UPN

UNIVERSIDAD PRIVADA DEL NORTE

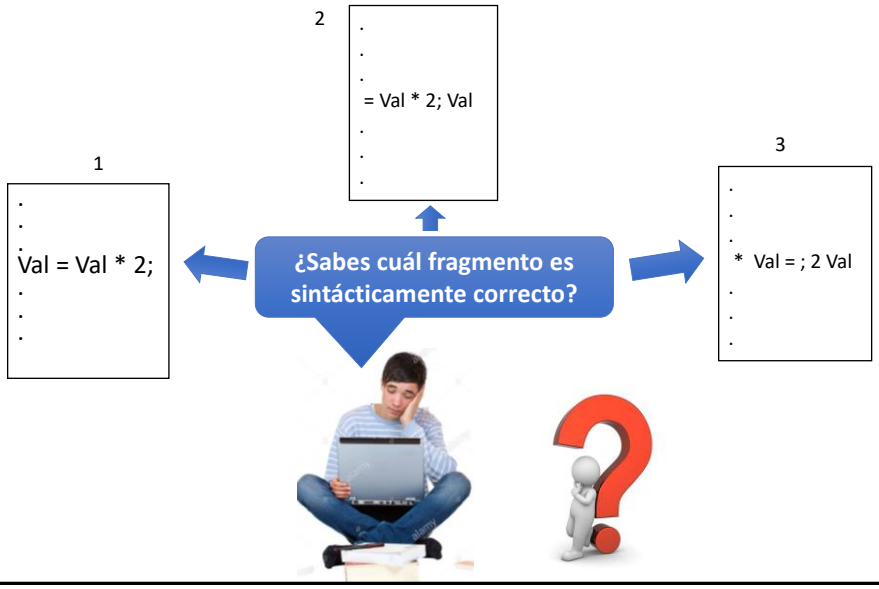
COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 2

SESIÓN N° 9

ANÁLISIS SINTÁCTICO

Carlos Enrique Castillo Diestra



LOGRO

Logro de la Unidad:

Al finalizar la unidad, el estudiante construye analizadores sintácticos usando C/C++ y BISON; a partir de la técnica de análisis sintáctico ascendente; presentando la implementación del analizador sintáctico con orden y precisión

Logro de la Sesión:

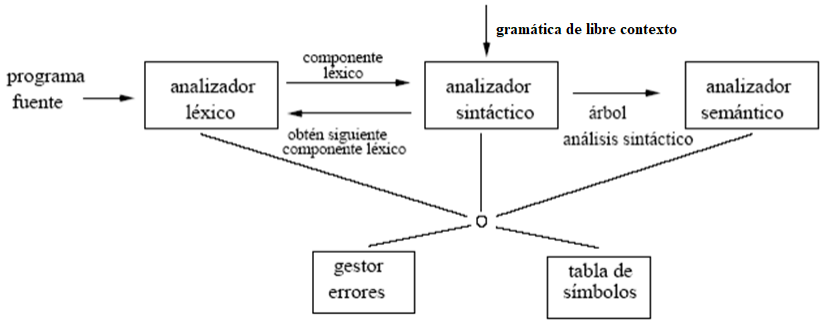
Al finalizar la Sesión, el estudiante realiza:

- El análisis sintáctico de un fragmento; usando árboles de derivación; presentando la solución en forma ordenada y correcta.
- La eliminación de la recursividad por la izquierda y/o la factorización por la izquierda de una GLC; siguiendo los respectivos algoritmos; presentando la solución en forma o correcta

CONTENIDO

DESCRIPCIÓN FUNCIONAL DEL ANÁLISIS SINTÁCTICO

El analizador sintáctico o parser verifica que los tokens que recibe del scanner, estén debidamente combinados de tal manera que cumplan con las reglas sintácticas del lenguaje fuente especificadas mediante una gramática libre de contexto.



ESPECIFICACIÓN SINTÁCTICA DE LOS LENGUAJES DE PROGRAMACIÓN

- La estructura sintáctica de los lenguajes de programación se especifica mediante Gramáticas Libres de Contexto que además son la base para los diferentes esquemas de traducción de la etapa de síntesis.

- Las producciones de la gramática libre de contexto obedecen al formato:

$$P = \{ A \rightarrow \alpha \ / \ A \in N \ \text{ y } \ \alpha \in (N \cup \Sigma)^* \}$$

Donde:

- N = Símbolos no terminales o variables sintácticas
- Σ = Símbolos terminales o tokens

DERIVACIONES Y ÁRBOL DE ANÁLISIS SINTÁCTICO

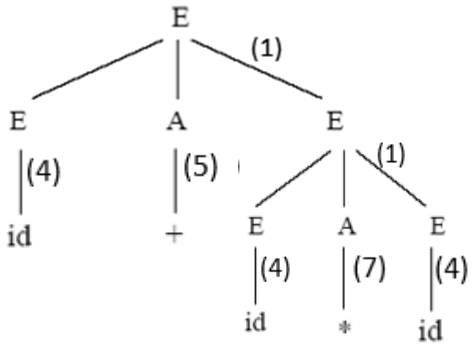
- Analizar sintácticamente una cadena de tokens es construir para ella un árbol sintáctico o de derivación, que tenga como raíz el símbolo inicial de la gramática y dicha cadena como hojas del árbol sintáctico.

- Un árbol de análisis sintáctico se caracteriza por:
  - La raíz esta etiquetada con el símbolo inicial.
  - Cada hoja esta etiquetada con un símbolo terminal. (cadena de tokens).
  - Cada nodo interior esta etiquetado con un no terminal.
  - Si A es un no terminal y $X_1, X_2, \ldots, X_n$ son sus hijos de izquierda a derecha, entonces existe la producción $A \rightarrow X_1 X_2 \ldots X_n$.

EJEMPLO

- Sea la gramática:

E → E A E	(1)
| (E)	(2)
| - E	(3)
| id	(4)
A → +	(5)
| -	(6)
| *	(7)
| /	(8)



- Analizar si la cadena de tokens:
  id+id*id

  cumple con las reglas sintácticas del lenguaje.

Como se logro generar dicha cadena se dice que si cumple con las reglas sintácticas del lenguaje (gramática).

Para ello construimos su árbol sintáctico:

EJERCICIO

Construir el árbol sintáctico para probar si la secuencia ***ent+ent*ent*** es sintáctica correcta .

Usar la siguiente gramática:

E → E+E
| E*E
| (E)
| ent
| id

GRAMÁTICAS AMBIGUAS

- Una gramática es ambigua si el lenguaje que define contiene alguna sentencia que tiene más de un único árbol sintáctico.
- La ambigüedad es difícil de resolver, no existe una metodología para eliminarla y tampoco hay otra fórmula para saber que una gramática es ambigua.
- Las gramáticas ambiguas se rediseñan para encontrar una gramática no ambigua equivalente (que genere el mismo lenguaje).

GRAMÁTICAS AMBIGUAS

E → E+E
| E*E
| (E)
| ent
|id

Ambigua

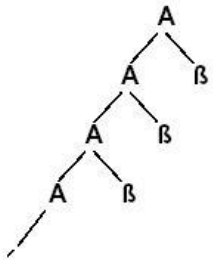
E → E+E
| T
T → T*F
| F
F → (E)
| ent
|id

No ambigua

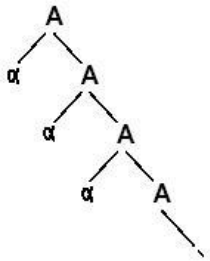
DISEÑO DE GRAMÁTICAS PARA LP

RECURSIVIDAD

Una producción es recursiva a izquierda si es de la forma A→Aβ; es recursiva a la derecha si es de la forma A→αA.

