queda:

1. <S> → c<A><rs>
2. <S> → <rs>
3. <rs> → c<rs>
4. <rs> → λ
5. <A> → a<ra>
6. <ra> → <A>
7. <ra> → λ

La gramática resultante no es LL(1) por producciones 1 y 2. El conjunto de selección de la producción 1 es el Terminal c, el conjunto de selección de la producción 2 son los primeros de <rs>, el cual es el mismo terminal c. Aplicando sustitución de la esquina en la producción 2 la gramática queda:

1. <S> → c<A><rs>
2. <S> → c<rs>
3. <S> → λ
4. <rs> → c<rs>
5. <rs> → λ
6. <A> → a<ra>
7. <ra> → <A>
8. <ra> → λ

Ahora aplicamos factorización por la izquierda en producciones 1 y 2 y obtenemos:

1. <S> → c<rt>
2. <rt> → <A><rs>
3. <rt> → <rs>
4. <S> → λ
5. <rs> → c<rs>
6. <rs> → λ
7. <A> → a<ra>

8. <ra> → <A>
9. <ra> → λ

La cual, ya es una gramática LL(1):

Selección(1) = {c}
Selección(2) = {a}
Selección(3) = {c, ┤}
Selección(4) = {┤}
Selección(5) = {c}
Selección(6) = {┤}
Selección(7) = {a}
Selección(8) = {a}
Selección(9) = {c, ┤}

3. Determine si la siguiente gramática es LL(1). En caso de no serla, trate de convertirla a LL(1).

1. <S> → <A><B><C><D>
2. <A> → a<A>p
3. <A> → λ
4. <B> → <B>q
5. <B> → b
6. <C> → <A><F>
7. <C> → <C>
8. <D> → d<D>r
9. <D> → λ
10. <F> → f<F>g
11. <F> → λ

Solución:

La gramática no es LL(1). El mero hecho de que la producción 4 es recursiva a sí misma por la izquierda es suficiente para asegurar que la gramática no es LL(1). Adicionalmente, la producción 7 sobra.

Aplicando la transformación descrita en 11.8 y eliminando la producción 7 la gramática queda:

1. <S> → <A><B><C><D>
2. <A> → a<A>p
3. <A> → λ
4. <B> → b<rb>
5. <rb> → q<rb>
6. <rb> → λ
7. <C> → <A><F>
8. <D> → d<D>r
9. <D> → λ
10. <F> → f<F>g
11. <F> → λ

Los no terminales <B> y <C> sólo tiene una producción que los define, por lo tanto, aplicamos sustitución única en la producción 1. Nuestra gramática queda:

1. <S> → <A>b<rb><A><F><D>
2. <A> → a<A>p
3. <A> → λ
4. <rb> → q<rb>
5. <rb> → λ
6. <D> → d<D>r
7. <D> → λ
8. <F> → f<F>g
9. <F> → λ

Al construir el conjunto de selección de cada producción tenemos:

Selección(1) = {a, b}
Selección(2) = {a}
Selección(3) = {b, d, f, p, ┤}
Selección(4) = {q}
Selección(5) = {a, d, f, ┤}
Selección(6) = {d}
Selección(7) = {r, ┤}
Selección(8) = {f}
Selección(9) = {d, g, ┤}

Lo cual significa que la gramática es LL(1).

5. Determine si la siguiente gramática es LL(1). En caso de no serla, trate de convertirla a LL(1).

1. <A> → <A>c<B>
2. <A> → c<C>
3. <A> → <C>
4. <B> → b<B>
5. <B> → i
6. <C> → <C>a<B>
7. <C> → <B>b<B>
8. <C> → <B>

Solución:

La gramática no es LL(1). El hecho de que las producciones 1 y 6 sean recursivas a sí mismas por la izquierda es suficiente para deducir que la gramática no es LL(1).

Aplicando la transformación descrita en el numeral 11.8 a los conjuntos de producciones 1 a 3 y 6 a 8 la gramática queda:

1. <A> → c<C><ra>
2. <A> → <C><ra>
3. <ra> → c<B><ra>
4. <ra> → λ
5. <B> → b<B>
6. <B> → i
7. <C> → <B>b<B><rc>
8. <C> → <B><rc>

9. <rc> → a<B><rc>
10. <rc> → λ

La gramática obtenida con esta transformación tampoco es LL(1) ya que las producciones 7 y 8, cuyo lado izquierdo es el mismo, comienzan con el mismo no terminal <B>. Aplicando la transformación definida en el numeral 11.4, factorización por la izquierda obtenemos:

1. <A> → c<C><ra>
2. <A> → <C><ra>
3. <ra> → c<B><ra>
4. <ra> → λ
5. <B> → b<B>
6. <B> → i
7. <C> → <B><rcc>
8. <rcc> → b<B><rc>
9. <rcc> → <rc>
10. <rc> → a<B><rc>
11. <rc> → λ

Construyamos ahora el conjunto de selección de cada producción para determinar si es LL(1) o no.

Primeros(<A>) = {b, c, i}
Primeros(<ra>) = {c}
Primeros(<B>) = {b, i}
Primeros(<C>) = {b, i}
Primeros(<rc>) = {a}
Primeros(<rcc>) = {a, b}

Siguientes(<A>) = {┤}
Siguientes(<ra>) = {┤}
Siguientes(<B>) = {a, b, c, ┤}
Siguientes(<C>) = {c, ┤}
Siguientes(<rc>) = {c, ┤}
Siguientes(<rcc>) = {c, ┤}

Selección(1) = {c}
Selección(2) = {b, i}
Selección(3) = {c}
Selección(4) = {┤}
Selección(5) = {b}
Selección(6) = {i}
Selección(7) = {b, i}
Selección(8) = {b}
Selección(9) = {a, c, ┤}
Selección(10) = {a}
Selección(11) = {c, ┤}

Como se puede observar, producciones cuyo símbolo del lado izquierdo es el mismo, tienen conjuntos de selección disyuntos, por consiguiente la gramática ya es LL(1).

MODULO 12
RECONOCIMIENTO ASCENDENTE TECNICA SHIFT-IDENTIFY

12.1 INTRODUCCIÓN
Hemos venido tratando en los módulos anteriores la construcción de autómatas de pila para reconocer el lenguaje que genera una gramática. Dicha técnica se denomina reconocimiento descendente. Veremos ahora una forma diferente de efectuar el reconocimiento del lenguaje que genera una gramática: reconocimiento ascendente.

12.2 OBJETIVOS
1. Entender cómo funciona el reconocimiento ascendente.
2. Construir autómata de pila que efectúe reconocimiento ascendente, técnica shift-identify, del lenguaje que genera una gramática dada.

12.3 PREGUNTAS BÁSICAS
1. Qué es un manejador.
2. Cuál es la configuración inicial de la pila en un reconocedor ascendente.
3. Para qué se utiliza el principio de shift.
4. Para que se utiliza el principio de reducción.
5. Cuál debe ser la configuración final de la pila en un reconocimiento ascendente.
6. Cuándo se hace proceso de identificación.
7. Qué características debe tener una gramática para poder construir su reconocedor ascendente de pila.

12.4 Conceptos básicos
El primer concepto básico es el de manejador. **Un manejador es el último símbolo del lado derecho de una producción**. La idea básica del reconocimiento ascendente es construir en la pila el lado derecho de alguna producción.

Comencemos presentando un ejemplo para ilustrar cómo funciona el reconocimiento ascendente. Sea la siguiente gramática:

1. <S> → (<A><S>)
2. <S> → (b)
3. <A> → (<S>a<A>)
4. <A> → (a)

y la siguiente hilera **(((b)a(a))(b))**┤ cuyo árbol de derivación es:

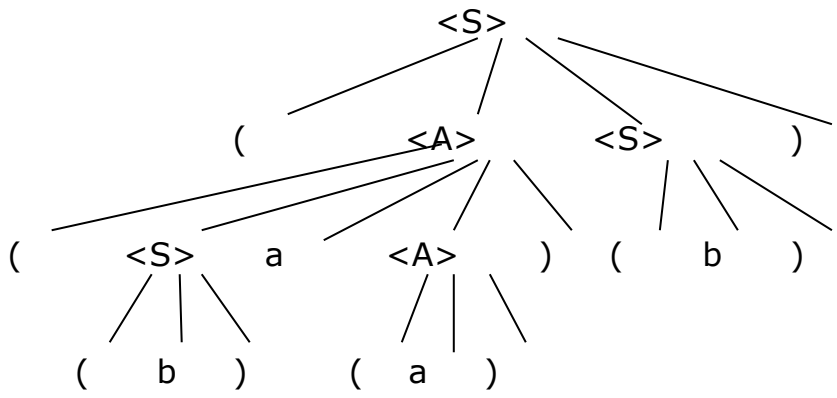


Figura 12.1

El proceso de reconocimiento ascendente, al igual que el descendente funciona con un autómata de pila, con la diferencia de que la configuración inicial de la pila es sólo el símbolo de pila vacía (▼).

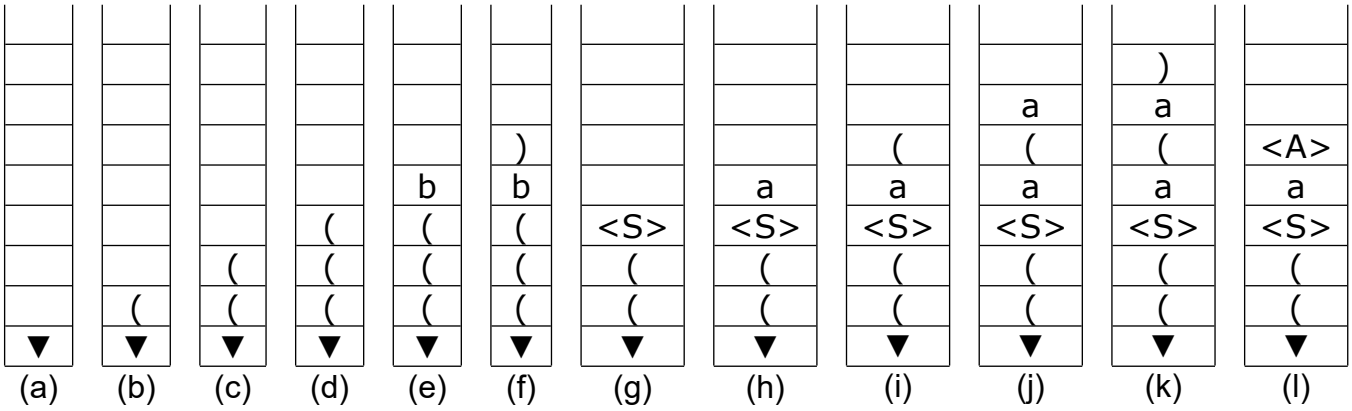


Figura 12.9a

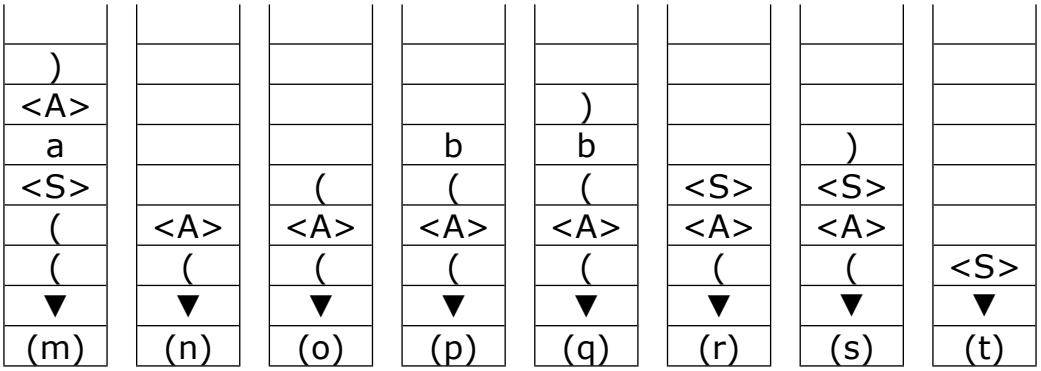


Figura 12.9b

Como hemos dicho la técnica ascendente consiste en construir el lado derecho de las producciones en la pila, de tal manera que se pueda ir identificando cuáles producciones fueron las que se aplicaron. Con este propósito en mente, leemos la hilera de entrada símbolo por símbolo, y los vamos llevando a la pila hasta que detectemos que en el tope de la pila haya un manejador (el último símbolo del lado derecho de una producción). Cuando en el tope de la pila haya un manejador procedemos a hacer un proceso de identificación en el cual se determina, los últimos símbolos en la pila, al lado derecho de cuál producción corresponden.

En nuestro ejemplo, los casos desde la (a) hasta la (f), de la figura 12.9a, corresponden a los primeros símbolos de la hilera que vamos a reconocer: **(((b)**

Cuando estamos en el caso (f), en el tope de la pila hay un cierre paréntesis y el símbolo de entrada es un abre paréntesis. En el tope de la pila hay un manejador, por consiguiente, debemos determinar, los últimos símbolos en la pila, al lado derecho de cuál producción corresponden. Como se podrá ver, corresponden al lado derecho de la producción 2, por tanto, se hace un proceso de **reducción** por la producción número 2, el cual consiste en desapilar tantos símbolos como tenga el lado derecho de la producción 2 y apilar el símbolo del lado izquierdo de la producción 2: la pila queda como en el caso (g) de la figura 12.9a y el árbol de derivación queda como en la figura 12.2.

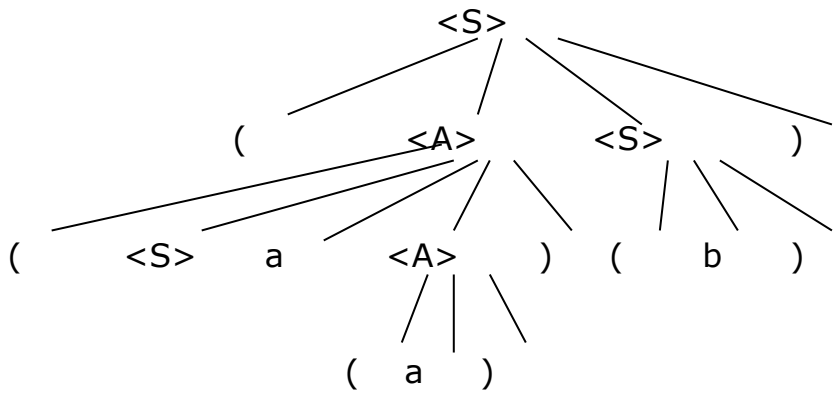


Figura 12.2

En el caso (g) de la figura 12.9, en el tope de la pila está el no terminal <S> y el símbolo de entrada es un abre paréntesis. Como el no terminal <S> no es un manejador simplemente se apila el abre paréntesis y se lee el siguiente símbolo de la hilera a reconocer. Este proceso, apile y avance, se continúa hasta que en el tope de la pila haya nuevamente un manejador. En el caso (k) de la figura 12.9a aparece nuevamente un manejador en el tope de la pila, y los últimos símbolos en la pila corresponden al lado derecho de la producción 4, por tanto reducimos por la producción 4, es decir, se desapilan tres símbolos y se lleva a la pila el símbolo del lado izquierdo de la producción 4, quedando la pila como en el caso (l) de la figura 12.9a. El árbol de derivación queda como en la figura 12.3.