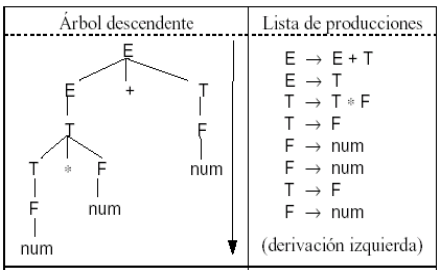


Recursividad a Izquierda



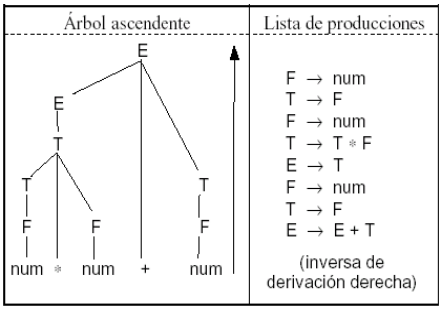
Recursividad a Derecha

ESTRATEGIAS DE ANÁLISIS SINTÁCTICO



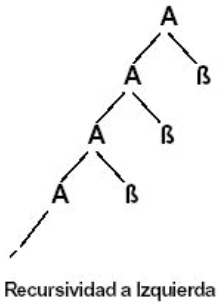
← ANALISIS SINTACTICO DESCENDENTE

ANALISIS SINTACTICO ASCENDENTE →



PROBLEMAS EN EL ANÁLISIS SINTÁCTICO DESCENDENTE

- La recursividad a izquierdas da lugar a un bucle infinito de recursión.



Recursividad a Izquierda

- Las producciones que comparten el mismo prefijo da lugar a indeterminismo.

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$

No sabríamos cual producción elegir. Probaríamos una y si se produce un error haríamos retroceso.

ELIMINACIÓN DE LA RECURSIVIDAD IZQUIERDA DIRECTA

Elimina recursividad izquierda de un paso:

$$A \rightarrow A\,\alpha \mid \beta$$

Pasos:

- Ordenar las producciones de A en la forma siguiente:
  - $A \rightarrow A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$
- Reemplazar las producciones de A por:
  - $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
  - $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

EJEMPLO

Eliminar la recursividad izquierda de la gramática:

E → E + T
| E – T
| T
T → T * F
| T / F
| F
F → (E)
| ent

Solución:

- Eliminamos recursividad izquierda directa en las producciones de E:

E → E + T | E - T | T

E → T E'
E' → + T E' | - T E' | ε

- Eliminamos recursividad izquierda directa en las producciones de T:

T → T * F | T / F | F

T → F T'
T' → * F T' | / F T' | ε

- La gramática resultante es:

E → T E'
E' → + T E'
| - T E'
| ε
T → F T'
T' → * F T'
| / F T'
| ε
F → (E)
| ent

FACTORIZACIÓN POR LA IZQUIERDA

Una gramática se factoriza por la izquierda cuando dos producciones para un no terminal tienen símbolos comunes por la izquierda: $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$.

Pasos:

- Para cada no terminal A, encontrar el prefijo α más largo común a dos o más de sus producciones.
- Ordenar la producciones de A: $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_3$
- Reemplazar las producciones de A por:

$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_3$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

- Repetir el paso anterior hasta que no existan producciones con prefijos comunes.

EJEMPLO

Factorizar la gramática:

S → if E then S else S | if E then S | a

Solución:

- Fijándonos en la regla para factorizar, podemos identificar:

  α = if E then S

  β_1 = else S

  β_2 = ε

  γ = a

- Reemplazar las producciones de S por:

  S → if E then S S' | a

  S' → else S | ε

CONCLUSIONES

- La ambigüedad es difícil de resolver, no existe una metodología para eliminarla y tampoco hay otra fórmula para saber que una gramática es ambigua.
- Mediante un árbol de derivación se puede probar si una secuencia es sintácticamente correcta.

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

1. Eliminar la recursividad por la izquierda y/o factorizar las siguientes gramáticas:

a)
S → X Y
X → a X
| a
Y → b Y
| b

b)
E → E or T
| T
T → T and F
|F
F → not F
|T % Q
|(E)
|true
|false

c)
R → d R
|d A
A → # B
B → d B
|d

d)
S → d S w C;
|d { S } w C;
|a
C → b

2. Usando la gramática, mediante un árbol de derivación, probar si la cadena ***(((a,a), b) , b)*** es sintácticamente correcta.

S → (A)
A → A , D
|D
D → a
|b
|(A)

Gracias

UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

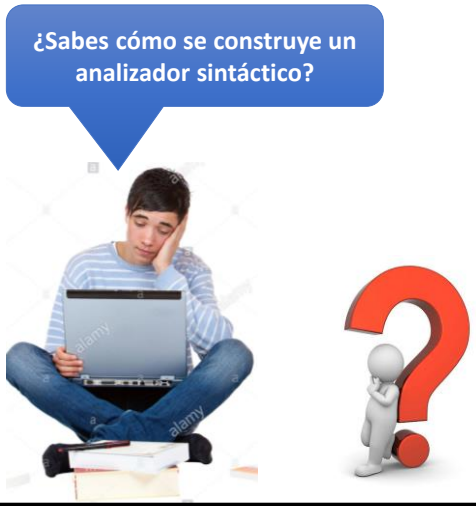
COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 3

SESIÓN N° 10

ANALIZADORES SINTÁCTICOS LL(1)

Carlos Enrique Castillo Diestra



LOGRO

Al finalizar la Sesión, el estudiante construye la tabla de análisis sintáctico LL(1) a partir de una GLC, utilizando el algoritmo para construir un analizador sintáctico LL(1); presentando la tabla con orden y precisión

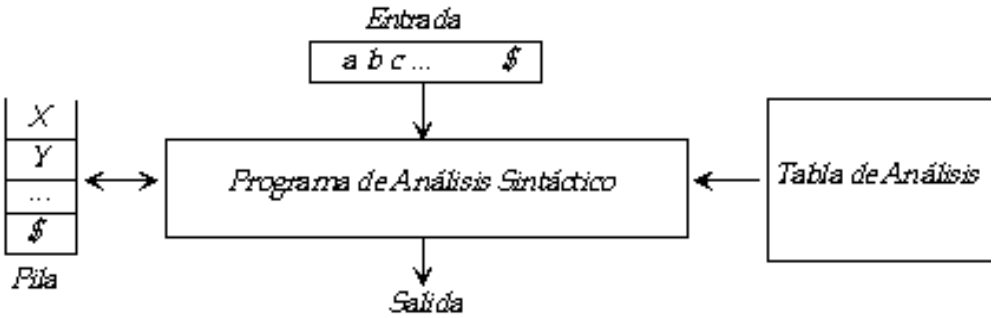
CONTENIDO

GRAMÁTICAS LL(1)

- Son gramáticas que permiten construir analizadores deterministas descendentes con solo leer en cada momento el símbolo actual de la cadena de entrada para determinar que producción aplicar.
- Significado de las iniciales del nombre:
  - L: procesamos la entrada de izquierda a derecha
  - L: obtenemos derivación más a la izquierda.
  - 1: usamos un símbolo para decidir la producción a aplicar.
- **Teoremas:**
  - **Teorema 1:** Una gramática LL(1) no puede ser recursiva a la izquierda y debe estar factorizada.
  - **Teorema 2:** Una gramática LL(1) no es ambigua.

ANALIZADOR SINTÁCTICO LL(1)

Método que busca en una tabla de análisis sintáctico para determinar que producción debe aplicarse en cada momento, en función del símbolo de pre análisis leído en ese momento y del símbolo no terminal en la cima de la pila.



ELEMENTOS

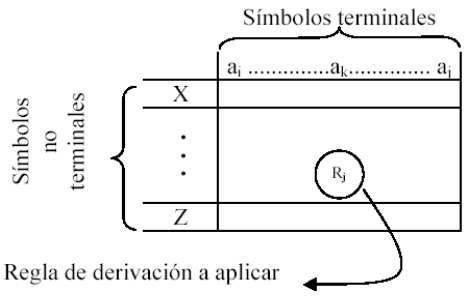
Entrada: cadena de tokens, acabada en $ (fin de cadena)

Pila de análisis:

- Almacena los símbolos gramaticales que faltan emparejar, con $ en su base.
- Inicialmente contiene el símbolo inicial encima de $

Salida: producciones usadas durante las derivaciones.

Tabla de Análisis: contiene las producciones a usar cuando se tiene que derivar el no terminal A y cuando el símbolo terminal es a.



- El programa que controla el análisis es siempre el mismo; lo único que cambia es la tabla de análisis en función de cada gramática.

ALGORITMO DE ANÁLISIS

Repetir

Sea: X = símbolo del tope de la pila

a = símbolo actual de la entrada

Si X es un terminal **Entonces**

Si X == a **Entonces**

Extraer X de la pila y obtener siguiente token

Colocar como salida: el símbolo actual ha sido emparejado

Sino Error()

Sino

Si M[X,a] = X → $Y_1Y_2...Y_n$ **Entonces**

Extraer X de la pila

Meter $Y_n...Y_2Y_1$, con Y_1 en la cima de la pila

Colocar como salida: la producción utilizada para derivar

Sino Error()

Hasta que X==a==$ (la pila está vacía y hemos procesado toda la entrada)

Colocar como salida: cadena ha sido reconocida con éxito

EJEMPLO

Sea la gramática y la tabla de análisis, analizar la cadena: id+id*id.

E → T E'

E'→ + T E' | ε

T → F T'

T' → * F T' | ε

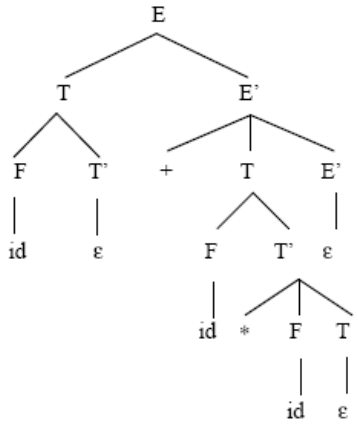
F → (E) | id

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

EJEMPLO

PILA	ENTRADA	SALIDA
$ E	id + id * id $	E → TE'
$ E' T	id + id * id $	T → FT'
$ E' T' F	id + id * id $	F → id
$ E' T' id	id + id * id $	emparejar id
$ E' T'	+ id * id $	T → ε
$ E'	+ id * id $	E' → +TE'
$ E' T +	+ id * id $	emparejar +
$ E' T	id * id $	T → FT'
$ E' T' F	id * id $	F → id
$ E' T' id	id * id $	emparejar id
$ E' T'	* id $	T' → *FT
$ E' T' F *	* id $	emparejar *
$ E' T' F	id $	F → id
$ E' T' id	id $	emparejar id
$ E' T'	$	T' → ε
$ E'	$	E' → ε
$	$	aceptar!

ÁRBOL DE ANÄLISIS SINTÁCTICO



EJEMPLO

Sea la gramática
E → T E'
E'→ + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

y la tabla de análisis, analizar la cadena: id+id*id.

	id	+	*	(	)	$
E	E→TE´			E→TE´		
E´		E´→+TE´			E´→ε	E´→ε
T	T→FT´			T→FT´		
T´		T´→ε	T´→*FT´		T´→ε	T´→ε
F	F→id			F→(E)		

Repetir
Sea: X = símbolo del tope de la pila
a = símbolo actual de la entrada
Si X es un terminal **Entonces**
Si X == a **Entonces**
Extraer X de la pila y obtener siguiente token
Colocar como salida: el símbolo actual ha sido emparejado
Sino Error()
Sino
Si M[X,a] = X → $Y_1Y_2...Y_n$ **Entonces**
Colocar como salida: la producción utilizada para derivar
Extraer X de la pila
Meter $Y_n...Y_2Y_1$, con Y_1 en la cima de la pila
Sino Error()
Hasta que X==a==$ (la pila está vacía y hemos procesado toda la entrada)

PILA	ENTRADA	SALIDA
$E	Id + id * id $	E → TE'
$E'T	Id + id * id $	T → FT'
$E'T'F	Id + id * id $	F → id
$E'T' id	Id + id * id $	Emparejar id
$E'T'	+ id * id $	T → ε
$E'	+ id * id $	E' → +TE'
$E'T +	+ id * id $	Emparejar +
$E'T	id * id $	T → FT'
$E'T' F	id * id $	F → id
$E'T' id	id * id $	Emparejar id
$E'T'	* id $	T' → *FT
$E'T' F *	* id $	emparejar *
$E'T' F	id $	F → id
$E'T' id	id $	emparejar id
$E'T	$	T' → ε
$E	$	E' → ε
$	$	Aceptar

CONSTRUCCIÓN DE LA TABLA DE ANÁLISIS SINTÁCTICO

1. Eliminar la recursividad por la izquierda y/o factorizar por la izquierda la gramática.
2. Calcular los conjuntos primero y siguiente para cada no terminal de la gramática.
3. Aplicar el algoritmo de construcción de la tabla de análisis sintáctico.

CONJUNTO PRIMERO

Definición formal:

PRIMEROS(α)={a / a$\in\Sigma \wedge \alpha \rightarrow +a\beta\} \cup \{\varepsilon\}$ si $\alpha \rightarrow \varepsilon$ donde $\alpha \in (\Sigma \cup N)^*$

Para calcular PRIMERO(*X*) para todos los símbolos gramaticales *X*, aplicamos las siguientes reglas hasta que no pueden agregarse más terminales o **ξ** a ningún conjunto PRIMERO.

1. Si X es un terminal, entonces PRIMERO$(X) = \{X\}$.
2. Si $X \rightarrow \epsilon$ es una producción, entonces se agrega ϵ a PRIMERO(X).
3. Si X es un no terminal y $X \rightarrow Y_1 Y_2 \cdots Y_k$ es una producción para cierta $k \geq 1$, entonces se coloca a en PRIMERO(X) si para cierta i, a está en PRIMERO(Y_i), y ϵ está en todas las funciones PRIMERO(Y_1), ..., PRIMERO(Y_{i-1}); es decir, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. Si ϵ está en PRIMERO(Y_j) para todas las $j = 1, 2, \ldots, k$, entonces se agrega ϵ a PRIMERO(X). Por ejemplo, todo lo que hay en PRIMERO(Y_1) se encuentra sin duda en PRIMERO(X). Si Y_1 no deriva a ϵ, entonces no agregamos nada más a PRIMERO(X), pero si $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, entonces agregamos PRIMERO(Y_2), y así sucesivamente.

CONJUNTO PRIMERO

Ahora, podemos calcular PRIMERO para cualquier cadena $X_1X_2 \cdots X_n$ de la siguiente manera. Se agregan a PRIMERO$(X_1X_2 \cdots X_n)$ todos los símbolos que no sean ϵ de PRIMERO(X_1). También se agregan los símbolos que no sean ϵ de PRIMERO(X_2), si ϵ está en PRIMERO(X_1); los símbolos que no sean ϵ de PRIMERO(X_3), si ϵ está en PRIMERO(X_1) y PRIMERO(X_2); y así sucesivamente. Por último, se agrega ϵ a PRIMERO$(X_1X_2 \cdots X_n)$ si, para todas las i, ϵ se encuentra en PRIMERO(X_i).