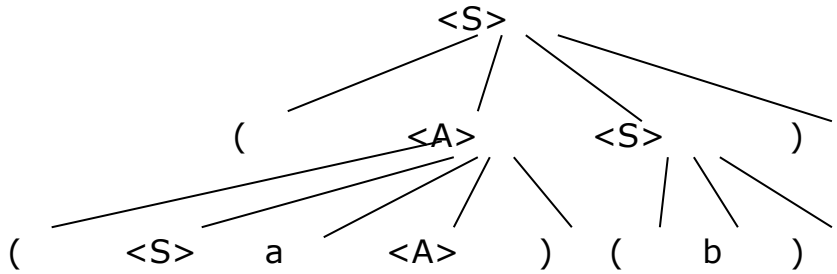


Figura 12.3

Fíjese que estamos "podando" el árbol, cada vez que identificamos cuál producción se aplicó. Fíjese que estamos reconociendo desde abajo hacia arriba en el árbol de derivación. El método se denomina reconocimiento *ascendente*.

En el caso (l), en el tope de la pila está el no terminal <A> y el símbolo de entrada es un cierre paréntesis. Como el no terminal <A> no es un manejador simplemente apilamos el cierre paréntesis y leemos el siguiente símbolo quedando la pila como en el caso (m) de la figura 9.12b.

En el caso (m) de la figura 9.12b hay nuevamente un manejador en el tope de la pila, y los último símbolos en la pila corresponden al lado derecho de la producción 3, por tanto, hay que reducir por la producción 3, lo cual consiste en desapilar 5 símbolos y apilar el no terminal <A>, quedando la pila como en el caso (n) de la figura 12.9b y el árbol de derivación como en la figura 12.4.

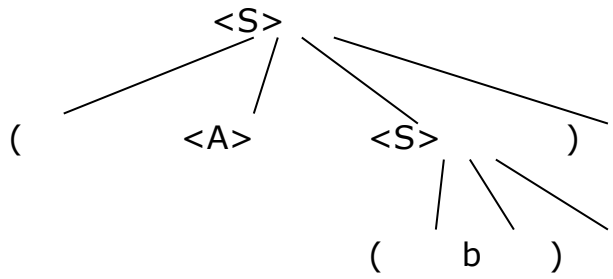


Figura 12.4

Continuamos nuestro proceso de leer símbolo e ir apilándolo hasta que en el tope de la pila haya nuevamente un manejador, lo cual sucede en el caso (q) de la figura 12.9b. En este caso, los últimos símbolos en la pila corresponden al lado de recho de la producción 2, lo cual implica reducir por dicha producción, quedando la pila como en el caso (r) de la figura 12.9b, y el árbol de derivación como en la figura 12.5.

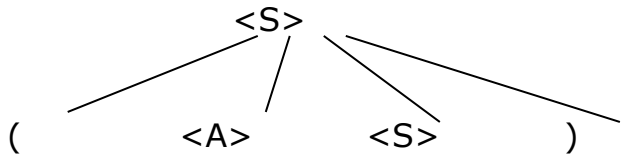


Figura 12.5

El siguiente símbolo es un cierre paréntesis, por tanto lo apilamos y leemos el siguiente símbolo. La pila queda como en el caso (s) de la figura 12.9b.

En el caso (s) hay un manejador en el tope de la pila, los últimos símbolos en la pila corresponden al lado derecho de la producción 1, por tanto reducimos por la producción 1: se desapilan cuatro símbolos y se apila el símbolo del lado izquierdo de la producción 1, quedando la pila como en el caso (t) de la figura 12.9b, y el árbol de derivación como en la figura 12.6.

<S>
Figura 12.6

En el caso (t) de la figura 12.9b en el tope de la pila está el no terminal <S> y el símbolo de entrada es el símbolo de fin de secuencia (┤). El no terminal <S> es el símbolo inicial de la gramática.

Cuando en el tope de la pila se encuentre el símbolo inicial de la gramática y el símbolo de entrada sea el símbolo de fin de secuencia (┤) debemos hacer nuevamente un proceso de identificación: si los últimos símbolos en la pila son el símbolo de pila vacía (▼) y el símbolo inicial de la gramática, entonces se acepta la hilera procesada, de lo contrario se rechaza.

Esta forma de reconocer el lenguaje que genera una gramática se conoce como reconocimiento ascendente, técnica SHIFT-IDENTIFY.

12.5 Construcción autómata de pila.

En el numeral anterior vimos la forma como se efectúa el reconocimiento ascendente, técnica shift-identify, con base en una gramática. Fijémonos que hemos utilizado un autómata de pila de un estado y que las operaciones que hemos efectuado han sido: apilar y avanzar, hacer proceso de identificación y reducir. Veamos cómo es el proceso para construir dicho autómata de pila.

Recordemos que para construir un autómata de pila es necesario conocer los símbolos de entrada, los símbolos en la pila, la configuración inicial de la pila y las transiciones.

Los símbolos de entrada serán los terminales de la gramática y el símbolo de fin de secuencia (┤).

Los símbolos en la pila serán los terminales de la gramática, los no terminales de la gramática y el símbolo de pila vacía (▼).

La configuración inicial de la pila es el símbolo de pila vacía (▼).

Las transiciones. Para determinar las transiciones se define una convención: sea **A** el símbolo en el tope de la pila y sea **x** el símbolo de entrada. Se utilizan dos principios los cuales son:

Principio de SHIFT: se utiliza para determinar cuáles transiciones son shift (apile, avance). Se hará shift cuando se presente una de las dos siguientes situaciones:

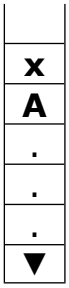
1. Cuando **A** sea el símbolo inicial de la gramática y **x** pertenezca a primeros(<S>) siendo <S> el símbolo inicial de la gramática.

2. Existe una hilera **A**B en el lado derecho de alguna producción y **x** pertenece a primeros(B).

El principio de shift se enuncia:

A estáDebajoDe x

Cuando se dice que **A** está debajo de **x** se refiere a su ubicación en la pila. Fíjese en esto: **A** está en el tope de la pila y se apila **x**, por consiguiente la pila quedará:



Es decir, **A** está debajo de **x**.

Principio de reducción: se utiliza para determinar cuáles transiciones serán proceso de identificación. Se hará proceso de identificación cuando se presente una de las dos siguientes situaciones:

1. **A** es el símbolo inicial de la gramática y **x** es el símbolo de fin de secuencia (┤).

2. Existe una producción de la forma: <N> → **αA** y **x** pertenece a siguientes(<N>).

El principio de reducción se define:

A esReducidoPor x

12.6 Un ejemplo.

Teniendo definida la forma como se construye el autómata de pila para hacer reconocimiento ascendente, técnica shift-identify, consideremos el siguiente ejemplo:

1. <S> → b<A><S><B>
2. <S> → b<A>
3. <A> → d<S>ca
4. <A> → e
5. <B> → c<A>a
6. <B> → c

Nuestro primer paso es construir los conjuntos de primeros y siguientes para cada no terminal. En este tipo de gramáticas es bastante sencillo, puesto que no se tienen producciones cuyo lado derecho es la secuencia nula (λ). Estos conjuntos son necesarios para aplicar los principios de shift y reducción.

Primeros(<S>) = {b} Siguientes(<S>) = {c, ┤}
Primeros(<A>) = {d, e} Siguientes(<A>) = {b, c, ┤}
Primeros(<B>) = {c} Siguientes(<B>) = {c, ┤}

Símbolos de entrada = {a, b, c, d, e, ┤}
Símbolos en la pila = {<S>, <A>, <B>, a, b, c, d, e, ▼}
Configuración inicial de la pila: ▼
Tabla de transiciones:

	a	b	c	d	e	┤
<S>			S			id0
<A>	S	S	id2			id2
<B>			id1			id1
a		id3	id3			id3
b				S	S	
c	S		id6	S	S	id6
d		S				
e		id4	id4			id4
▼		S				

Tabla 12.1

donde:

S: apile, avance.

id0: if usep == ▼<S> then Acepte
else Rechace

id1: if usep == b<A><S><B> then reduce(1)
else Rechace

```
id2:  if usep == b<A> then reduce(2)
      else Rechace

id3:  if usep == d<S>ca then reduce(3)
      else if usep == c<A>a then reduce(5)
           else Rechace

id4:  reduce(4)

id6:  reduce(6)
```

Las operaciones de reducción son:

Reduce(1): desapile(4), apile(<S>), retenga.
Reduce(2): desapile(2), apile(<S>), retenga.
Reduce(3): desapile(4), apile(<A>), retenga.
Reduce(4): desapile(1), apile(<A>), retenga.
Reduce(5): desapile(3), apile(<B>), retenga.
Reduce(6): desapile(1), apile(<B>), retenga.

Las transiciones shift (S: apile y avance) se determinan usando el principio de shift.

La palabra "**usep**" que se utiliza en los procesos de identificación es una forma abreviada de "**ú**ltimos **s**ímbolos **e**n la **p**ila".

La primera situación es que se hace shift cuando en el tope de la pila esté el símbolo de pila vacía (▼) y el símbolo de entrada pertenezca a **primeros(<S>)**. Como vimos, los **primeros(<S>)** es sólo el terminal **b**, por tanto, en la intersección de ▼ con **b** la transición es **S** (shift).

La segunda situación del principio de shift se aplica considerando parejas consecutivas de símbolos en el lado derecho de cada producción.

La primera pareja del lado derecho de la producción 1 es **b<A>**, por consiguiente, cuando en el tope de la pila se halla el terminal **b** y el símbolo de entrada pertenece a **primeros(<A>)**, es decir, los terminales **d** y **e**, la transición es shift.

La siguiente pareja del lado derecho de la producción 1 es **<A><S>**, por consiguiente, cuando en el tope de la pila se halle el no terminal **<A>** y el símbolo de entrada pertenece a **primeros(<S>)**, es decir, el terminal **b**, la transición es shift.

La siguiente pareja del lado derecho de la producción 1 es **<S><B>**, por tanto, cuando en el tope de la pila se halle el no terminal **<S>** y el símbolo de entrada pertenezca a primeros de **<B>**, es decir, el terminal **c**, la transición es shift.

Ya no hay más parejas en el lado derecho de la producción 1, por tanto, se continúa analizando el lado derecho de la producción 2.

En el lado derecho de la producción 2 sólo hay una pareja: **b<A>**, la cual ya fue analizada en la producción 1.

En el lado derecho de la producción 3 la primera pareja es **d<S>**, por tanto, cuando en el tope de la pila se halle el terminal **d** y el símbolo de entrada pertenezca a **primeros(<S>)** la transición es shift.

La siguiente pareja del lado derecho de la producción 3 es **<S>c**, por tanto, cuando en el tope de la pila se halle el no terminal **<S>** y el símbolo de entrada pertenezca a **primeros(c)**, es decir el terminal **c**, la transición es shift.

La siguiente pareja del lado derecho de la producción 3 es **ca**, por tanto cuando en el tope de la pila se halle el terminal **c** y el símbolo de entrada sea el terminal **a**, la transición es shift.

El lado derecho de la producción 4 no tiene pareja de símbolos.

En la producción 5 la primera pareja de símbolos es **c<A>**, por tanto, cuando en el tope de la pila se halle el terminal **c** y el símbolo de entrada pertenezca a **primeros(<A>)**, es decir, los terminales **d** y **e**, la transición es shift.

La siguiente pareja de la producción 5 es **<A>a**, por tanto, cuando en el tope de la pila se halle el no terminal **<A>** y el símbolo de entrada sea el terminal **a**, la transición es shift.

En el lado derecho de la producción 6 no hay parejas de símbolos.

La aplicación del principio de reducción es para determinar cuáles transiciones son proceso de identificación.

La primera situación es cuando en el tope de la pila se halle el símbolo inicial de la gramática, (**<S>**), y el símbolo de entrada sea el símbolo de fin de secuencia (┤). Dicho proceso de identificación lo llamamos id0.

La segunda situación del principio de reducción implica analizar cada una de las producciones de la gramática.

En la producción 1 el manejador es el no terminal **<B>** y el símbolo del lado izquierdo es el no terminal **<S>**, por tanto, cuando en el tope de la pila se halle el no terminal **<B>** y el símbolo de entrada pertenezca a **siguientes(<S>)**, es decir, el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id1**.

En la producción 2 el manejador es el no terminal **<A>** y el símbolo del lado izquierdo es el no terminal **<S>**, por tanto, cuando en el tope de la pila se halle el no terminal **<A>** y el símbolo de entrada pertenezca a **siguientes(<S>)**, es decir, el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id2**.

En la producción 3 el manejador es el terminal **a** y el símbolo del lado izquierdo es el no terminal **<A>**, por tanto, cuando en el tope de la pila se halle el terminal **a** y el símbolo de entrada pertenezca a **siguientes(<A>)**, es decir, el terminal **b** o el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id3**.

En la producción 4 el manejador es el terminal **e** y el símbolo del lado izquierdo es el no terminal **<A>**, por tanto, cuando en el tope de la pila se halle el terminal **e** y el símbolo de entrada pertenezca a **siguientes(<A>)**, es decir, el terminal **b** o el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id4**.

En la producción 5 el manejador es el terminal **a** y el símbolo del lado izquierdo es el no terminal **<B>**, por tanto, cuando en el tope de la pila se halle el terminal **a** y el símbolo de entrada pertenezca a **siguientes(<B>)**, es decir, el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id3**. Fíjese que en las producciones 3 y 5 el manejador es el mismo símbolo (el terminal **a**). Cuando esto sucede sólo se hace un proceso de identificación por manejador.

En la producción 6 el manejador es el terminal **c** y el símbolo del lado izquierdo es el no terminal **<B>**, por tanto, cuando en el tope de la pila se halle el terminal **c** y el símbolo de entrada pertenezca a **siguientes(<B>)**, es decir, el terminal **c** o el símbolo de fin de secuencia (┤), la transición es un proceso de identificación, el cual se ha llamado **id6**.

12.7 Solución de conflictos shift-identify.

Una situación que se puede presentar es un conflicto shift-identify. Un conflicto shift-identify es una situación en la cual, para un mismo símbolo de entrada y un mismo símbolo en el tope de la pila, las transiciones resultantes al aplicar los principios de shift y de reducción son S (apile y avance) y proceso de identificación. Consideremos la siguiente gramática.