EJEMPLO

Calculo de los conjuntos PRIMEROS

E → T E’

E’→ + T E’ | ε

T → F T’

T’ → * F T’ | ε

F → (E) | id

NO TERMINAL	PRIMERO
E	(, id
E’	+ , ε
T	(, id
T’	* , ε
F	(, id

CONJUNTO SIGUIENTES

Definición formal:

SIGUIENTES(A)={a / a∈Σ ∧ S→*αAaβ} ∪ {$} si A→ε donde A∈N

Para calcular SIGUIENTE(*A*) para todas las no terminales *A*, se aplican las siguientes reglas hasta que no pueda agregarse nada a cualquier conjunto SIGUIENTE.

1. Póngase $ en SIGUIENTE(*S*), donde *S* es el símbolo inicial y $ es el delimitador derecho de la entrada.
2. Si hay una producción $A \rightarrow \alpha B \beta$, entonces todo lo que esté en PRIMERO(β) excepto ϵ se pone en SIGUIENTE(*B*).
3. Si hay una producción $A \rightarrow \alpha B$ o una producción $A \rightarrow \alpha B \beta$, donde PRIMERO(β) contenga ϵ (es decir, $\beta \stackrel{*}{\Rightarrow} \epsilon$), entonces todo lo que esté en SIGUIENTE(*A*) se pone en SIGUIENTE(*B*).

EJEMPLO

Calculo de los conjuntos PRIMEROS y SIGUIENTES asociados a la gramática:

E → T E'

E'→ + T E' | ε

T → F T'

T' → * F T' | ε

F → (E) | id

NO TERMINAL	PRIMERO	SIGUIENTE
E	(, id	$,)
E'	+ , ε	$,)
T	(, id	+ , $,)
T'	* , ε	+ , $,)
F	(, id	*,+,$,)

1. Siguiente (E) = $

2. Si A→ α B β entonces Siguiente (B) = Primero (β), excepto ε

3. Si A→ α B entonces Siguiente (B) = Siguiente (A)
   o A→ α B β , Primero (β) = {ε}

ALGORITMO PARA CONSTRUIR LA TABLA DE ANÁLISIS SINTÁCTICO

Las filas de la tabla se etiquetan con los símbolos no terminales de la gramática, y las columnas con los terminales y el símbolo $.

Y luego aplicar el siguiente método:

1. Para cada producción $A \rightarrow \alpha$ de la gramática, dense los pasos 2 y 3.
2. Para cada terminal a de PRIMERO(α), añádase $A \rightarrow \alpha$ a $M[A, a]$.
3. Si ϵ está en PRIMERO(α), añádase $A \rightarrow \alpha$ a $M[A, b]$ para cada terminal b de SIGUIENTE(A). Si ϵ está en PRIMERO(α) y \$ está en SIGUIENTE(A), añádase $A \rightarrow \alpha$ a $M[A, \$]$.
4. Hágase que cada entrada no definida de M sea **error**.

EJEMPLO

Construir la tabla de análisis sintáctico LL(1) para la siguiente gramática:

E → E + T
  | T
T → T * F
  | F
F → (E)
  | id

Paso 1: Eliminar la recursividad por la izquierda y/o factorizar por la izquierda

E → T E'
E' → + T E'
  | ε
T → F T'
T' → * F T'
  | ε
F → (E)
  | id

EJEMPLO

Paso 2: Obtener los conjuntos primero y siguiente de los no terminales

NO TERMINAL	PRIMERO	SIGUIENTE
E	(, id	$,)
E’	+ , ε	$,)
T	(, id	+ , $,)
T’	* , ε	+ , $,)
F	(, id	*,+,$,)

EJEMPLO

Paso 3: Construir la tabla de análisis sintáctico LL(1)

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

ALGORITMO PARA CONSTRUIR LA TABLA DE ANÁLISIS SINTÁCTICO

Las filas de la tabla se etiquetan con los símbolos no terminales de la gramática, y las columnas con los terminales y el símbolo $.

Y luego aplicar el siguiente método:

1. Para cada producción $A \rightarrow \alpha$ de la gramática, dense los pasos 2 y 3.
2. Para cada terminal a de PRIMERO(α), añádase $A \rightarrow \alpha$ a $M[A, a]$.
3. Si ϵ está en PRIMERO(α), añádase $A \rightarrow \alpha$ a $M[A, b]$ para cada terminal b de SIGUIENTE(A). Si ϵ está en PRIMERO(α) y $ está en SIGUIENTE(A), añádase $A \rightarrow \alpha$ a $M[A, \$]$.
4. Hágase que cada entrada no definida de M sea **error**.

	Id	+	*	(	)	$
E	E→ TE'			E→ TE'		
E'					E' → ε	E' → ε
T						
T'						
F						

Paso 2:

E→ TE'

Primero (TE') = {(, id }

Añadir: E→ TE' a M[E, (]

M[E, id]

E → T E'

E'→ + T E'

| ε

T → F T'

T' → * F T'

| ε

F → (E)

| id

NO TERMINAL	PRIMERO	SIGUIENTE
E	(, id	$,)
E'	+ , ε	$,)
T	(, id	+ , $,)
T'	* , ε	+ , $,)
F	(, id	*,+,$,)

Paso 3:

E'→ ε

Siguiente (E') = {$,) }

Añadir: E'→ ε a M[E', $]

M[E',)]

CONCLUSIONES

Para construir un analizador sintáctico LL(1) la GLC no puede ser recursiva a la izquierda y debe estar factorizada.

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Construir la tabla de análisis LL(1) para la siguiente gramática

P→ i E t P e P
| i E t P
| a

E→ b

Elimina recursividad izquierda de un paso:

- Ordenar las producciones de A en la forma siguiente:
  - A → A α_1 | A α_2 | … | A α_m | β_1 | β_2 | … | β_n
- Reemplazar las producciones de A por:
  - A → β_1 A' | β_2 A' | … | β_n A'
  - A' → α_1 A' | α_2 A' | … | α_m A' | ε

Factorización:

- Para cada no terminal A, encontrar el prefijo α más largo común a dos o más de sus producciones.
- Ordenar la producciones de A: A → αβ1 | αβ2 | … | αβn | γ
- Reemplazar las producciones de A por:

  A → αA' | γ

  A' → β1 | β2 | … | βn
- Repetir el paso anterior hasta que no existan producciones con prefijos comunes.

1. Si X es un terminal, entonces PRIMERO$(X) = \{X\}$.
2. Si $X \to \epsilon$ es una producción, entonces se agrega ϵ a PRIMERO(X).
3. Si X es un no terminal y $X \to Y_1 Y_2 \cdots Y_k$ es una producción para cierta $k \geq 1$, entonces se coloca a en PRIMERO(X) si para cierta i, a está en PRIMERO(Y_i), y ϵ está en todas las funciones PRIMERO(Y_1), …, PRIMERO(Y_{i-1}); es decir, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. Si ϵ está en PRIMERO(Y_j) para todas las $j = 1, 2, \ldots, k$, entonces se agrega ϵ a PRIMERO(X). Por ejemplo, todo lo que hay en PRIMERO(Y_1) se encuentra sin duda en PRIMERO(X). Si Y_1 no deriva a ϵ, entonces no agregamos nada más a PRIMERO(X), pero si $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, entonces agregamos PRIMERO(Y_2), y así sucesivamente.

1. Póngase \$ en SIGUIENTE(S), donde S es el símbolo inicial y \$ es el delimitador derecho de la entrada.
2. Si hay una producción $A \to \alpha B \beta$, entonces todo lo que esté en PRIMERO(β) excepto ϵ se pone en SIGUIENTE(B).
3. Si hay una producción $A \to \alpha B$ o una producción $A \to \alpha B \beta$, donde PRIMERO(β) contenga ϵ (es decir, $\beta \stackrel{*}{\Rightarrow} \epsilon$), entonces todo lo que esté en SIGUIENTE(A) se pone en SIGUIENTE(B).