1. <S> → c<A>a
2. <A> → c<B>
3. <A> → d<B>a
4. <B> → d<S>e<B>f
5. <B> → g

Para construir el autómata de pila para el reconocimiento ascendente, técnica shift-identify, los pasos son:

Primeros(<S>) = {c}
Primeros(<A>) = {c, d}
Primeros(<B>) = {d, g}

Siguientes(<S>) = {e, ┤}
Siguientes(<A>) = {a}
Siguientes(<B>) = {a, f}

Símbolos de entrada = {a, c, d, e, f, g, ┤}
Símbolos en la pila = { <S>, <A>, <B>, a, c, d, e, f, g, ▼}
Configuración inicial de la pila = ▼
Tabla de transiciones:

	a	c	d	e	f	g	┤
<S>				S			id0
<A>	S						
<B>	S, id2				S		
a	id1			id1			id1

c		S	S			S	
d		S	S			S	
e			S			S	
f	id4						
g	id5						
▼		S					

Tabla 12.2

Dichas transiciones se han construido aplicando los principios de shift y de reducción.

Como se podrá observar, la transición correspondiente a cuando en el tope de la pila se encuentre el no terminal **<B>** y el símbolo de entrada sea el terminal **a** tiene dos opciones: según el principio de shift, la transición debe ser shift, y según el principio de reducción, la transición debe ser un proceso de identificación.

Esta situación se denomina **conflicto shift-identify**, y el **símbolo del conflicto es el símbolo en el tope de la pila**, en nuestro caso el no terminal **<B>**.

Estas situaciones se deben evitar ya que no se garantiza que el proceso de reconocimiento funcione adecuadamente. Transformar la gramática para evitar este conflicto es bastante simple.

Teniendo identificado el símbolo del conflicto lo que se hace es:

1. Reemplazar el símbolo del conflicto por un nuevo no terminal, en el lado derecho de las producciones en las cuales el símbolo del conflicto no sea manejador.

2. Agregar una nueva producción, cuyo lado izquierdo sea el nuevo no terminal definido, y el lado derecho el símbolo del conflicto.

Aplicando estos pasos en nuestra gramática, ella queda así:

1. <S> → c<A>a
2. <A> → c<B>
3. <A> → d<NB>a
4. <B> → d<S>e<NB>f
5. <B> → g
6. <NB> → <B>

Fíjese, que el símbolo del conflicto, que en nuestro caso es el no terminal <B>, se ha reemplazado en las producciones 3 y 4 por un nuevo no terminal, el cual se ha llamado <NB>. Se ha hecho sólo en las producciones 3 y 4 porque en esas producciones el no terminal <B> no es manejador, en cambio en la producción 2 se ha conservado el no terminal <B>, puesto que en dicha producción el no terminal <B> es manejador.

Teniendo definida esta nueva gramática veamos cómo queda el autómata de pila.

Primeros(<S>) = {c}
Primeros(<A>) = {c, d}
Primeros(<B>) = {d, g}

Siguientes(<S>) = {e, ┤}
Siguientes(<A>) = {a}
Siguientes(<B>) = {a, f}

Primeros(<NB>) = {d, g} Siguientes(<NB>) = {a, f}

Símbolos de entrada = {a, c, d, e, f, g, ┤}

Símbolos en la pila = {<S>, <A>, <B>, <NB>, a, c, d, e, f, g, ▼}

Configuración inicial de la pila = ▼

Tabla de transiciones:

	a	c	d	e	f	g	┤
<S>				S			id0
<A>	S						
<B>	id2				id2		
<NB>	S				S		
a	id1			id1			id1
c		S	S			S	
d		S	S			S	
e			S			S	
f	id4						
g	id5						
▼		S					

Tabla 12.3

Observe que las transiciones shift (S: apile y avance) sólo aparecen para el nuevo no terminal definido, que para nuestro ejemplo es el no terminal <NB>.

EJERCICIOS PROPUESTOS

Construya reconocedor ascendente, técnica SHIFT-IDENTIFY para las siguientes gramáticas:

1.
    1. <S> → e<S><S>a
    2. <S> → f<S>b
    3. <S> → g<S><S><S>c
    4. <S> → d

2.
    1. <S> → a<S><A>1
    2. <S> → a<A>1
    3. <S> → a<S>0
    4. <A> → b<A><S>0
    5. <A> → b<A><A>1
    6. <A> → ab0

3.
    1. <S> → 1<S>1
    2. <S> → 0<S>0
    3. <S> → 2

4.
    1. <S> → (<S>.<S>)
    2. <S> → a

SOLUCIONES

1. Construya reconocedor ascendente, técnica SHIFT-IDENTIFY para la siguiente gramática:

1. <S> → e<S><S>a
2. <S> → f<S>b
3. <S> → g<S><S><S>c
4. <S> → d

Solución:

Primeros(<S>) = {d, e, f, g}
Siguientes(<S>) = {a, b, c, d, e, f, g, ┤}

Símbolos de entrada = {a, b, c, d, e, f, g, ┤}
Símbolos en la pila = {<S>, a, b, c, ,d , e, f, g, ▼}
Configuración inicial de la pila: ▼
Tabla de transiciones:

	a	b	c	d	e	f	g	┤
<S>	S	S	S	S	S	S	S	id0
a	id1	id1	id1	id1	id1	id1	id1	id1
b	id2	id2	id2	id2	id2	id2	id2	id2
c	id3	id3	id3	id3	id3	id3	id3	id3
d	id4	id4	id4	id4	id4	id4	id4	id4
e				S	S	S	S	
f				S	S	S	S	
g				S	S	S	S	
▼				S	S	S	S	

S: apile, avance

```
id0:  if usep == <S> then
            Acepte
      else
            Rechace

id1:  if usep == e<S><S>a  then
            reduce(1)
      else
            Rechace
id2:  if usep == f<S>b  then
            reduce(2)
      else
            Rechace
id3:  if usep == g<S><S><S>c  then
            reduce(3)
      else
            Rechace
id4:  reduce(4)
```

reduce(1): desapile(4), apile(<S>), retenga.

reduce(2): desapile(3), apile(<S>), retenga.
reduce(3): desapile(5), apile(<S>), retenga.
reduce(4): desapile(1), apile(<S>), retenga.

3. Construya reconocedor ascendente, técnica SHIFT-IDENTIFY para la siguiente gramática:

1. <S> → 1<S>1
2. <S> → 0<S>0
3. <S> → 2

Solución:

Primeros(<S>) = {0, 1, 2}
Siguientes(<S>) = {0, 1, ┤}

Símbolos de entrada = {0, 1, 2, ┤}
Símbolos en la pila = {<S>, 0, 1, 2, ▼}
Configuración inicial de la pila: ▼
Tabla de transiciones:

	0	1	2	┤
<S>	S	S		id0
0	S, id2	S, id2	S	id2
1	S, id1	S, id1	S	id1
2	id3	id3		id3
▼	S	S	S	

Como se puede ver, hay conflictos SHIFT-IDENTIFY cuando en el tope de la pila se halla un 0 y el símbolo de entrada es un 0 o un 1, y cuando en el tope de la pila se halla un 1 y el símbolo de entrada es un 0 o un 1.

Para resolver estos conflictos usamos la técnica descrita en 12.7. La gramática queda:

1. <S> → <U><S>1
2. <S> → <C><S>0
3. <S> → 2
4. <U> → 1
5. <C> → 0

Con esta nueva gramática tenemos:

Primeros(<S>) = {0, 1, 2} Siguientes(<S>) = {0, 1, ┤}
Primeros(<U>) = {1} Siguientes(<U>) = {0, 1, 2}
Primeros(<C>) = {0} Siguientes(<C>) = {0, 1, 2}

Símbolos de entrada = {0, 1, 2, ┤}
Símbolos en la pila = {<S>, <U>, <C>, 0, 1, 2, ▼}
Configuración inicial de la pila: ▼
Tabla de transiciones:

	0	1	2	⊣
<S>	S	S		id0
<U>	S	S	S	
<C>	S	S	S	
0	id2	id2		id2
1	id1	id1		id1
2	id3	id3		id3
▼	S	S	S	

S: apile, avance
id0: if usep == ▼<S> then
Acepte
else
Rechace
id1: if usep == <U><S>1 then
reduce(1)
else
reduce(4)
id2: if usep == <C><S>0 then
reduce(2)
else
reduce(5)
id3: reduce(3)

reduce(1): desapile(3), apile(<S>), avance.
reduce(2): desapile(3), apile(<S>), avance.
reduce(3): desapile(1), apile(<S>), avance.
reduce(4): desapile(1), apile(<S>), avance.
reduce(5): desapile(1), apile(<S>), avance.

MODULO 13
GRAMATICAS DE PRECEDENCIA DEBIL Y MIXTA

13.1 INTRODUCCIÓN
Las gramáticas que hemos venido tratando hasta ahora se denominan gramáticas independientes del sufijo, es decir, gramáticas en las cuales el lado derecho de todas las producciones son diferentes. Veremos en este módulo gramáticas en las cuales el lado derecho de alguna producción es sufijo del lado derecho de otra producción y gramáticas que tienen más de una producción cuyo lado derecho es exactamente el mismo.

13.2 OBJETIVOS
1. Identificar cuándo una gramática es de precedencia débil.
2. Identificar cuándo una gramática es de precedencia mixta.
3. Determinar si se puede construir el reconocedor ascendente de pila en una gramática de precedencia débil.
4. Determinar si se puede construir el reconocedor ascendente de pila en una gramática de precedencia mixta.

13.3 PREGUNTAS BÁSICAS
1. Qué es una gramática de precedencia débil.
2. Qué condición se debe cumplir para poder construir el reconocedor ascendente en una gramática de precedencia débil.
3. Qué es una gramática de precedencia mixta.
4. Qué condición se debe cumplir para poder construir el reconocedor ascendente en una gramática de precedencia mixta.
5. Qué se debe tener en cuenta en los procesos de identificación en gramáticas de precedencia débil.
6. Qué se debe tener en cuenta en los procesos de identificación en gramàticas de precedencia mixta.

13.4 Gramáticas de precedencia débil
Son gramáticas en las cuales el lado derecho de alguna producción es sufijo del lado derecho de otra producción. Simbólicamente, existen dos producciones de la forma:

i. <A> → α**x**β
j. <B> → β

con α en (T + N)*, β en (T + N)$^+$ y **x** un símbolo cualquiera, terminal o no terminal.

Por ejemplo, las producciones:

k. <S> → a<A>b<B>c<D>
l. <T> → <B>c<D>

tienen esa característica. En esas dos producciones α es la hilera **a<A>**, β es la hilera **<B>c<D>** y **x** es el terminal **b**. En esas dos producciones se debe cumplir la condición de que el terminal **b** no puede estar debajo del no terminal **<T>**.

Cuando se tenga una gramática con esta característica, se podrá construir el reconocedor ascendente de pila, técnica shift-identify, sí y sólo sí, es falso que

x *estáDebajoDe* **<B>**

Recuerde que la relación está debajo de se refiere a la ubicación consecutiva de los símbolos en la pila.

Lo anterior implica que cuando se tenga una gramática con esa característica se debe controlar que **x** no esté debajo de **<B>** para poder proceder a construir el autómata de pila.

Además, cuando se vaya a efectuar el proceso de identificación, se deberá preguntar primero por la hilera de mayor longitud. Es decir, para las producciones i. y j. que tenemos en la representación simbólica, el proceso de identificación será:

```
if usep == αxβ then reduce(i)
else if usep == β then reduce(j)
       else Rechace
```

Consideremos una de nuestras gramáticas para generar expresiones aritméticas.

1. <E> → <E> + <T>
2. <E> → <T>
3. <T> → <T> * <P>
4. <T> → <P>
5. <P> → (<E>)
6. <P> → I

Esta gramática tiene esa característica: el lado derecho de la producción 2 es sufijo del lado derecho de la producción 1 y el lado derecho de la producción 4 es sufijo del lado derecho de la producción 3.

En las dos primeras producciones se debe cumplir que el símbolo + no esté debajo del no terminal <E>, y en las producciones 3 y 4 se debe cumplir que el símbolo * no esté debajo del no terminal <T>.

Para controlar estas condiciones se debe construir la relación ***estáDebajoDe*** para los símbolos en la pila de la gramática. Para construir la relación ***estáDebajoDe*** debemos construir primero el conjunto de primeros para cada no terminal de la gramática. Estos conjuntos son:

Primeros(<E>) = {<E>, <T>, <P>, (, I}
Primeros(<T>) = {<T>, <P>, (, I}
Primeros(<P>) = {(, I}

Los símbolos en la pila son todos los terminales y no terminales de la gramática más el símbolo de pila vacía (▼).

Para nuestro ejemplo la tabla que representa la relación *estáDebajoDe* es:

	<E>	<T>	<P>	+	*	(	I	)
▼	1	1	1			1	1	
<E>				1				1
<T>					1			
<P>								
+		1	1			1	1	
*			1			1	1	
(	1	1	1			1	1	
I								
)								

Tabla 13.1

La construcción de esta relación es algo simple: el símbolo de pila vacía se halla debajo del símbolo inicial de la gramática y de los primeros del símbolo inicial de la gramática; luego se analiza el lado derecho de cada producción considerando parejas consecutivas de símbolos, digamos AB, entonces, A *estáDebajoDe* los primeros de B.

Teniendo construida la relación *estáDebajoDe* se obtiene el conjunto de símbolos que están debajo de cada no terminal.

Los símbolos que están debajo de cada no terminal son los símbolos de la fila correspondiente a cada uno de los unos que se hallan en las columnas de los no terminales. En nuestro ejemplo:

EstánDebajoDe(<E>) = {(, ▼}
EstánDebajoDe(<T>) = {(, +, ▼}
EstánDebajoDe(<P>) = {(, +, *, ▼}

Como podrá observar, se cumplen las condiciones: el + no está debajo del no terminal <E>, y el * no está debajo del no terminal <T>.

Por consiguiente se podrá construir el autómata de pila con el cual se hace el reconocimiento ascendente, técnica shift-identify, de dicha gramática.