1. **Para cada producción $A \to \alpha$ de la gramática, dense los pasos 2 y 3.**
2. **Para cada terminal a de PRIMERO(α), añádase $A \to \alpha$ a $M[A, a]$.**
3. **Si ϵ está en PRIMERO(α), añádase $A \to \alpha$ a $M[A, b]$ para cada terminal b de SIGUIENTE(A). Si ϵ está en PRIMERO(α) y \$ está en SIGUIENTE(A), añádase $A \to \alpha$ a $M[A, \$]$.**
4. **Hágase que cada entrada no definida de M sea error.**

Gracias

UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 3

SESIÓN N° 11

ANALIZADORES SINTÁCTICOS LR(1)

Carlos Enrique Castillo Diestra

¿Sabes cómo se construye un analizador sintáctico ascendente?

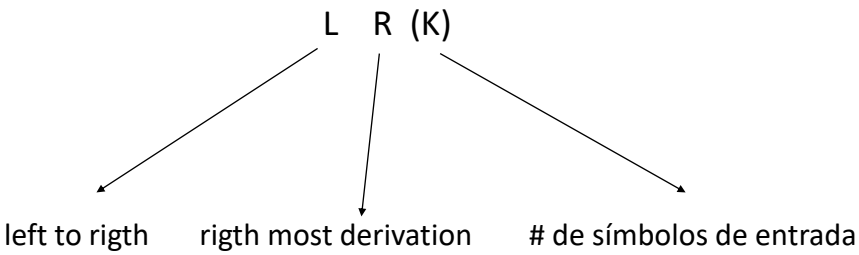
LOGRO

Al finalizar la Sesión, el estudiante construye la tabla de análisis sintáctico SLR(1) a partir de una GLC, utilizando el algoritmo para construir un analizador sintáctico SLR(1); presentando la tabla con orden y precisión

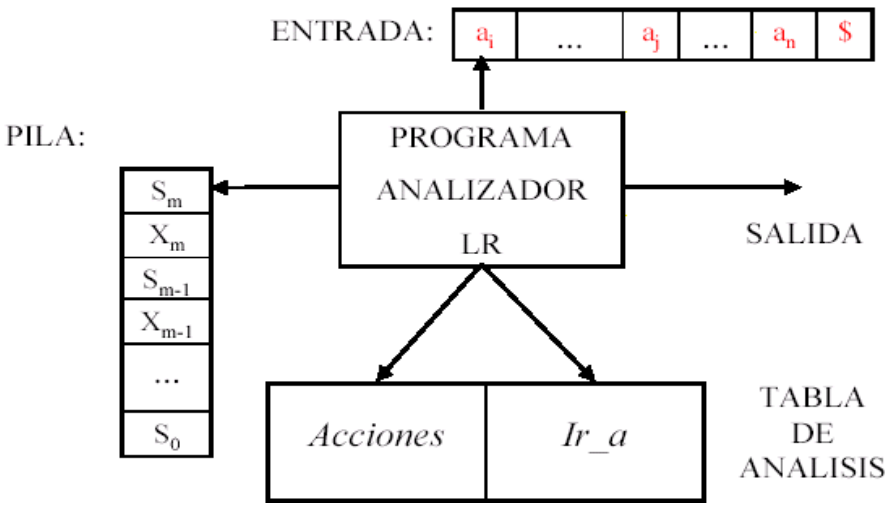
CONTENIDO

ANALIZADOR SINTÁCTICO LR

Es una técnica eficiente de análisis sintáctico ascendente que se puede utilizar para analizar una clase mas amplia de GLC



ESQUEMA DEL ANALIZADOR LR



ESQUEMA DEL ANALIZADOR LR

Entrada: Contiene la cadena a analizar seguida de $

Pila: Almacena una cadena de la forma :

$S_0X_0\ S_1X_1\ S_2X_2$…… S_mX_m

Donde:

$\mathbf{S_m}\rightarrow$ está en la cima y cada X_i es un símbolo gramatical

$\mathbf{S_i}\rightarrow$ cada $\mathbf{S_i}$ es un símbolo llamado estado

Al principio la pila solo contiene el símbolo inicial S_0

Tabla de Análisis Sintáctico:

Consta de 2 partes la función ACCION que indica una acción del analizador y la función GOTO que indica las transiciones del estado

Salida: Contiene las acciones realizadas por el analizador sintáctico

ALGORITMO DE ANÁLISIS SINTÁCTICO LR

Entrada:

Cadena w

Tabla de análisis sintáctico LR para la gramática G

Salida :

Si w Є L(G) un análisis sintáctico ascendente de w

Sino

Error

Método:

Inicialmente

S_0 está en la pila

w$ está en la entrada

Algoritmo:

```
Apuntar ae al primer símbolo de w $
REPEAT
BEGIN
     Sea:  S el estado de la  en la cima de la pila
           a el símbolo apuntado por ae
IF  Acción [S,a] = desplazar  S'   THEN
    BEGIN
         Meter a y después S' en la cima de la pila
         Avanzar ae  al siguiente símbolo de entrada
    END
ELSE
    IF  Acción  [S,a] = reducir  A→  β  THEN
        BEGIN
                Emitir la producción     A→  β
                Sacar  2* | β |  símbolos de la pila
                Sea  S' el estado que ahora esta en la cima  de la pila
                Meter  A y después  GOTO[ S', A] en la cima de la pila
         END
    ELSE
         IF  Acción [S,a]  = Aceptar THEN
              RETURN (Fin del análisis, el programa ha sido reconocido)
         ELSE
              Error()
END
```

EJEMPLO

Sea la gramática y su tabla de análisis SLR, analizar la cadena: id*id+id$

E → E + T | T 1,2

T → T * F | F 3,4

F → (E) | id 5,6

Estado	*Acción*						*Ir_a*		
	Id	+	*	(	)	$	**E**	**T**	**F**
0	d5			d4			1	2	3
1		d6				**Aceptar**			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

EJEMPLO

PILA	ENTRADA	ACCION
0	id*id+id$	Desplazar s5
0id5	*id+id$	Reducir F → id
0F3	*id+id$	Reducir T → F
0T2	*id+id$	Desplazar s7
0T2*7	id+id$	Desplazar s5
0T2*7id5	+id$	Reducir F → id
0T2*7F10	+id$	Reducir T → T * F
0T2	+id$	Reducir E → T
0E1	+id$	Desplazar s6
0E1+6	Id$	Desplazar s5
0E1+6id5	$	Reducir F → id
0E1+6F3	$	Reducir T → F
0E1+6T9	$	Reducir E → E + T
0E1	$	Aceptar

EJEMPLO

Entrada:
- Cadena w
- Tabla de análisis sintáctico LR para la gramática G

Salida :
Si w Є L(G) un análisis sintáctico ascendente de w
Sino Error

Método:
Inicialmente: S_0 está en la pila
w$ está en la entrada

Algoritmo:
Apuntar ae al primer símbolo de w $
REPEAT
BEGIN
Sea: S el estado de la en la cima de la pila
a el símbolo apuntado por ae
IF Acción [S,a] = desplazar S' THEN
BEGIN
Meter a y después S' en la cima de la pila
Avanzar ae al siguiente símbolo de entrada
END
ELSE
IF Acción [S,a] = reducir A→ β THEN
BEGIN
Emitir la producción A→ β
Sacar 2* | β | símbolos de la pila
Sea S' el estado que ahora esta en la cima de la pila
Meter A y después GOTO[S', A] en la cima de la pila
END
ELSE
IF Acción [S,a] = Aceptar THEN
RETURN (Fin del análisis, el programa ha sido reconocido)
ELSE
Error()
END

Sea la gramática y su tabla de análisis SLR, analizar la cadena: id*id+id$

E → E + T | T 1,2
T → T * F | F 3,4
F → (E) | id 5,6

Estado	Acción						Ir_a		
	Id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Aceptar			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