Construyamos primero el conjunto de siguientes de cada no Terminal para poder aplicar el principio de reducción:

Siguientes(<E>) = {+,), ┤}
Siguientes(<T>) = {+, *,), ┤}
Siguientes(<P>) = {+, *,), ┤}

El autómata es:
Símbolos de entrada = {+, *, (, I,), ┤}
Símbolos en la pila = {<E>, <T>, <P>, +, *, (, I,), ▼}
Configuración inicial de la pila = ▼
Tabla de transiciones:

	+	*	(	I	)	┤
<E>	S				S	id0
<T>	id1	S			id1	id1
<P>	id3	id3			id3	id3
+			S	S		

*			S	S		
(			S	S		
I	id6	id6			id6	id6
)	id5	id5			id5	id5
▼			S	S		

Tabla 13.2

S: apile, avance.
id0: if usep == ▼ then acepte
else Rechace
id1: if usep == <E>+<T> then reduce(1)
else reduce(2)
id3: if usep == <T>*<P> then reduce(3)
else reduce(4)
id5: if usep == (<E>) then reduce(5)
else Rechace
id6: reduce(6)

reduce(1): desapile(3), apile(<E>), retenga.
reduce(2): desapile(1), apile(<E>), retenga.
reduce(3): desapile(3), apile(<T>), retenga.
reduce(4): desapile(1), apile(<T>), retenga.
reduce(5): desapile(3), apile(<P>), retenga.
reduce(6): desapile(1), apile(<P>), retenga.
Es bastante importante hacer notar que en aquellas producciones en las cuales el lado derecho de una de ellas es sufijo del lado derecho de la otra producción (producciones 1 y 2, y producciones 3 y 4), el proceso de identificación se efectuó preguntando primero por la hilera de mayor longitud.

13.5 Gramáticas de precedencia mixta

Son gramáticas en las cuales existe más de una producción cuyo lado derecho es el mismo. Simbólicamente, existen dos producciones de la forma:

i. <A> → α
j. <B> → α

con α en $(T + N)^+$.

Cuando se tiene una gramática con esta característica, se podrá construir el autómata de pila para reconocimiento ascendente, técnica shift-identify, sí y sólo sí, no existe algún símbolo **x** tal que **x *estáDebajoDe* <A>** y **x *estáDebajoDe* <B>**.

En otras palabras, el conjunto de los que están debajo de <A> debe ser disyunto con el conjunto de los que están debajo de <B>, es decir, su intersección debe ser el conjunto vacío.

Consideremos la siguiente gramática:

1. <S> → <B>v
2. <S> → v<C>
3. <A> → u

4. <A> → v<B><S>
5. <B> → u
6. <B> → yw
7. <C> → <B>v
8. <C> → y<A>w

Esta gramática tiene la característica que hemos descrito: producciones 1 y 7 tienen el mismo lado derecho, igual sucede con las producciones 3 y 5.

Debemos determinar si se puede construir el reconocedor ascendente de pila, técnica shift-identify. Para ello debemos averiguar si los símbolos que están debajo del no terminal <S> son diferentes de los símbolos que están debajo del no terminal <C>, y que los símbolos que están debajo del no terminal <A> son diferentes de los símbolos que están debajo del no terminal <B>. Para construir el conjunto de los símbolos que están debajo de los no terminales debemos construir la relación *estáDebajoDe* de la gramática. Para construir la relación *estáDebajoDe* comenzamos construyendo el conjunto de primeros de cada no terminal.

Primeros(<S>) = {<B>, v, u, y}
Primeros(<A>) = {u, v}
Primeros(<B>) = {u, y}
Primeros(<C>) = {<B>, y, u}

Teniendo construidos los conjuntos de primeros de cada no terminal construyamos la relación *estáDebajoDe*: los símbolos en la pila son los terminales, los no terminales y el símbolo de pila vacía (▼).

	<S>	<A>	<B>	<C>	u	v	w	y
▼	1		1		1	1		1
<S>								
<A>							1	
<B>	1		1		1	1		1
<C>								
u								
v			1	1	1			1
w								
y		1			1	1	1	

Tabla 13.3

El símbolo de pila vacía (▼) está debajo del símbolo inicial de la gramática, el no terminal <S>, y de los primeros del símbolo inicial de la gramática: esos son los unos de la fila correspondiente al símbolo de pila vacía (▼).

Luego se analiza el lado derecho de cada producción considerando parejas consecutivas de símbolos: en la producción 1 el no terminal **<B>** está antes del terminal **v**, por tanto, el no terminal **<B>** está debajo del terminal **v**; en la producción 2 el terminal **v** está debajo del no terminal **<C>** y de los primeros de **<C>**; en la producción 3 no hay parejas de símbolos; en la producción 4 el no terminal **v** está debajo del no terminal **<B>** y de los primeros de <B>, y el no terminal **<B>** está debajo del no terminal **<S>** y de los primeros de **<S>**; en la producción 5 no hay parejas de símbolos; en la producción 6 el terminal **y** está

debajo del terminal **w**; en la producción 7 el no terminal **<B>** está debajo del terminal **v**, y en la producción 8 el terminal **y** está debajo del no terminal **<A>**, y el no terminal **<A>** está debajo del terminal **w**.

Teniendo construida la relación *estáDebajoDe* se construye el conjunto de *estánDebajoDe* para cada no terminal. Recuerde que los símbolos correspondientes a cada uno de cada columna son los que están debajo del símbolo correspondiente a cada columna.

estánDebajoDe(<S>) = {▼, <B>}
estánDebajoDe(<A>) = {y}
estánDebajoDe(<B>) = {▼, <B>, v}
estánDebajoDe(<C>) = {v}

Construidos estos conjuntos determinaremos si cumplen la condición: por producciones 1 y 7 los símbolos que está debajo del no terminal <S> deben ser diferentes de los símbolos que está debajo del no terminal <C>, cumplen la condición; por producciones 3 y 5 los símbolos que stán debajo del no terminal <A> deben ser diferentes de los símbolos que están debajo del no terminal <B>, también cumplen la condición. Por consiguiente, se podrá construir el reconocedor ascendente de pila, técnica shift-identify, para la gramática dada.

Ahora, con el fin de construir el autómata de pila, se construye el conjunto de siguientes de cada no terminal:

Siguientes(<S>) = {┤, w}
Siguientes(<A>) = {w}
Siguientes(<B>) = {u, v, y}
Siguientes(<C>) = {w, ┤}

El autómata de pila es:

Símbolos de entrada = {u, v, w, y, ┤}
Símbolos en la pila = {<S>, <A>, <B>, <C>, u, v, w, y, ┤}
Configuración inicial de la pila = ▼
Tabla de transiciones:

	u	v	w	y	┤
<S>			id4		id0
<A>			S		
<B>	S	S		S	
<C>			id2		id2
u	id3	id3	id3	id3	
v	S		id1	S	id1
w	id6	id6	id6	id6	id6
y	S	S	S		
▼	S	S		S	

Tabla 13.4

Consideremos ahora los procesos de identificación, especialmente para aquellas producciones cuyo lado derecho es el mismo.

Cuando en el tope de la pila se halle el terminal **v**, el cual es el manejador de las producciones 1 y 7, se debe determinar si se reduce por la producción 1 o por la producción 7. Si se reduce por la producción 1, en el tope de la pila quedará el no terminal <S>, pero si se reduce por la producción 7, en el tope de la pila quedará el no terminal <C>. En ambos casos los símbolos a desapilar son dos. Si se apila el no terminal <S>, el símbolo que debe quedar debajo de la <S> es uno de los símbolos que pertenece al conjunto de los que *estánDebajoDe* <S>, pero si se apila el no terminal <C> el símbolo que debe quedar debajo de <C> es alguno de los símbolos que pertenece al conjunto de los que *estánDebajoDe* <C>. Por consiguiente, cuando se haga el proceso de identificación se deberán concatenar los símbolos que están debajo de <S> y <C> con los símbolos del lado derecho de las producciones cuyo lado derecho es el mismo, con el fin de determinar cuál es la producción correcta por la que se reduce.

Gráficamente, las diferentes situaciones que se pueden presentar, se muestran en la figura a continuación (figura 13.1).

Si la situación es como en el caso (a) ó en el caso (b) de la figura 13.1, se deberá reducir por la producción 1, en cambio si la situación es como en el caso (c) de la misma figura, se deberá reducir por la producción 7.

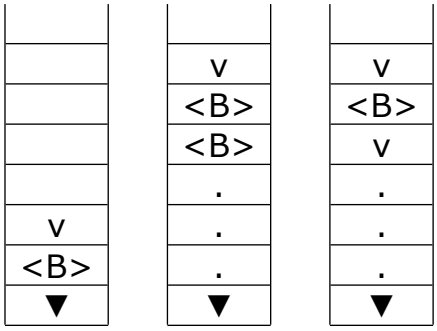


Figura 13.1

Consideremos ahora las producciones 3 y 5. Cuando en el tope de la pila se halle el terminal u, debemos determinar si se reduce por la producción 3 o por la producción 5. Si se reduce por la producción 3, en el tope de la pila quedará el no terminal <A>, pero, si se reduce por la producción 5, en el tope de la pila quedará el no terminal <B>. Si en el tope de la pila ubicamos el no terminal <A>, el símbolo que quede debajo del no terminal <A> debe ser alguno de los símbolos que pertenece al conjunto *estánDebajoDe(<A>)*, en cambio, si en el tope de la pila ubicamos el no terminal <B>, el símbolo que quede debajo debe ser alguno de los símbolos que pertenece al conjunto *estánDebajoDe(<B>)*. Por consiguiente, cuando se haga el proceso de identificación se deberán concatenar los símbolos que están debajo de <A> y <B> con el terminal **u**, con el fin de determinar cuál es la producción correcta por la que se reduce.

Gráficamente, las diferentes situaciones que se pueden presentar, se muestran en la figura a continuación (figura 13.2).

Si la situación es como en el caso (a) de la figura 13.2, se deberá reducir por la producción 3, pero, si la situación es como en los casos (b), (c) ó (d) de la misma figura, se deberá reducir por la producción 5.

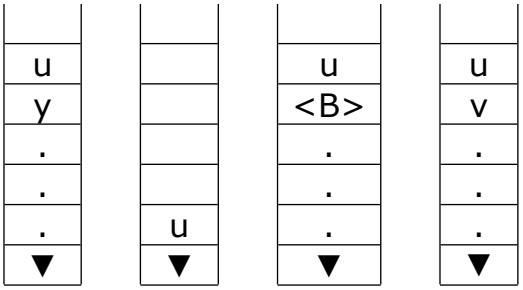


Figura 13.2

Dado lo anterior, los procesos de identificación son:

id0: if usep == ▼<S> then Acepte
else Rechace.

id1: if usep == ▼<B>v or usep == <B><B>v then reduce(1)
else if usep == v<B>v then reduce(7)
else Rechace

id2: if usep == v<C> then reduce(2)
else Rechace

id3: if usep == yu then reduce(3)
else if usep == ▼u **or** usep == <B>u **or** usep == vu then reduce(5)
else Rechace

id4: if usep == v<B><S> then reduce(4)
else Rechace

id6: if usep == yw then reduce(6)
else if usep == y<A>w then reduce(8)
else Rechace

Los procesos de reducción son:

reduce(1): desapile(2), apile(<S>), retenga
reduce(2): desapile(2), apile(<S>), retenga
reduce(3): desapile(1), apile(<A>), retenga
reduce(4): desapile(3), apile(<A>), retenga
reduce(5): desapile(1), apile(<B>), retenga
reduce(6): desapile(2), apile(<B>), retenga
reduce(7): desapile(2), apile(<C>), retenga
reduce(8): desapile(3), apile(<C>), retenga

EJERCICIOS PROPUESTOS

Construya reconocedor ascendente de pila, técnica SHIFT-IDENTIFY, para las siguientes gramáticas:

1.
    1. <S> → <S><A>
    2. <S> → <A>
    3. <A> → 1<S>0
    4. <A> → 10

2.
    1. <P> → <B>
    2. <P> → <CS>
    3. <B> → <BH>;<CT>
    4. <BH> → bd
    5. <BH> → <BH>;d
    6. <CS> → b<CT>
    7. <CT> → se
    8. <CT> → s;<CT>

3.
    1. <S> → a<A>
    2. <S> → b<B>
    3. <A> → <C><A>1
    4. <A> → <C>1
    5. <B> → <D><B><E>1
    6. <B> → <D><E>1
    7. <C> → 0
    8. <D> → 0
    9. <E> → 1

SOLUCIONES

1. Construya reconocedor ascendente de pila, técnica SHIFT-IDENTIFY, para la siguiente gramática:

1. <S> → <S><A>
2. <S> → <A>
3. <A> → 1<S>0
4. <A> → 10

Solución:

Como se puede observar, el lado derecho de la producción 2 es sufijo del lado del lado derecho de la producción 1, por consiguiente debemos controlar que se cumpla la condición descrita en el numeral 13.4, según la cual, para esta gramática, se debe cumplir que <S> no esté debajo de <S>. Para ello comenzamos construyendo la relación estaDebajoDe para los símbolos de la gramática. Dicha relación se presenta en la tabla 13.5.

	<S>	<A>	0	1
<S>		1	1	1
<A>				
0				
1	1		1	1
▼	1	1		1

Tabla 13.5

Con base en esta tabla, los conjuntos estánDebajoDe son:

estánDebajoDe(<S>) = {1, ▼}
estánDebajoDe(<A>) = {<S>, ▼}

Por consiguiente, nuestra gramática cumple la condición. Procedamos a construir el autómata de pila que haga el reconocimiento ascendente, técnica SHIFT-IDENTIFY, del lenguaje que genera dicha gramática.

Primeros(<S>) = {1}
Primeros(<A>) = {1}

Siguientes(<S>) = {0, 1, ┤}
Siguientes(<A>) = {0, 1, ┤}

Símbolos de entrada = {0, 1, ┤}
Símbolos en la pila = {<S>, <A>, 0, 1, ▼}

Configuración inicial de la pila = ▼
Transiciones:

	0	1	⊣
<S>	S	S	id0
<A>	id1	id1	id1
0	id2	id2	id2
1	S	S	
▼		S	

S: Apile, avance.
id0: if usep == ▼<S> then
Acepte
else
Rechace

id1: if usep == <S><A> then reduce(1)
else reduce(2)

id2: if usep == 1<S>0 then reduce(3)
else if usep == 10 then reduce(4)

reduce(1): desapile(2), apile(<S>), retenga
reduce(2): desapile(1), apile(<S>), retenga
reduce(3): desapile(3), apile(<A>), retenga
reduce(4): desapile(2), apile(<A>), retenga

3. Construya reconocedor ascendente de pila, técnica SHIFT-IDENTIFY, para la siguiente gramática:

1. <S> → a<A>
2. <S> → b<B>
3. <A> → <C><A>1
4. <A> → <C>1
5. <B> → <D><B><E>1
6. <B> → <D><E>1
7. <C> → 0
8. <D> → 0
9. <E> → 1

Solución:

Como se puede observar, el lado derecho de la producción 9 es sufijo del lado del lado derecho de las producciones 3, 4, 5 y 6, y además, el lado derecho de las producciones 7 y 8 son iguales, por consiguiente debemos controlar que se cumplan la condiciones descritas en los numerales 13.4 y 13.5, según las cuales, para esta gramática, se debe cumplir: que los no terminales <A>, <C> y <E> no estén debajo de <E>, y que los símbolos debajo de <C> y <D> sean diferentes. Para ello comenzamos construyendo la relación estaDebajoDe para los símbolos de la gramática. Dicha relación se presenta en la tabla 13.6.

	<S>	<A>	<B>	<C>	<D>	<E>	a	b	0	1
<S>										
<A>										1
<B>						1				1
<C>		1		1					1	1
<D>			1		1	1			1	1
<E>										1
a		1		1					1	
b			1		1				1	
0										
1										
▼	1						1	1		

Tabla 13.6

Con base en esta tabla, los conjuntos estánDebajoDe son:

estánDebajoDe(<S>) = {▼}
estánDebajoDe(<A>) = {<C>, a}
estánDebajoDe(<B>) = {<D>, b}
estánDebajoDe(<C>) = {<C>, a}
estánDebajoDe(<D>) = {<D>, b}
estánDebajoDe(<E>) = {<B>, <D>}

Por consiguiente, nuestra gramática cumple las condiciones. Procedamos a construir el autómata de pila que haga el reconocimiento ascendente, técnica SHIFT-IDENTIFY, del lenguaje que genera dicha gramática.

Primeros(<S>) = {a, b}
Primeros(<A>) = {<C>, 0}
Primeros(<B>) = {<D>, 0}
Primeros(<C>) = {0}
Primeros(<D>) = {0}
Primeros(<E>) = {1}

Siguientes(<S>) = {┤}
Siguientes(<A>) = {1, ┤}
Siguientes(<B>) = {1, ┤}
Siguientes(<C>) = {0, 1}
Siguientes(<D>) = {0, 1}
Siguientes(<E>) = {1}

Símbolos de entrada = {a, b, 0 , 1, ┤}
Símbolos en la pila = {<S>, <A>, <B>, <C>, <D>, <E>, a, b, 0, 1, ▼}

Configuración inicial de la pila = ▼
Transiciones:

	a	b	0	1	⊣
<S>					id0
<A>				S	id1
<B>				S	id2
<C>			S	S	
<D>			S	S	
<E>				S	
a			S		
b			S		
0			id4	id4	
1				id3	id3
▼	S	S			

S: Apile, avance.

id0: if usep == ▼<S> then
Acepte
else
Rechace

id1: if usep == a<A> then reduce(1)
else Rechace

id2: if usep == b<B> then reduce(2)
else Rechace

id3: if usep == <C><A>1 then reduce(3)
else if usep == <C>1 then reduce(4)
else if usep == <D><B><E>1 then reduce(5)
else if usep == <D><E>1 then reduce(6)
else reduce(9)

id4: if usep == <C>0 or a0 then reduce(7)
else if usep == <D>0 or b0 then reduce(8)
else Rechace

reduce(1): desapile(2), apile(<S>), retenga
reduce(2): desapile(2), apile(<S>), retenga
reduce(3): desapile(3), apile(<A>), retenga
reduce(4): desapile(2), apile(<A>), retenga
reduce(5): desapile(4), apile(<B>), retenga
reduce(6): desapile(3), apile(<B>), retenga
reduce(7): desapile(1), apile(<C>), retenga
reduce(8): desapile(1), apile(<D>), retenga
reduce(9): desapile(1), apile(<E>), retenga

MODULO 14
RECONOCIMIENTO ASCENDENTE, TECNICA SHIFT-REDUCE

14.1 INTRODUCCIÓN
En los módulos anteriores se trató el método de reconocimiento ascendente, técnica shift-identify, en la cual, como vimos hay que hacer un proceso de identificación cuando se detecte que en el tope de la pila hay un manejador. Este proceso de identificación hace que el proceso de reconocimiento se vuelva un poco lento. Veremos en este módulo cómo evitar el proceso de identificación.

14.2 OBJETIVOS
1. Construir autómata de pila que efectúe reconocimiento ascendente del lenguaje generado por una gramática sin tener que hacer procesos de identificación.

14.3 PREGUNTAS BÁSICAS
1. Qué es una ocurrencia gramatical.
2. Cuál es la función de una ocurrencia gramatical.
3. Con qué fin se construye la relación estáDebajoDe para las ocurrencias gramaticales.

14.4 Conceptos básicos.
La idea básica de la técnica shift-reduce es evitar los procesos de identificación, ya que éstos generan ineficiencia en tiempo de ejecución. Recordemos que la idea básica del reconocimiento ascendente es construir en la pila el lado derecho de alguna producción, para luego reducir por esa producción e ir efectuando el reconocimiento.

Con el fin de evitar los procesos de reconocimiento introducimos un nuevo concepto, el cual es el de ocurrencia gramatical.

Una ocurrencia gramatical es la presencia de un símbolo gramatical en el lado derecho de alguna producción.

Las ocurrencia gramaticales son los símbolos que se llevarán a la pila, de tal manera que en todo momento se sepa cuál es la producción cuyo lado derecho es el que se está construyendo.

14.5 Un ejemplo.
Consideremos la siguiente gramática.

1. <S> → a<A>b
2. <S> → c
3. <A> → b<S>
4. <A> → <B>b
5. <B> → a<A>
6. <B> → c

El terminal **a** aparece en el lado derecho de las producciones 1 y 5. Cuando se esté haciendo un proceso de reconocimiento y el símbolo de entrada sea el terminal **a** habrá que apilar una ocurrencia gramatical que identifique si se va a construir el lado derecho de la producción 1 o el lado derecho de la producción 5. Lo anterior implica establecer cierta codificación: llamaremos $\mathbf{a_1}$ la ocurrencia gramatical que identifique que se va a construir el lado derecho de la producción 1, y $\mathbf{a_5}$ la ocurrencia gramatical

que identifique que se va a construir el lado derecho de la producción 5. El terminal **b** aparece en el lado derecho de las producciones 1, 3 y 4, por consiguiente, debemos diferenciar cada una de dichas bees: llamaremos $\mathbf{b_1}$ la ocurrencia gramatical correspondiente a la **b** de la producción 1; $\mathbf{b_3}$ la ocurrencia gramatical correspondiente a la **b** de la producción 3, y $\mathbf{b_4}$ la ocurrencia gramatical correspondiente a la **b** de la producción 4. Teniendo en cuenta lo anterior, las ocurrencias gramaticales de la gramática anterior son:

1. <S> → a<A>b $a_1<A>_1b_1$
2. <S> → c c_2
3. <A> → b<S> $b_3<S>_3$
4. <A> → <B>b $<B>_4b_4$
5. <B> → a<A> $a_5<A>_5$
6. <B> → c c_6

Además, debemos definir una ocurrencia gramatical que determine el estado de aceptación. Llamemos esta ocurrencia gramatical $<S>_0$, la cual evita el último proceso de identificación. Cuando en el tope de la pila se halle la ocurrencia gramatical $<S>_0$ significa que los símbolos en la pila son ▼<S>, y si el símbolo de entrada es el símbolo de fin de secuencia (┤) entonces se acepta la hilera de entrada.

Teniendo definidas las ocurrencias gramaticales, se debe construir una tabla de apilamiento que permita determinar cuál ocurrencia gramatical es la que hay que llevar a la pila. Esta determinación se hace con base en la ocurrencia gramatical que haya en el tope de la pila y el símbolo de entrada a la pila.

Para construir esta tabla de apilamiento se debe construir primero la relación *estáDebajoDe* para las ocurrencias gramaticales. Como ya hemos visto en el módulo anterior, para construir la relación *estáDebajoDe*, es necesario tener el conjunto de primeros de cada no terminal.

Primeros(<S>) = $\{a_1, c_2\}$
Primeros(<A>) = $\{b_3, <B>_4, a_5, c_6\}$
Primeros(<B>) = $\{a_5, c_6\}$

La relación *estáDebajoDe* para las ocurrencias gramaticales es:

	$<S>_0$	a_1	$<A>_1$	b_1	c_2	b_3	$<S>_3$	$<B>_4$	b_4	a_5	$<A>_5$	c_6
▼	1	1			1							
$<S>_0$												
a_1			1			1		1		1		1
$<A>_1$				1								
b_1												
c_2												
b_3		1			1		1					
$<S>_3$												
$<B>_4$									1			
b_4												
a_5						1		1		1	1	1
$<A>_5$												
c_6												

Tabla 14.1

Teniendo construida la relación estáDebajoDe se continúa construyendo la tabla de apilamiento, con base en esta relación.

La tabla de apilamiento es una tabla en la cual las filas son las ocurrencias gramaticales y las columnas son los símbolos gramaticales. Con esta tabla se especifica cuál ocurrencia gramatical se debe apilar dependiendo de la ocurrencia gramatical que se halle en el tope de la pila y del símbolo gramatical que haya que apilar.

La tabla de apilamiento se presenta en la tabla 14.2

	<S>	<A>	<B>	a	b	c	Acción
▼	$<S>_0$	$<A>_1$				c_2	continúe
$<S>_0$							acepte
a_1		$<A>_1$	$<B>_4$	a_5	b_3	c_6	continúe
$<A>_1$				a_1			continúe
b_1							reduce(1)
c_2							reduce(2)
b_3	$<S>_3$			a_1		c_2	continúe
$<S>_3$							reduce(3)
$<B>_4$					b_4		continúe
b_4							reduce(4)
a_5		$<A>_5$	$<B>_4$	a_5	b_3	c_6	continúe
$<A>_5$							reduce(5)
c_6							reduce(6)

Tabla 14.2

La tabla de apilamiento se trata como si fuera un autómata finito. Un autómata finito reconocedor de manejadores, cuyo estado inicial es el símbolo de pila vacía (▼). La característica de este autómata es que, dependiendo del símbolo que haya en el tope de la pila se efectúa determinada acción. Esta acción se presenta en la columna adicional que se ha llamado *acción*.

En la tabla de apilamiento las transiciones que se han dejado en blanco significa que son situaciones de error.

Con base en la tabla de apilamiento y en la aplicación del principio de reducción se construye el autómata de pila con el cual se reconoce el lenguaje generado por la gramática.

El autómata finito se presenta en la tabla 14.3.

	A	b	c	⊣
▼			S	
$<S>_0$				Acepte
a_1	S	S	S	
$<A>_1$	S			
b_1		reduce(1)		reduce(1)
c_2		reduce(2)		reduce(2)
b_3	S		S	

$<S>_3$		reduce(3)		
$<B>_4$		S		
b_4		reduce(4)		
a_5	S	S	S	
$<A>_5$		reduce(5)		
c_6		reduce(6)		

Tabla 14.3

Las transición **S** significa: apile según la tabla de apilamiento y avance.

Dichas transiciones se han construido simplemente analizando la tabla de apilamiento tabla 14.2). Basta con fijarse en las columnas correspondientes a los terminales y en aquellos sitios en los que la transición no sea rechace implica que la transición será **S** en el autómata de pila.

Para determinar cuáles transiciones son proceso de reducción se aplica el principio de reducción que se había definido en el método shift-identify. Es decir, analizamos cada producción con su correspondiente ocurrencia gramatical que sea la manejadora de esa producción y en los sitios correspondientes al conjunto de siguientes del símbolo del lado izquierdo de esa producción se efectúa la acción de reducir.

Por consiguiente, se necesita el conjunto de siguientes de los no terminales de la gramática.

Siguientes(<S>) = {b, ┤}
Siguientes(<A>) = {b}
Siguientes(<B>) = {b}

Reduce(1) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{b_1}$ y el símbolo de entrada pertenezca a **siguientes(<S>)**, es decir, el terminal **b** y el símbolo de fin de secuencia (┤).

Reduce(2) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{c_2}$ y el símbolo de entrada pertenezca a **siguientes(<S>)**, es decir, el terminal **b** y el símbolo de fin de secuencia (┤).

Reduce(3) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{<S>_3}$ y el símbolo de entrada pertenezca a **siguientes(<A>)**, es decir, el terminal **b**.

Reduce(4) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{b_4}$ y el símbolo de entrada pertenezca a **siguientes(<A>)**, es decir, el terminal **b**.

Reduce(5) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{<A>_5}$ y el símbolo de entrada pertenezca a **siguientes(<B>)**, es decir, el terminal **b**.

Reduce(6) se efectúa cuando en el tope de la pila se encuentre la ocurrencia gramatical $\mathbf{c_6}$ y el símbolo de entrada pertenezca a **siguientes(<B>)**, es decir, el terminal **b**.

Las operaciones de reducción son:

reduce(1): desapile(3), apile(<S>) según la tabla de apilamiento, retenga.
reduce(2): desapile(1), apile(<S>) según la tabla de apilamiento, retenga.
reduce(3): desapile(2), apile(<A>) según la tabla de apilamiento, retenga.
reduce(4): desapile(2), apile(<A>) según la tabla de apilamiento, retenga.
reduce(5): desapile(2), apile(<B>) según la tabla de apilamiento, retenga.
reduce(6): desapile(1), apile(<B>) según la tabla de apilamiento, retenga.

14.6 Síntesis

Habiendo presentado un ejemplo veamos cuáles son los pasos que se siguen para construir un reconocedor ascendente, técnica shift-reduce.

1. Definir las ocurrencias gramaticales para la gramática.

2. Construir el conjunto de primeros para las ocurrencias gramaticales correspondientes a cada no terminal de la gramática. Dada una producción de la forma <P> → Aα siendo α una hilera cualquiera de terminales y no terminales, primeros(<P>) será A_i, siendo A_i la ocurrencia gramatical correspondiente al símbolo gramatical A, independientemente de si A es un terminal o un no terminal; si A es un no terminal hay que agregar los primeros(A).

3. Construir la relación *estáDebajoDe* para las ocurrencias gramaticales definidas: x_i *estáDebajoDe* A_i, sí y sólo sí, existe una hilera x_iA_i en el lado derecho de alguna producción; adicionalmente, ▼ *estádebajoDe* $<S>_0$ y de los primeros(<S>), siendo <S> el símbolo inicial de la gramática.

4. Construir la tabla de apilamiento con base en la relación *estáDebajoDe*.

5. Construir el conjunto de siguientes para cada no terminal de la gramática.

6. Construir el autómata de pila.

EJERCICIOS PROPUESTOS

Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para las siguientes gramáticas:

1.
    1. <S> → a
    2. <S> → (<S><R>
    3. <R> → , <S><R>
    4. <R> →)

2.
    1. <S> → a<S>b
    2. <S> → a<S>c
    3. <S> → ab

3.
    1. <S> → a<S><S>b
    2. <S> → d<S><S><S>
    3. <S> → c

SOLUCIONES

1. Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para la siguiente gramática:

1. <S> → a
2. <S> → (<S><R>
3. <R> → , <S><R>
4. <R> →)

Solución:

Comencemos construyendo las ocurrencias gramaticales:

Gramática	Ocurrencia gramatical
	$<S>_0$
1. <S> → a	a
2. <S> → (<S><R>	$(<S>_2<R>_2$
3. <R> → , <S><R>	$,<S>_3<R>_3$
4. <R> →)	)

Observe que no hemos definido ocurrencia gramatical especial para los símbolos a, (y), ya que sólo aparecen una vez en toda la gramática.