PILA	ENTRADA	SALIDA
0	id*id+id$	Desplazar 5
0id5	*id+id$	Reducir F→id
0F3	*id+id$	Reducir T→ F
0T2	*id+id$	Desplazar 7
0T2*7	id+id$	Desplazar 5
0T2*7id5	+id$	Reducir F→ id
0T2*7F10	+id$	Reducir T→ T*F
0T2	+id$	Reducir E→ T
0E1	+id$	Desplazar 6
0E1+6	Id$	Desplazar 5
0E1+6id5	$	Reducir F→ id
0E1+6F3	$	Reducir T→ F
0E1+6T9	$	Reducir E→ E + T

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Entrada:
- Cadena w
- Tabla de análisis sintáctico LR para la gramática G

Salida :
Si w Є L(G) un análisis sintáctico ascendente de w
Sino Error
Método:
Inicialmente: S_0 está en la pila
w$ está en la entrada

Algoritmo:
Apuntar ae al primer símbolo de w $
REPEAT
BEGIN
Sea: S el estado de la en la cima de la pila
a el símbolo apuntado por ae
IF Acción [S,a] = desplazar S' THEN
BEGIN
Meter a y después S' en la cima de la pila
Avanzar ae al siguiente símbolo de entrada
END
ELSE
IF Acción [S,a] = reducir A→ β THEN
BEGIN
Emitir la producción A→ β
Sacar 2* | β | símbolos de la pila
Sea S' el estado que ahora esta en la cima de la pila
Meter A y después GOTO[S', A] en la cima de la pila
END
ELSE
IF Acción [S,a] = Aceptar THEN
RETURN (Fin del análisis, el programa ha sido reconocido)
ELSE
Error()
END

Sea la gramática y su tabla de análisis SLR, analizar la cadena: ***(id+id*id$***

E → E + T | T 1,2
T → T * F | F 3,4
F → (E) | id 5,6

Estado	Acción						Ir_a		
	Id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Aceptar			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

PILA	ENTRADA	SALIDA

Gracias

UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 3

SESIÓN N° 12

ANALIZADORES SINTÁCTICOS LR(1)

Carlos Enrique Castillo Diestra

¿Sabes cómo se construye un analizador sintáctico ascendente?

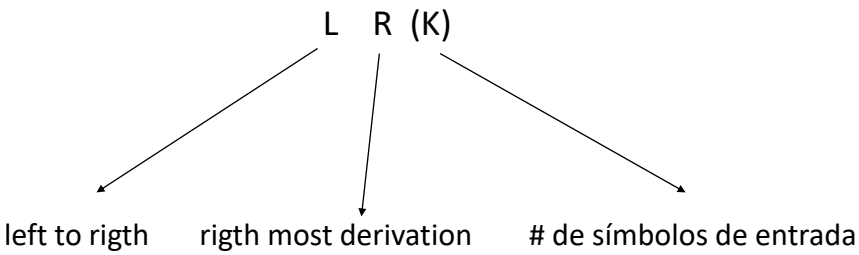
LOGRO

Al finalizar la Sesión, el estudiante construye la tabla de análisis sintáctico SLR(1) a partir de una GLC, utilizando el algoritmo para construir un analizador sintáctico SLR(1); presentando la tabla con orden y precisión

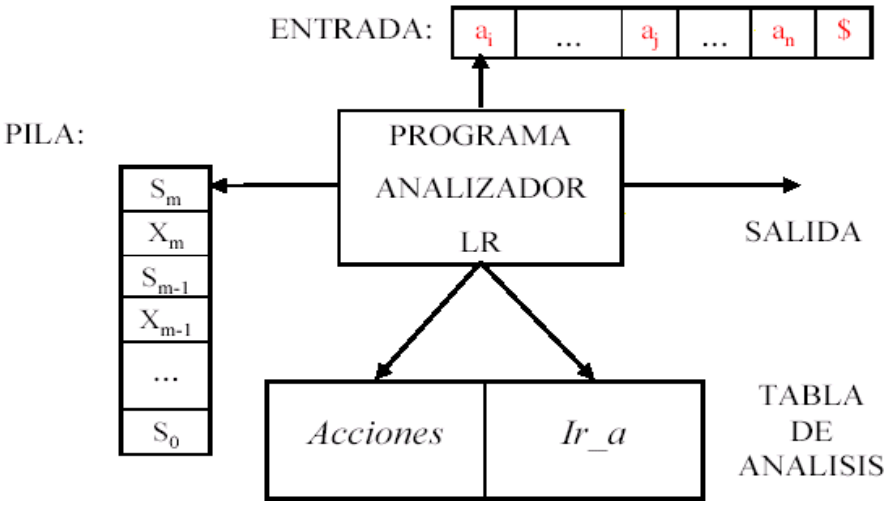
CONTENIDO

ANALIZADOR SINTÁCTICO LR

Es una técnica eficiente de análisis sintáctico ascendente que se puede utilizar para analizar una clase mas amplia de GLC



ESQUEMA DEL ANALIZADOR LR



ESQUEMA DEL ANALIZADOR SLR

Definiciones Básicas:

Gramática Aumentada:

Gramática a la que se agrega la regla:

S' → S , donde S es el símbolo inicial

Ejemplo: Para la siguiente GLC

E → E + T | T
T → T * F | F
F → (E) | id

Su gramática aumentada es:

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

ESQUEMA DEL ANALIZADOR SLR

- **Ítem:** regla de producción que contiene un punto en el consecuente.
  Representa que parte de una producción han sido reconocida.

Ejemplo:

Para la regla:

$A \rightarrow b\ C$

Se tienen los siguientes ítems:

$A \rightarrow .\ b\ C$
$A \rightarrow b\ .\ C$
$A \rightarrow b\ C\ .$

ESQUEMA DEL ANALIZADOR SLR

Función Clausura:

- Se define sobre un conjunto de Ítem I, y genera otro conjunto de Ítem según las reglas siguientes:
  - I se agrega a clausura(I)
  - Si $A \rightarrow \alpha.B\gamma \in$ clausura(I) y $\exists$ la producción $B \rightarrow \beta$, entonces $B \rightarrow .\beta$ se agrega a clausura(I)

Función Goto:

- Se define sobre un conjunto de ítems I y un símbolo X de la gramática.
- Es la clausura de ítems de la forma $A \rightarrow \alpha X.\beta$ tal que ítem de la forma $A \rightarrow \alpha.X\beta \in I$

ESQUEMA DEL ANALIZADOR SLR

Colección Canónica:

- Colecciones de todos los conjuntos de ítems para determinar los estados del AFD.
- $C=\{I_0,I_1,I_2,...\}$

Construcción:

- Ampliar la gramática con S' → S
- C = Clausura({S'→.S}) = I_0
- $\forall I_i \in C$ y $\forall X \in (N \cup \Sigma)$ calcular goto(I_i,X). Si este conjunto no es vacío, y no pertenece a la colección C, se añade a él.
- Repetir el paso anterior hasta que no se pueda incorporar nuevos conjuntos a la colección C.

ALGORITMO PARA CONSTRUIR LA TABLA DE ANÁLISIS SINTÁCTICO SLR

1. Si $A \rightarrow \alpha.a\beta \in Ii$ y goto (Ii,a) = Ij
   hacer ACCION[i,a]= Desplazar j
2. Si $A \rightarrow \alpha. \in Ii$ entonces
   hacer ACCION[i,a]=reducir $A \rightarrow \alpha$
   para todo $a \in$ (follow(A)) , $A \neq S'$
3. Si $S' \rightarrow S. \in Ii$ entonces ACCION[i,$]=Aceptar
4. Si goto [Ii,A]= Ij entonces goto [i ,A]= j
5. Las entradas vacías son errores
6. Estado inicial quien contenga S'→S.