Continuamos construyendo el conjunto el conjunto de primeros de cada no terminal teniendo en cuenta las ocurrencias gramaticales.

Primeros(<S>) = {a, (}
Primeros(<R>) = {",",)}

Ahora construyamos la relación estáDebajoDe para las ocurrencias gramaticales, la cual se presenta en la tabla 14.4:

	$<S>_0$	a	(	$<S>_2$	$<R>_2$	,	$<S>_3$	$<R>_3$	)
▼	1	1	1						
$<S>_0$									
a									
(		1	1	1					
$<S>_2$					1	1			1
$<R>_2$									
,		1	1				1		
$<S>_3$						1		1	1
$<R>_3$									
)									

Tabla 14.4

Con base en esta relación, se construye la tabla de apilamiento, la cual se presenta en la tabla 14.5:

	<S>	<R>	a	(	,	)
▼	$<S>_0$		a	(		
$<S>_0$						
a						
(	$<S>_2$		a	(		
$<S>_2$		$<R>_2$			,	)
$<R>_2$						
,	$<S>_3$		a	(		
$<S>_3$		$<R>_3$			,	)
$<R>_3$						
)						

Tabla 14.5

Procedemos ahora a construir el autómata de pila. Para ello construyamos el conjunto de siguientes de cada no terminal.

Siguientes(<S>) = {",",), ┤}
Siguientes(<R>) = {",",), ┤}

Símbolos de entrada = {a, (, ",",), ┤}
Símbolos en la pila = {▼, $<S>_0$, a, (, $<S>_2$, $<R>_2$, ",", $<S>_3$, $<R>_3$,)}
Configuración inicial de la pila = ▼
Transiciones:

	a	(	,	)	┤
▼	S	S			
$<S>_0$					
a			reduce(1)	reduce(1)	reduce(1)
(	S	S			
$<S>_2$			S	S	
$<R>_2$			reduce(2)	reduce(2)	reduce(2)
,	S	S			
$<S>_3$			S	S	
$<R>_3$			reduce(3)	reduce(3)	reduce(3)
)			reduce(4)	reduce(4)	reduce(4)

Tabla 14.6

S: apile según tabla de apilamiento, avance.
reduce(1): desapile(1), apile(<S>) según tabla de apilamiento, retenga.
reduce(2): desapile(3), apile(<S>) según tabla de apilamiento, retenga.
reduce(3): desapile(3), apile(<R>) según tabla de apilamiento, retenga.
reduce(4): desapile(1), apile(<R>) según tabla de apilamiento, retenga.

3. Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para la siguiente gramática.

1. <S> → a<S><S>b
2. <S> → d<S><S><S>
3. <S> → c

Solución:

Comencemos construyendo las ocurrencias gramaticales:

Gramática	Ocurrencia gramatical
	$<S>_0$
1. $<S> \rightarrow a<S><S>b$	$a<S>_1<S>_2b$
2. $<S> \rightarrow d<S><S><S>$	$d<S>_3<S>_4<S>_5$
3. $<S> \rightarrow c$	c

Construyamos el conjunto de primeros teniendo en cuenta las ocurrencias gramaticales.

Primeros(<S>) = {a, c, d}

Continuemos con la relación estáDebajoDe para dichas ocurrencias, la cual presentamos en la tabla 14.7:

	$<S>_0$	a	$<S>_1$	$<S>_2$	b	$<S>_3$	$<S>_4$	$<S>_5$	c	d
▼	1	1							1	1
$<S>_0$										
a		1	1						1	1
$<S>_1$		1		1					1	1
$<S>_2$					1					
b										
$<S>_3$		1					1		1	1
$<S>_4$		1						1	1	1
$<S>_5$										
c										
d		1				1			1	1

Tabla 14.7

Ahora, con base en esta relación construimos la tabla de apilamiento, la cual se presenta en la tabla 14.8:

	<S>	a	b	c	d
▼	$<S>_0$	a		c	d
$<S>_0$					
a	$<S>_1$	a		c	
$<S>_1$	$<S>_2$	a		c	d
$<S>_2$			b		
b					
$<S>_3$	$<S>_4$	a		c	d
$<S>_4$	$<S>_5$	a		c	d
$<S>_5$					
c					
d	$<S>_3$	a		c	d

Tabla 14.8

Y por último, procedemos a construir el autómata de pila, el cual se presenta en la tabla 14.9. Para ello requerimos el conjunto de siguientes.

Siguientes(<S>) = {a, b, c, d, ┤}

Símbolos de entrada = {a, b, c, d, ┤}
Símbolos en la pila = {▼, <S>$_0$, a, <S>$_1$, <S>$_2$, b, <S>$_3$, <S>$_4$, <S>$_5$, c, d}
Configuración inicial de la pila: ▼
Transiciones:

	a	b	c	d	┤
▼	S		S	S	
<S>$_0$					
a	S		S	S	
<S>$_1$		S			
<S>$_2$					
b	reduce(1)	reduce(1)	reduce(1)	reduce(1)	reduce(1)
<S>$_3$	S		S		
<S>$_4$	S		S		
<S>$_5$	reduce(2)	reduce(2)	reduce(2)	reduce(2)	reduce(2)
c	reduce(3)	reduce(3)	reduce(3)	reduce(3)	reduce(3)
d	S		S	S	

Tabla 14.9

S: apile según tabla de apilamiento, avance.
reduce(1): desapile(4), apile(<S>) según tabla de apilamiento, retenga.
reduce(2): desapile(4), apile(<S>) según tabla de apilamiento, retenga.
reduce(3): desapile(1), apile(<S>) según tabla de apilamiento, retenga.

MODULO 15
RECONOCIMIENTO ASCENDENTE

15.1 INTRODUCCIÓN
Presentamos en el módulo anterior la construcción de reconocedores ascendente de pila, técnica shift-reduce, para gramáticas que producen tablas de apilamiento determinísticas. Veremos en este módulo gramáticas que generan tablas de apilamiento no determinísticas y su conversión a determinísticas.

15.2 OBJETIVOS
1. Reconocer cuándo el autómata finito reconocedor de manejadores es no determinístico.
2. Convertir el autómata finito reconocedor de manejadores, no determinístico, a determinístico teniendo cuidado con las diferentes situaciones que se puedan presentar.

15.3 PREGUNTAS BÁSICAS
1. Cuándo un autómata finito reconocedor de manejadores es no determinístico.
2. Qué es un conflicto reduce-reduce.
3. Cómo se resuelve un conflicto reduce-reduce.

15.4 Un ejemplo.
En el módulo anterior se presentó un primer ejemplo en el cual se construyó el autómata de pila con el cual se reconoce el lenguaje que genera una gramática. En dicho ejemplo se vio que la tabla de apilamiento se mira como un autómata finito reconocedor de manejadores. Como la tabla de apilamiento se está considerando un autómata finito, y se construye con base en la relación *estáDebajoDe* para las ocurrencias gramaticales de la gramática, hay situaciones en las que dicho autómata es no determinístico, por tanto se debe convertir a determinístico.

Veamos un ejemplo en el cual se presenta esta situación.

1. <E> → <E>+<T>
2. <E> → <T>
3. <T> → <T>*<P>
4. <T> → <P>
5. <P> → (<E>)
6. <P> → I

Se comienza definiendo las ocurrencias gramaticales de la gramática. Para la gramática mostrada las ocurrencias gramaticales son:

Gramática	Ocurrencias gramaticales
	$<E>_0$
1. <E> → <E>+<T>	$<E>_1+<T>_1$
2. <E> → <T>	$<T>_2$
3. <T> → <T>*<P>	$<T>_3*<P>_3$
4. <T> → <P>	$<P>_4$
5. <P> → (<E>)	$(<E>_5)$
6. <P> → I	I

Recuerde que $<E>_0$ es la ocurrencia gramatical correspondiente al estado de aceptación.

El siguiente paso es construir la relación *estáDebajoDe* para las ocurrencias gramaticales. Para construir la relación *estáDebajoDe* es necesario construir el conjunto de primeros de cada no terminal de la gramática. Estos conjuntos son:

Primeros(<E>) = {$<E>_1$, $<T>_2$, $<T>_3$, $<P>_4$, (, I}
Primeros(<T>) = {$<T>_3$, $<P>_4$, (, I}
Primeros(<P>) = {(, I}

La relación *estáDebajoDe* se presenta en la tabla 15.1.

	$<E>_0$	$<E>_1$	+	$<T>_1$	$<T>_2$	$<T>_3$	*	$<P>_3$	$<P>_4$	(	$<E>_5$	)	I
▼	1	1			1	1			1	1			1
$<E>_0$													
$<E>_1$			1										
+				1		1			1	1			1
$<T>_1$													
$<T>_2$													
$<T>_3$							1						
*								1		1			1
$<P>_3$													
$<P>_4$													
(		1			1	1			1	1	1		1
$<E>_5$												1	
)													
I													

Tabla 15.1

Con base en la relación *estáDebajoDe* se construye la tabla de apilamiento, la cual se muestra a continuación en la tabla 15.2.

	<E>	<T>	<P>	+	*	(	)	I	Acción
▼	$<E>_0$, $<E>_1$	$<T>_2$, $<T>_3$	$<P>_4$			(		I	continúe
$<E>_0$									Acepte
$<E>_1$				+					continúe
+		$<T>_1$, $<T>_3$	$<P>_4$			(		I	continúe
$<T>_1$									reduce(1)
$<T>_2$									reduce(2)
$<T>_3$					*				continúe
*			$<P>_3$			(		I	continúe
$<P>_3$									reduce(3)
$<P>_4$									reduce(4)
(	$<E>_1$, $<E>_5$	$<T>_2$, $<T>_3$	$<P>_4$			(		I	continúe
$<E>_5$							)		continúe
)									reduce(5)
I									reduce(6)

Tabla 15.2

Como se podrá observar, el autómata finito resultante es un autómata finito no determinístico. Las transiciones correspondientes a cuando el autómata se halle en el estado **(▼)** y el símbolo de entrada sea el no terminal <E> o el no terminal <T>, la transición puede ser hacia más de un estado. Igual sucede cuando el autómata se halle en el estado **+** y el símbolo de entrada sea el no terminal <T>, y cuando el autómata se halle en el estado **(** y el símbolo de entrada sea una <E> o una <T>.

Dicho autómata se debe convertir a determinístico aplicando la misma técnica que se definió en el módulo 2.

El autómata determinístico se presenta en la tabla 15.3

	<E>	<T>	<P>	+	*	(	)	I	Acción
▼	$<E>_{01}$	$<T>_{23}$	$<P>_{4}$			(		I	continúe
$<E>_{01}$				+					Acepte continúe
$<T>_{23}$					*				reduce(2) continúe
$<P>_{4}$									reduce(4)
(	$<E>_{15}$	$<T>_{23}$	$<P>_{4}$			(		I	continúe
I									reduce(6)
+		$<T>_{13}$	$<P>_{4}$			(		I	continúe
*			$<P>_{3}$			(		I	continúe
$<E>_{15}$				+			)		continúe
$<T>_{13}$					*				reduce(1) continúe
$<P>_{3}$									reduce(3)
)									reduce(5)

Tabla 15.3

Como podrá observar en la tabla de apilamiento determinística en la columna de acción aparece la unión de Acepte con continúe, reduce(2) con continúe y reduce(1) con continúe. Más adelante veremos cómo tratar esta situación en la construcción del autómata de pila, el cual se muestra en la tabla 15.4.

Es importante en este punto hacer notar que si en el proceso de conversión de determinístico a no determinístico aparece que hay que efectuar la unión de dos operaciones de reducción (reduce(i) con reduce(j), por ejemplo) significa que la gramática tiene un conflicto reduce-reduce, el cual no se puede aceptar puesto que no se garantiza que el proceso de reconocimiento funcione correctamente. Cuando se presenta un conflicto reduce-reduce habrá que replantear la gramática, puesto que no existe alguna transformación gramatical formal que permita solucionar este conflicto.

Recordemos que para determinar los procesos de reducción se requiere tener construido el conjunto de siguientes para cada no terminal. Estos son:

Siguientes(<E>) = {+,), ┤}
Siguientes(<T>) = {+, *,), ┤}
Siguientes(<P>) = {+, *,), ┤}

	+	*	(	)	I	┤
▼			S		S	
$<E>_{01}$	S					Acepte
$<T>_{23}$	reduce(2)	S		reduce(2)		reduce(2)
$<P>_{4}$	reduce(4)	reduce(4)		reduce(4)		reduce(4)
(			S		S	
I	reduce(6)	reduce(6)		reduce(6)		reduce(6)
+			S		S	
*			S		S	
$<E>_{15}$	S			S		
$<T>_{13}$	reduce(1)	S		reduce(1)		reduce(1)
$<P>_{3}$	reduce(3)	reduce(3)		reduce(3)		reduce(3)
)	reduce(5)			reduce(5)		reduce(5)

Tabla 15.4

La unión de Acepte con continúe es muy sencilla: cuando en el tope de la pila se halle la ocurrencia gramatical **$<E>_{01}$** y el símbolo de entrada sea el **+** la transición será **S** (apile y avance) y cunado el símbolo de entrada sea el símbolo de fin de secuencia (┤) la transición será Acepte.

La unión de reduce(2) con continúe: Cuando en el tope de la pila se halle la ocurrencia gramatical $<T>_{23}$, ésta significa que en el tope de la pila podría estar $<T>_2$ ó $<T>_3$, si fuera $<T>_2$ hay que hacer proceso de reducción cuando en el símbolo de entrada pertenezca a **Siguientes(<E>)**, en virtud de la producción 2, y si fiera $<T>_3$ la transición es S (apile y avance) cuando el símbolo de entrada sea el asterisco (*).

La unión de reduce(1) con continúe se resuelve de una manera similar. La ocurrencia gramatical <T>13 significa que en el tope de la pila está $<T>_1$ o $<T>_3$. Si fuera $<T>_1$ habrá que hacer reduce(1) cuando el símbolo de entrada pertenezca a siguientes(<E>),en virtud de la producción 1, y si fuera $<T>_3$ la transición es S cuando el símbolo de entrada sea el asterisco.

Las operaciones de reducción son:

reduce(1): desapile(3), apile(<E>) según la tabla de apilamiento, retenga.
reduce(2): desapile(1), apile(<E>) según la tabla de apilamiento, retenga.
reduce(3): desapile(3), apile(<T>) según la tabla de apilamiento, retenga.
reduce(4): desapile(1), apile(<T>) según la tabla de apilamiento, retenga.
reduce(5): desapile(3), apile(<P>) según la tabla de apilamiento, retenga.
reduce(6): desapile(1), apile(<P>) según la tabla de apilamiento, retenga.

Ilustremos cómo funciona el autómata de la tabla 15.4 con la expresión **a+b*c┤.**

La configuración inicial de la pila se muestra en el caso (a) de la figura 15.1 y el símbolo de entrada es el identificador **a**. Según el autómata de pila de la tabla 15.4 la transición es **S** (apile según la tabla de apilamiento y avance), quedando la pila como en el caso (b) de la figura 15.1.

Estando **I** en el tope de la pila y siendo **+** el símbolo de entrada la transición, según la tabla 15.4, es reduce(6) (desapile(1), apile(<P>) según la tabla de apilamiento y retenga) quedando la pila como en el caso (c) de la figura 15.1.

Al efectuar la reducción por la producción 6 el símbolo de entrada sigue siendo el mismo: en el tope de la pila está la ocurrencia gramatical **<P>$_4$** y el símbolo de entrada es el **+**, que según el autómata finito de la tabla 15.4 la transición es **reduce(4)** (desapile(1), apile(<T>) según la tabla de apilamiento y retenga), por tanto, la pila queda como en el caso (d) de la figura 15.1.

Al reducir por la producción 4 el símbolo de entrada continúa siendo el mismo: en el tope de la pila está la ocurrencia gramatical **<T>$_{23}$** y el símbolo de entrada es el **+**, que según el autómata de pila de la tabla 15.3, la transición es **reduce(2)** (desapile (1), apile(<E>) según la tabla de apilamiento y retenga), por consiguiente, la pila queda como en el caso (e) de la figura 15.1.

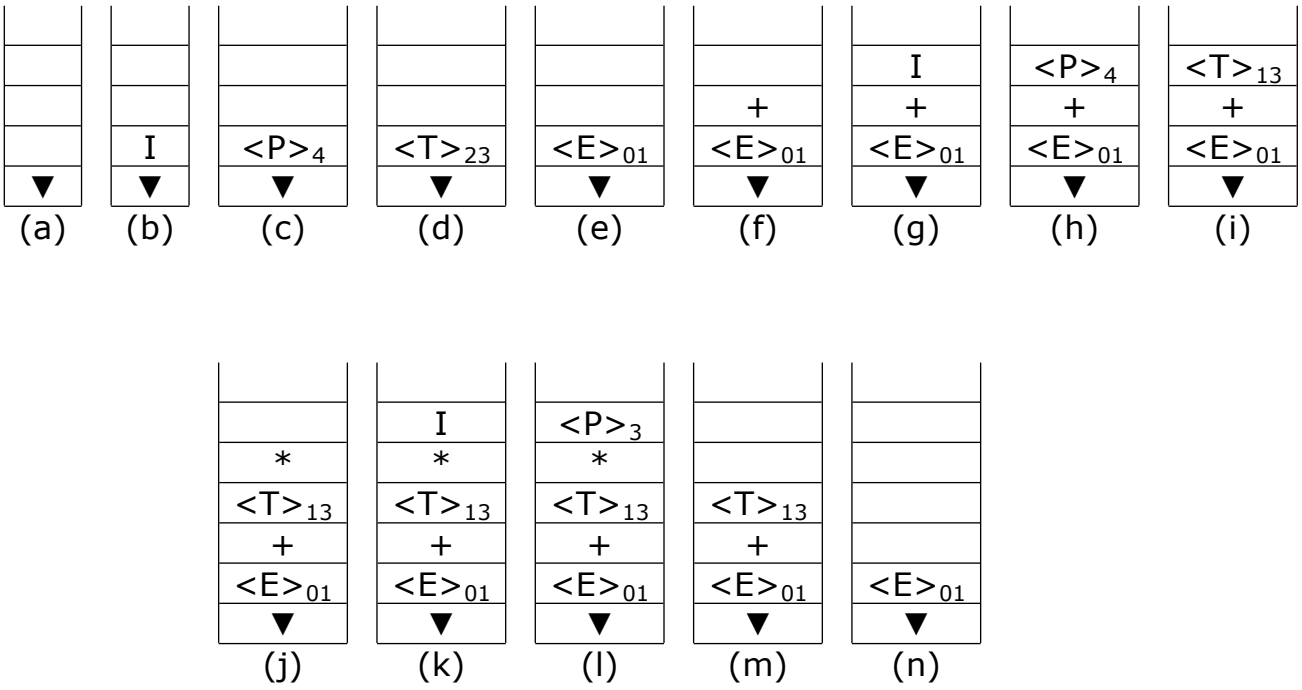


Figura 15.1

Al reducir por la producción 1 en el tope de la pila queda la ocurrencia gramatical **<E>$_{01}$** y el símbolo de entrada continúa el símbolo **+**, lo cual según el autómata de pila de la tabla 15.4, la transición es **S** (Apile según la tabla de apilamiento y avance), quedando la pila como en el caso (f) de la figura 15.1.

Estamos en la situación en la que en el tope de la pila está el símbolo **+** y el símbolo de entrada es el identificador **b**, que según el autómata finito de la tabla 15.4, la transición es S (Apile según la tabla de apilamiento y avance), quedando la pila como en el caso (g) de la figura 15.1.

Nuevamente en el tope de la pila está la ocurrencia gramatical **I** y el símbolo de entrada es *****, que según el autómata de pila de la figura 15.4, la transición es **reduce(6)** (desapile(1), apile(<P>) según la tabla de apilamiento y retenga), quedando la pila como en el caso (h) de la figura 15.1.

Habiendo quedado la ocurrencia gramatical **<P>$_4$** en el tope de la pila y siendo ***** el símbolo de entrada, la transición según el autómata de pila es S (Apile según la tabla de apilamiento y avance), quedando la pila como en el caso (j) de la figura 15.1.

Teniendo un asterisco (*****) en el tope de la pila y siendo el símbolo de entrada el identificador **c**, la transición según el autómata de pila es **S** (Apile según la tabla de apilamiento y avance), quedando la pila como en el caso (k) de la figura 15.1.

En el tope de la pila está la ocurrencia gramatical **I** y el símbolo de entrada es el símbolo de fin de secuencia (┤), que según el autómata de pila de la tabla 15.4 es **reduce(6)** (desapile(1), apile(<P>) según la tabla de apilamiento y retenga), quedando la pila como en el caso (l) de la figura 15.1.

En el tope de la pila quedó la ocurrencia gramatical **<P>$_3$** y el símbolo de entrada es el símbolo de fin de secuencia (┤), por tanto, la transición según el autómata de pila es **reduce(3)** (Desapile(3), apile(<T>) según la tabla de apilamiento y retenga), quedando el autómata de pila como en el caso (m) de la figura 15.1.

En el tope de la pila quedó la ocurrencia gramatical **<T>$_{13}$** y el símbolo de entrada continúa siendo el símbolo de fin de secuencia (┤), por consiguiente, la transición según el autómata de pila es **reduce(1)** (Desapile(3), apile(<E>) según la tabla de apilamiento y retenga), quedando la pila como en el caso (n) de la figura 15.1.

En el tope de la pila quedó la ocurrencia gramatical **<E>$_{01}$** y el símbolo de entrada es el símbolo de fin de secuencia (┤), por tanto, se acepta la hilera de entrada.

EJERCICIOS PROPUESTOS

Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para la siguiente gramática:

1. <S> → <A><A>d
2. <S> → c<A>d
3. <S> → b
4. <A> → <A><S>c
5. <A> → cd
6. <A> → a
7. <A> → <S>b

SOLUCIÓN

Comencemos definiendo las ocurrencias gramaticales:

Gramática	Ocurrencias gramaticales
	$<S>_0$
1. <S> → <A><A>d	$<A>_1<A>_2d_1$
2. <S> → c<A>d	$c_2<A>_3d_2$
3. <S> → b	b_3
4. <A> → <A><S>c	$<A>_4<S>_4c_4$
5. <A> → cd	c_5d_5
6. <A> → a	a
7. <A> → <S>b	$<S>_7b_7$

Para construir la relación estáDebajoDe para las ocurrencias gramaticales, la cual se presenta en la tabla 15.5, se construye el conjunto de primeros de cada no terminal:

Primeros(<S>) = {$<A>_1$, c_2, b_3}
Primeros(<A>) = {$<A>_4$, c_5, a, $<S>_7$}

	$<S>_0$	$<A>_1$	$<A>_2$	d_1	c_2	$<A>_3$	d_2	b_3	$<A>_4$	$<S>_4$	c_4	c_5	d_5	a	$<S>_7$	b_7
▼	1	1			1			1	1			1		1	1	
$<S>_0$																
$<A>_1$		1	1		1			1	1			1		1	1	
$<A>_2$				1												
d_1																
c_2		1			1	1		1	1			1		1	1	
$<A>_3$							1									
d_2																
b_3																
$<A>_4$		1			1			1	1	1		1		1	1	
$<S>_4$											1					
c_4																
c_5													1			
d_5																
a																
$<S>_7$																1
b_7																

Tabla 15.5

Ahora, con base en la relación estáDebajoDe se construye la tabla de apilamiento, la cual se presenta en la tabla 15.6.

	<S>	<A>	a	b	c	d	Acción
▼	$<S>_0$, $<S>_7$	$<A>_1$, $<A>_4$	a	b_3	c_2, c_5		continúe
$<S>_0$							Acepte
$<A>_1$	$<S>_7$	$<A>_1$, $<A>_2$, $<A>_4$	a	b_3	c_2, c_5		continúe
$<A>_2$						d_1	continúe
d_1							reduce(1)
c_2	$<S>_7$	$<A>_1$, $<A>_3$, $<A>_4$	a	b_3	c_2, c_5		continúe
$<A>_3$						d_2	continúe
d_2							reduce(2)
b_3							reduce(3)
$<A>_4$	$<S>_4$, $<S>_7$	$<A>_1$, $<A>_4$	a	b_3	c_2, c_5		continúe

$<S>_4$					c_4		continúe
c_4							reduce(4)
c_5						d_5	continúe
d_5							reduce(5)
a							reduce(6)
$<S>_7$				b_7			continúe
b_7							reduce(7)

Tabla 15.6

Como la tabla de apilamiento resulta no determinística, procedemos a convertirla a determinística. La tabla de apilamiento determinística se presenta en la tabla 15.7.

	<S>	<A>	a	b	c	d	Acción
▼	$<S>_{07}$	$<A>_{14}$	a	b_3	c_{25}		continúe
$<S>_{07}$				b_7			Acepte, continúe
$<A>_{14}$	$<S>_{47}$	$<A>_{124}$	a	b_3	c_{25}		continúe
a							reduce(6)
b_3							reduce(3)
c_{25}	$<S>_7$	$<A>_{134}$	a	b_3	c_{25}	d_5	continúe
b_7							reduce(7)
$<S>_{47}$				b_7	c_4		continúe
$<A>_{124}$	$<S>_{47}$	$<A>_{124}$	a	b_3	c_{25}	d_1	continúe
$<S>_7$				b_7			continue
$<A>_{134}$	$<S>_{47}$	$<A>_{124}$	a	b_3	c_{25}	d_2	continúe
d_5							reduce(5)
c_4							reduce(4)
d_1							reduce(1)
d_2							reduce(2)

Tabla 15.7

Teniendo construida la tabla de apilamiento determinística procedemos a construir el autómata de pila. Para ello requerimos construir primero el conjunto de siguientes de cada no terminal.

Siguientes(<S>) = {b, c, ┤}
Siguientes(<A>) = {a, b, c, d}

Símbolos de entrada = {}
Símbolos en la pila = {}
Configuración inicial de la pila: ▼
Transiciones:

	a	b	c	d	┤
▼	S	S	S		
$<S>_{07}$		S			Acepte
$<A>_{14}$	S	S	S		
a	reduce(6)	reduce(6)	reduce(6)	reduce(6)	
b_3		reduce(3)	reduce(3)		reduce(3)
c_{25}	S	S	S	S	
b_7	reduce(7)	reduce(7)	reduce(7)	reduce(7)	

$<S>_{47}$		S	S		
$<A>_{124}$	S	S	S	S	
$<S>_{7}$		S			
$<A>_{134}$	S	S	S	S	
d_5	reduce(5)	reduce(5)	reduce(5)	reduce(5)	
c_4	reduce(4)	reduce(4)	reduce(4)	reduce(4)	
d_1		reduce(1)	reduce(1)		reduce(1)
d_2		reduce(2)	reduce(2)		reduce(2)

S: Apile según tabla de apilamiento, avance.
Reduce(1): Desapile(3), apile(<S>) según tabla de apilamiento, retenga.
Reduce(2): Desapile(3), apile(<S>) según tabla de apilamiento, retenga.
Reduce(3): Desapile(1), apile(<S>) según tabla de apilamiento, retenga.
Reduce(4): Desapile(3), apile(<A>) según tabla de apilamiento, retenga.
Reduce(5): Desapile(2), apile(<A>) según tabla de apilamiento, retenga.
Reduce(6): Desapile(1), apile(<A>) según tabla de apilamiento, retenga.
Reduce(7): Desapile(2), apile(<A>) según tabla de apilamiento, retenga.

MODULO 16
TECNICA SHIFT-REDUCE EN GRAMATICAS CON PRODUCCIONES NULAS

16.1 INTRODUCCIÓN
Hemos trabajado el reconocimiento ascendente, técnica shift-reduce, para gramáticas que no tienen producciones cuyo lado derecho es la secuencia nula. Trataremos en este módulo gramáticas que tengan producciones cuyo lado derecho es la secuencia nula.

16.2 OBJETIVOS
3. Construir reconocedor de pila, técnica shift-reduce, para una gramática que tenga producciones cuyo lado derecho es la secuencia nula.

16.3 PREGUNTAS BÁSICAS
11. Cuándo se reduce por una producción cuyo lado derecho es la secuencia nula.
12. Cómo inciden los no terminales anulables en la relación *estáDebajoDe*.

16.4 Un ejemplo.
Consideremos la siguiente gramática, la cual tiene dos producciones cuyo lado derecho es la secuencia nula: producciones 2 y 6.

1. <S> → d<A>a
2. <S> → λ
3. <A> → a<S><B>
4. <A> → d<S>c
5. <B> → b
6. <B> → λ

La situación a solucionar aquí es cuándo reducir por la producción 2 y por la producción 6. Recuerde que se reduce por una producción cuando en el tope de la pila se halla la ocurrencia gramatical correspondiente al manejador de esa producción. En este caso, ninguna de las dos producciones tiene manejador.

Los pasos a seguir son los mismos. Comenzamos construyendo las ocurrencias gramaticales para la gramática dada.

Gramática	Ocurrencias gramaticales
	$<S>_0$
1. <S> → d<A>a	$d_1<A>_1a_1$
2. <S> → λ	
3. <A> → a<S><B>	$a_3<S>_3<B>_3$
4. <A> → d<S>c	$d_4<S>_4c_4$
5. <B> → b	b_5
6. <B> → λ	

Teniendo definidas las ocurrencias gramaticales continuamos construyendo la relación *estáDebajoDe* para dichas ocurrencias.