CONSTRUCCIÓN DE LA TABLA DE ANÁLISIS SINTÁCTICO SLR

1. Aumentar la gramática
2. Obtener la colección canónica de la gramática aumentada
3. Calcular los conjuntos primero y siguiente para cada no terminal de la grámatica.
4. Aplicar el algoritmo de construcción de la tabla de análisis sintáctico SLR.

EJEMPLO

Construir la tabla de análisis sintáctico SLR de la siguiente gramática:

E → E + T | T
T → T * F | F
F → (E) | id

1. Aumentar la gramática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

EJEMPLO

1. Aumentar la gramática

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

2. Obtener la colección canónica de la gramática aumentada

I_0 = Clausura({E'→.E}) = { E'→.E
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id }

I_1 = goto(I_0,E) = { E'→E.
E →E.+T }

I_2 = goto(I_0,T) = { E→T.
T→T.*F }

I_3 = goto(I_0,F) = { T→F. }

I_4 = goto(I_0,() = { F→(.E)
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id }

I_5 = goto(I_0,id) = { F→id. }

I_6 = goto(I_1,+) = { E→E+.T
T→.T*F
T→.F
F→.(E)
F→.id}

I_7 = goto(I_2,*) = { T→T*.F
F→.(E)
F→.id }

EJEMPLO

I_8 = goto(I_4,E) = { F→(E.)
E→E.+T }

I_2 = goto(I_4,T)
{ E→T.
T→T.*F }

I_3 = goto(I_4,F)
{ T→F. }

I_4 = goto(I_4,()
{ F→(.E)
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id }

I_5 = goto(I_4,id)

I_9 = goto(I_6,T) = { E→E+T.
T→T.*F }

I_3 = goto(I_6,F)

I_4 = goto(I_6,()

I_5 = goto(I_6,id)

I_{10} = goto(I_7,F) = { T→T*F. }

I_4 = goto(I_7,()

I_5 = goto(I_7,id)

I_{11} = goto(I_8,)) = { F→(E). }

I_6 = goto(I_8,+)

I_7 = goto(I_9,*)

EJEMPLO

3. Calcular los conjuntos primero y siguiente para cada no terminal de la grámatica.

No terminal	PRIMERO	SIGUIENTE
E'	a, (	$
E	a, (	+ ,) , $
T	a, (	* ,) , + , $
F	a, (	* ,) , + , $

EJEMPLO

4. Aplicar el algoritmo de construcción de la tabla de análisis sintáctico SLR.

E → E + T | T 1,2
T → T * F | F 3,4
F → (E) | id 5,6

Estado	*Acción*						*Ir_a*		
	Id	+	*	(	)	$	**E**	**T**	**F**
0	d5			d4			1	2	3
1		d6				**Aceptar**			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

EJEMPLO

Colección canónica

I_0 = Clausura({E'→.E}) =
{ E'→.E
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id }

I_1 = goto(I_0,E) = { E'→E.
E→E.+T }

I_2 = goto(I_0,T) = { E→T.
T→T.*F }

I_3 = goto(I_0,F) = { T→F. }

I_4 = goto(I_0,() = { F→(.E)
E→.E+T
E→.T
T→.T*F
T→.F
F→.(E)
F→.id }

I_5 = goto(I_0,id) = { F→id. }

I_6 = goto(I_1,+) = { E→E+.T
T→.T*F
T→.F
F→.(E)
F→.id}

I_7 = goto(I_2,*) = { T→T*.F
F→.(E)
F→.id }

I_8 = goto(I_4,E) = { F→(E.)
E→E.+T }

I_2 = goto(I_4,T)

I_3 = goto(I_4,F)

I_4 = goto(I_4,()

I_5 = goto(I_4,id)

I_9 = goto(I_6,T) ={ E→E+T.
T→T.*F }

I_3 = goto(I_6,F)

I_4 = goto(I_6,()

I_5 = goto(I_6,id)

I_{10} = goto(I_7,F) = { T→T*F. }

I_4 = goto(I_7,()

I_5 = goto(I_7,id)

I_{11} = goto(I_8,)) ={ F→(E). }

I_6 = goto(I_8,+)

I_7 = goto(I_9,*)

Gramática
E → E + T | T 1,2

T → T * F | F 3,4

F → (E) | id 5,6

No terminal	PRIMERO	SIGUIENTE
E'	a, (	$
E	a, (	+ ,) , $
T	a, (	* ,) , + , $
F	a, (	* ,) , + , $

1. Si A→ α.aβ ∈ Ii y goto (Ii,a) = Ij
   hacer ACCION[i,a]= Desplazar j
2. Si A→ α. ∈ Ii entonces
   hacer ACCION[i,a]=reducir A→ α
   para todo a ∈ (follow(A)) , A≠S'
3. Si S'→S. ∈ Ii entonces ACCION[i,$]=Aceptar
4. Si goto [Ii,A]= Ij entonces goto [i ,A]= j
5. Las entradas vacías son errores
6. Estado inicial quien contenga S'→S.

Estado	ACCION						GOTO		
	Id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Acep			
2		r2			r2	r2			
3									
4									
5		r6	r6		r6	r6			
6									
7									
8									
9									
10									
11									

CONCLUSIONES

- Pueden construirse analizadores sintácticos LR para reconocer prácticamente todas las construcciones de lenguajes de programación para las cuales puedan escribirse gramáticas libres de contexto.
- La principal desventaja del método LR es que es demasiado trabajo construir un analizador sintáctico LR en forma manual para una gramática común de un lenguaje de programación. Se necesita una herramienta especializada: un generador de analizadores sintácticos LR, como por ejemplo BISON.

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Construir la tabla de análisis SLR(1) para la siguiente gramática

A → A = B
|B

B → aB
|R

R → Rb
|b

1. Si X es un terminal, entonces PRIMERO$(X) = \{X\}$.
2. Si $X \to \epsilon$ es una producción, entonces se agrega ϵ a PRIMERO(X).
3. Si X es un no terminal y $X \to Y_1 Y_2 \cdots Y_k$ es una producción para cierta $k \geq 1$, entonces se coloca a en PRIMERO(X) si para cierta i, a está en PRIMERO(Y_i), y ϵ está en todas las funciones PRIMERO(Y_1), ..., PRIMERO(Y_{i-1}); es decir, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. Si ϵ está en PRIMERO(Y_j) para todas las $j = 1, 2, \ldots, k$, entonces se agrega ϵ a PRIMERO(X). Por ejemplo, todo lo que hay en PRIMERO(Y_1) se encuentra sin duda en PRIMERO(X). Si Y_1 no deriva a ϵ, entonces no agregamos nada más a PRIMERO(X), pero si $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, entonces agregamos PRIMERO(Y_2), y así sucesivamente.

1. Póngase \$ en SIGUIENTE(S), donde S es el símbolo inicial y \$ es el delimitador derecho de la entrada.
2. Si hay una producción $A \to \alpha B \beta$, entonces todo lo que esté en PRIMERO(β) excepto ϵ se pone en SIGUIENTE(B).
3. Si hay una producción $A \to \alpha B$ o una producción $A \to \alpha B \beta$, donde PRIMERO(β) contenga ϵ (es decir, $\beta \stackrel{*}{\Rightarrow} \epsilon$), entonces todo lo que esté en SIGUIENTE(A) se pone en SIGUIENTE(B).

Algoritmo para construir la tabla de análisis sintáctico SLR
1. Si A→ α.aβ ∈ Ii y goto (Ii,a) = Ij
   hacer ACCION[i,a]= Desplazar j
2. Si A→ α. ∈ Ii entonces
   hacer ACCION[i,a]=reducir A→ α
   para todo a ∈ (follow(A)) , A≠S'
3. Si S'→S. ∈ Ii entonces ACCION[i,$]=Aceptar
4. Si goto [Ii,A]= Ij entonces goto [i ,A]= j
5. Las entradas vacías son errores
6. Estado inicial quien contenga S'→S.