Recuerde que para construir la relación *estáDebajoDe* para las ocurrencias gramaticales se requiere construir el conjunto de primeros para cada uno de los no terminales de la gramática.

Dichos conjuntos son:

Primeros(<S>) = {d_1}
Primeros(<A>) = {a_3, d_4}
Primeros(<B>) = {b_5}

Teniendo estos conjuntos construimos la relación *estáDebajoDe* para las ocurrencias gramaticales, la cual se presenta en la tabla 16.1.

	<S>$_0$	d_1	<A>$_1$	a_1	a_3	<S>$_3$	<B>$_3$	d_4	<S>$_4$	c_4	b_5
▼	1	1									
<S>$_0$											
d_1			1		1			1			
<A>$_1$				1							
a_1											
a_3		1				1					
<S>$_3$							1				1
<B>$_3$											
d_4		1							1		
<S>$_4$										1	
c_4											
b_5											

Tabla 16.1

Es importante hacer notar que para la construcción de la relación *estáDebajoDe* no se tienen en cuenta los no terminales anulables.

Ahora, con base en la relación *estáDebajoDe* para las ocurrencias gramaticales construimos la tabla de apilamiento, la cual se presenta en la tabla 16.2.

	<S>	<A>	<B>	a	b	c	d	Acción
▼	<S>$_0$						d_1	continúe
<S>$_0$								Acepte
d_1		<A>$_1$		a_3			d_4	continúe
<A>$_1$				a_1				continúe
a_1								reduce(1)
a_3	<S>$_3$						d_1	continúe
<S>$_3$			<B>$_3$		b_5			continúe
<B>$_3$								reduce(3)
d_4	<S>$_4$						d_1	continúe
<S>$_4$						c_4		continúe
c_4								reduce(4)
b_5								reduce(5)

Tabla 16.2

Nuestra tabla de apilamiento, que la vemos como un autómata finito reconocedor de manejadores, es determinística, por lo tanto podremos construir de una vez el autómata de pila. Fíjese que en la columna de acción se reconocen los manejadores

correspondientes a reducir por las producciones 1, 3, 4 y 5. Pero no, los correspondientes a reducir por la producción 2 o por la producción 6.

Recuerde que las posiciones en blanco, en la tabla de apilamiento, son situaciones de rechazo.

En la construcción del autómata de pila veremos cómo determinar cuándo reducir por las producciones 2 y 6.

El autómata de pila se presenta en la tabla 16.3.

Las transiciones S (apile y avance) son las correspondientes a las entradas en la tabla de apilamiento que no son rechazo, en las columnas correspondientes a los no terminales.

	a	b	c	d	⊣
▼	reduce(2)	reduce(2)	reduce(2)	S	reduce(2)
$<S>_0$					
d_1	S			S	
$<A>_1$	S				
a_1	reduce(1)	reduce(1)	reduce(1)		reduce(1)
a_3	reduce(2)	reduce(2)	reduce(2)	S	reduce(2)
$<S>_3$	reduce(6)	S			
$<B>_3$	reduce(3)				
d_4	reduce(2)	reduce(2)	reduce(2)	S	reduce(2)
$<S>_4$			S		
c_4	reduce(4)				
b_5	reduce(5)				

Tabla 16.3

Recuerde que para determinar cuáles transiciones son reducir es necesario construir el conjunto de siguientes para cada no terminal de la gramática. Dichos conjuntos son:

Siguientes(<S>) = {a, b, c, ⊣}
Siguientes(<A>) = {a}
Siguientes(<B>) = {a}

Se reduce por la producción 1 cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{a_1}$ y el símbolo de entrada pertenezca **Siguientes(<S>)**, es decir, **a**, **b**, **c** y ⊣.

Se reduce por la producción 3 cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{<B>_3}$ y el símbolo de entrada pertenezca **Siguientes(<A>)**, es decir, el terminal **a**.

Se reduce por la producción 4 cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{c_4}$ y el símbolo de entrada pertenezca **Siguientes(<A>)**, es decir, el terminal **a**.

Se reduce por la producción 5 cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{b_5}$ y el símbolo de entrada pertenezca **Siguientes(<B>)**, es decir, el terminal **a**.

Ahora, cuándo se reduce por la producción 2.

Recuerde que el proceso de reducción por una producción **p**, consiste en desapilar tantos símbolos como haya en el lado derecho de la producción **p** y llevar a la pila el lado izquierdo de dicha producción. Cuando se reduzca por la producción 2 no se desapilará ningún símbolo, lo único que debemos hacer es apilar el no terminal <S>.

Recuerde además, que se reduce por una producción cuando en el tope de la pila se halle la ocurrencia gramatical correspondiente al manejador de esa producción y el símbolo de entrada pertenezca al conjunto de siguientes del símbolo del lado izquierdo de dicha producción. La producción 2 no tiene manejador puesto que su lado derecho es la secuencia nula (**λ**). El proceso de reducir por la producción 2 es simplemente **apilar** el símbolo del lado izquierdo, es decir, **el no terminal <S>.**

De acuerdo a la tabla de apilamiento, la tabla 16.2, el no terminal <S> se apila cuando en el tope de la pila se halle el símbolo de pila vacía (▼), la ocurrencia gramatical $\mathbf{a_3}$ ó la ocurrencia gramatical $\mathbf{d_4}$.

Por consiguiente, se reduce por la producción 2 (simplemente apilar <S>) cuando en el tope de la pila se halle ▼, $\mathbf{a_3}$ ó $\mathbf{d_4}$ y el símbolo de entrada pertenezca a **Siguientes(<S>)**, es decir, **a**, **b**, **c** ó ┤.

Para determina cuando se reduce por la producción 6, es decir, simplemente apilar el no terminal <B>, nos fijamos en la tabla de apilamiento con cuáles símbolos se apila el no terminal <B>. El no terminal <B> se apila cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{<S>_3}$. Por consiguiente cuando en el tope de la pila se halle la ocurrencia gramatical $\mathbf{<S>_3}$ y el símbolo de entrada pertenezca a **Siguientes(<B>)**, es decir, el terminal **a**, la operación es reduce(6).

Las operaciones de reducción son:

reduce(1): desapile(3), apile(<S>) según la tabla de apilamiento, retenga.
reduce(2): apile(<S>) según la tabla de apilamiento, retenga.
reduce(3): desapile(3), apile(<A>) según la tabla de apilamiento, retenga.
reduce(4): desapile(3), apile(<A>) según la tabla de apilamiento, retenga.
reduce(5): desapile(1), apile(<B>) según la tabla de apilamiento, retenga.
reduce(6): apile(<B>) según la tabla de apilamiento, retenga.

Consideremos la hilera **daa** y veamos cómo es el proceso de reconocimiento de dicha hilera. En la figura 16.1 se muestra dicho proceso.

La configuración inicial de la pila se muestra en el caso (a).

El primer símbolo que se lee es el terminal **d**: en el tope de la pila está el símbolo de pila vacía y el símbolo de entrada es el terminal **d**. Según el autómata de pila de la tabla 16.3 la transición es **S** (apile según la tabla de apilamiento y avance), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{d_1}$, quedando la pila como en el caso (b) de la figura 16.1.

El siguiente símbolo que se lee es el terminal **a**: en el tope de la pila está $\mathbf{d_1}$ y el símbolo de entrada es el terminal **a**. Según el autómata de pila de la tabla 16.3 la

transición es **S** (apile según la tabla de apilamiento y avance), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{a_3}$, quedando la pila como en el caso (c) de la figura 16.1.

El siguiente símbolo que se lee es otro terminal **a**: en el tope de la pila está $\mathbf{a_3}$ y el símbolo de entrada es el terminal **a**. Según el autómata de pila de la tabla 16.3 la transición es **reduce(2)** (apile(<S>) según la tabla de apilamiento y retenga), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{<S>_3}$, quedando la pila como en el caso (d) de la figura 16.1.

En la transición anterior se retuvo el símbolo de entrada, por tanto, el símbolo de entrada continúa siendo el terminal **a**: en el tope de la pila está $\mathbf{<S>_3}$ y el símbolo de entrada es el terminal **a**. Según el autómata de pila de la tabla 16.3 la transición es **reduce(6)** (apile(<B>) según la tabla de apilamiento y retenga), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{<B>_3}$, quedando la pila como en el caso (e) de la figura 16.1.

En la transición anterior nuevamente se retuvo el símbolo de entrada, por tanto, el símbolo de entrada continúa siendo el terminal **a**: en el tope de la pila está $\mathbf{<B>_3}$ y el símbolo de entrada es el terminal **a**. Según el autómata de pila de la tabla 16.3 la transición es **reduce(3)** (desapile(3), apile(<A>) según la tabla de apilamiento y retenga), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{<A>_1}$, quedando la pila como en el caso (f) de la figura 16.1.

Nuevamente se retuvo el símbolo de entrada, por tanto, el símbolo de entrada continúa siendo el terminal **a**: en el tope de la pila está $\mathbf{<A>_1}$ y el símbolo de entrada es el terminal **a**. Según el autómata de pila de la tabla 16.3 la transición es **S** (apile según la tabla de apilamiento y avance), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{a_1}$, quedando la pila como en el caso (g) de la figura 16.1.

En este punto, el símbolo de entrada es el símbolo de fin de secuencia (┤) y en el tope de la pila se halla $\mathbf{a_1}$. Según el autómata de pila de la tabla 16.3 la transición es **reduce(1)** (desapile(3), apile(<S>) según la tabla de apilamiento y retenga), por consiguiente, según la tabla de apilamiento de la tabla 16.2, el símbolo a apilar es $\mathbf{<S>_0}$, quedando la pila como en el caso (h) de la figura 16.1.

Como se retuvo el símbolo de entrada, en el tope de la pila se tiene $\mathbf{<S>_0}$ y el símbolo de entrada es el símbolo de fin de secuencia (┤), por tanto según el autómata de pila de la tabla 16.3 la transición es Acepte. Es decir, la hilera dada pertenece al lenguaje que genera dicha gramática.

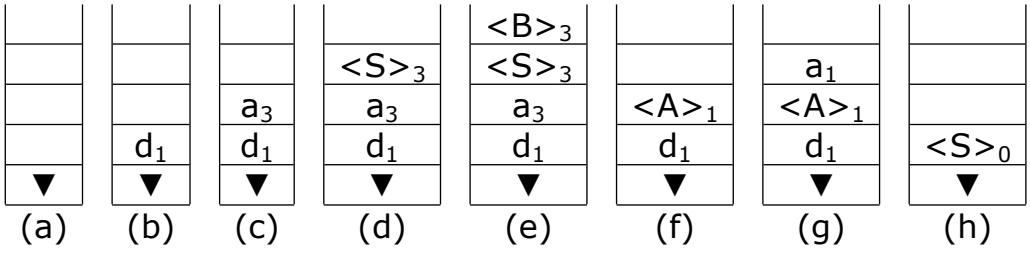


Figura 16.1

EJERCICIOS PROPUESTOS

Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para las siguientes gramáticas:

1.
    1. <S> → a<A>
    2. <S> → b<B>
    3. <A> → 1<A>0
    4. <A> → λ
    5. <B> → 1<B>00
    6. <B> → λ

2.
    1. <P> → b<D><S>e
    2. <D> → <D>d;
    3. <D> → λ
    4. <S> → <S>;<T>
    5. <S> → <T>
    6. <T> → a
    7. <T> → λ

SOLUCIONES

1. Construya reconocedor ascendente de pila, técnica SHIFT-REDUCE, para la siguiente gramática:

1. <S> → a<A>
2. <S> → b<B>
3. <A> → 1<A>0
4. <A> → λ
5. <B> → 1<B>00
6. <B> → λ

Solución:

Comenzamos definiendo las ocurrencias gramaticales:

Gramática	Ocurrencias gramaticales
	$<S>_0$
1. <S> → a<A>	$a_1<A>_1$
2. <S> → b<B>	$b_2<B>_2$
3. <A> → 1<A>0	$1_3<A>_3 0_3$
4. <A> → λ	
5. <B> → 1<B>00	$1_5<B>_5 0_5 0_6$
6. <B> → λ	

Construimos ahora el conjunto de primeros para las ocurrencias gramaticales:

Primeros(<S>) = $\{a_1, b_2\}$
Primeros(<A>) = $\{1_3\}$
Primeros(<B>) = $\{1_5\}$

Continuamos, construyendo la relación estáDebajoDe para las ocurrencias gramaticales, la cual se presenta en la tabla 16.4:

	$<S>_0$	a_1	$<A>_1$	b_2	$<B>_2$	1_3	$<A>_3$	0_3	1_5	$<B>_5$	0_5	0_6
▼	1	1		1								
$<S>_0$												
a_1			1			1						
$<A>_1$												
b_2					1				1			
$<B>_2$												
1_3						1	1					
$<A>_3$								1				
0_3												
1_5									1	1		
$<B>_5$											1	
0_5												1
0_6												

Tabla 16.4

Con base en esta relación construimos la tabla de apilamiento, la cual se muestra en la tabla 16.5:

	<S>	<A>	<B>	a	b	0	1
▼	$<S>_0$			a_1	b_2		
$<S>_0$							
a_1		$<A>_1$					1_3
$<A>_1$							
b_2			$<B>_2$				1_5
$<B>_2$							
1_3		$<A>_3$					1_3
$<A>_3$						0_3	
0_3							
1_5			$<B>_5$				1_5
$<B>_5$						0_5	
0_5						0_6	
0_6							

Tabla 16.5

Teniendo construida la tabla de apilamiento, procedemos a construir el autómata de pila, el cual se presenta en la tabla 16.6:

Siguientes(<S>) = {┤}
Siguientes(<A>) = {0, ┤}
Siguientes(<B>) = {0, ┤}

	a	b	0	1	┤
▼	S	S			
$<S>_0$					Acepte
a_1			reduce(4)	S	reduce(4)
$<A>_1$					reduce(1)
b_2			reduce(6)	S	reduce(6)
$<B>_2$					reduce(2)
1_3				S	
$<A>_3$			S		
0_3			reduce(3)		reduce(3)
1_5				S	
$<B>_5$			S		
0_5			S		
0_6			reduce(6)		reduce(6)

Tabla 16.6

S: apile según la tabla de apilamiento, avance.
reduce(1): desapile(2), apile(<S>) según tabla de apilamiento, retenga.
reduce(2): desapile(2), apile(<S>) según tabla de apilamiento, retenga.
reduce(3): desapile(3), apile(<A>) según tabla de apilamiento, retenga.
reduce(4): apile(<A>) según tabla de apilamiento, retenga.
reduce(5): desapile(4), apile(<B>) según tabla de apilamiento, retenga.
reduce(6): apile(<B>) según tabla de apilamiento, retenga.