Gracias

UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 4

SESIÓN N° 13

COMPROBADOR DE TIPOS

Carlos Enrique Castillo Diestra

1

```
//en C++
.
.
cout<<15%2;
.
.
.
```

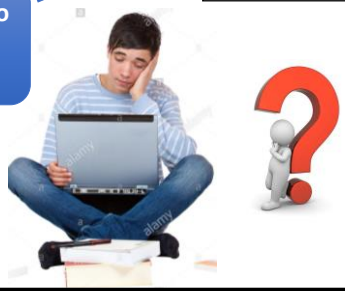
2

```
//en C++
.
.
cout<<15.2%2;
.
.
.
```

3

```
//en C++
.
.
 cout<<20.3%2.4;
.
.
.
```

¿Sabes cuál fragmento es semánticamente correcto?



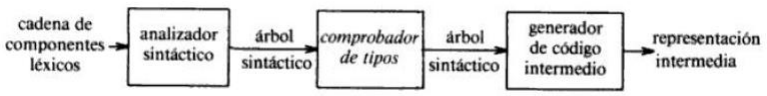
LOGRO

Al finalizar la Sesión, el estudiante construye un comprobador de tipos para un operador; a partir de una gramática de libre contexto; presentando el desarrollo del comprobador de tipos con orden y precisión

CONTENIDO

COMPROBADOR DE TIPOS

- Un compilador debe comprobar si el programa fuente sigue tanto las convenciones sintácticas como las semánticas del lenguaje fuente.
- La *comprobación de tipos* utiliza reglas lógicas para razonar acerca del comportamiento de un programa en tiempo de ejecución. En específico, asegura que los tipos de los operandos coincidan con el tipo esperado por un operador.
- Por ejemplo:
  - El operador && en Java espera que sus dos operandos sean booleanos; el resultado también es de tipo booleano.
  - El operador aritmético modulo % en C++ exige operandos de tipo entero.
- Hay comprobador de tipos cuando hay un operador que me indica alguna operación.
- Ubicación del comprobador de tipos:



COMPROBADOR DE TIPOS

Si se tiene, en C++

```
int A, B;
float C = 12.8, D = 11.9;
```

A = C/2; el valor de A es 6

cout<<C/2; escribe 6.4

B = C * D; C*B da 152.32, pero el valor de B es 152

No generan error semántico

A = C % D; error semántico, C y D tiene que ser del tipo int

COMPROBADOR DE TIPOS

Se tiene en JAVA:

```
int A, B;
float C =12.8f, D = 11.9f;
```

A = C/2 — error semántico, A tiene que ser del tipo float

System.out.print(C/2); — escribe 6.4

B = C * D; — error semántico, B tiene que ser del tipo float

A = C % D; — error semántico, C y D deben ser del tipo int

¿Qué ocurre?

CONSTRUCCIÓN DE UN COMPROBADOR DE TIPOS

Gramática de libre contexto:

```
exp  :   exp '+' ter
       |   exp '-' ter
       |   ter
       ;
ter    :  ter '*' fac
       |  ter '/' fac
       |  fac
       ;
fac  :  '(' exp ')'
       | IDENT
       |  CTE_ENT
              |  CTE_REAL
       ;
```

Se representa de la siguiente forma

```
exp  :   exp   '+'    ter
 $$       $1    $2   $3
```

EJEMPLO

Si se tiene:

7 + 2 = ?

7.5 + 3 = ?

5 + 4.5 = ?

2.5 + 3. 5 = ?

A + 2 = ?

4 + B = ?

C + 3.5 = ?

4.5 + D = ?

R + S = ?

Si
A =2
B=3
C=3
D=5
R=3
S=5

¿Cuál es el resultado?

9 → CTEENT

10.5, 9.5, 6.0 → CTEREAL

4, 7 → CTEENT

6.5, 9.5 → CTEREAL

8 → CTEENT

EJEMPLO

```
exp : exp '+' ter { if ((($1.tipo==CTEENT || $1.tipo==VARINT) && ($3.tipo==CTEENT || $3.tipo==VARINT))
                        $$.tipo=CTEENT;

                    if ((($1.tipo==CTEENT || $1.tipo==VARINT) && ($3.tipo==CTEREAL || $3.tipo==VARREAL))
                        $$.tipo=CTEREAL;

                   if ((($1.tipo==CTEREAL || $1.tipo==VARREAL) && ($3.tipo==CTEENT || $3.tipo==VARINT))
                        $$.tipo=CTEREAL;

                   if ((($1.tipo==CTEREAL || $1.tipo==VARREAL) && ($3.tipo==CTEREAL || $3.tipo==VARREAL))
                        $$.tipo=CTEREAL;

                   if ($$.tipo==CTEENT || $$.tipo==CTEREAL )
                        { $$.addr=$1.addr;
                          code[ip]=OP_SUMA;
                          ip++;
                        }
                    else
                        {
                         $$.tipo=TERROR;
                         ParserError("Tipos incompatibles");
                        }
                }
```

EJEMPLO

Puesto que las construcciones de los lenguajes, como las proposiciones, carecen típicamente de valores, se les puede asignar el tipo básico especial vacío. Si se detecta un error dentro de una proposición, el tipo asignado a la proposición es TERROR

```
Sent    : ID '=' Exp                {if  ($1.tipo == $3.tipo
                                          $$.tipo = TVACIO
                                     else
                                          $$.tipo=TERROR;  }

        | IF Exp THEN Sent1         {if  ($2.tipo == BOOLEAN
                                          $$.tipo = $4.tipo
                                     else
                                          $$.tipo=TERROR;  }

        | WHILE Exp DO Sent1        {if  ($2.tipo == BOOLEAN
                                          $$.tipo = $4.tipo
                                     else
                                          $$.tipo=TERROR;  }
```

CONCLUSIONES

- Un compilador debe informar de un error si se aplica un operador a un operando incompatible; por ejemplo, si se suman una variable tipo cadena con una variable tipo matriz. Esto se realiza con el comprobador de tipos.

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Escribir el comprobador de tipo para los operadores de las siguientes gramáticas:

1. Exp : Exp1 '%' Exp2 % es el operador módulo

2. Res : IDENT ^ ENTERO ^ es el operador potencia


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

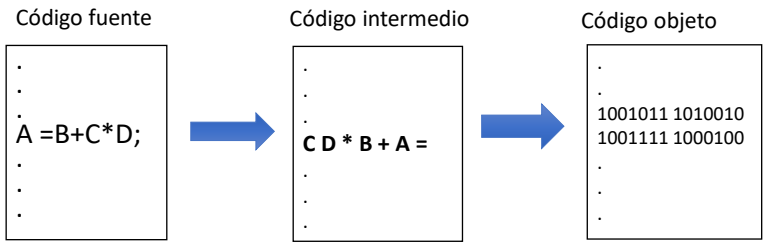
COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 4

SESIÓN N° 14

GENERACIÓN DE CÓDIGO INTERMEDIO

Carlos Enrique Castillo Diestra



LOGRO

Al finalizar la Sesión, el estudiante genera el código intermedio usando formas internas orientadas a pila; a partir de un código fuente; presentando el código intermedio con orden y precisión

CONTENIDO

FORMAS INTERNAS

- Las formas internas permiten generar una representación intermedia del programa fuente, lo cual brinda una cierta ventaja debido a que el código intermedio es independiente de la máquina.
- A partir del código intermedio tenemos dos opciones:
  1. Ejecutarlo mediante una máquina virtual (Intérprete).
  2. Construir un generador de código que transforme la forma interna al código máquinas.
- En las formas internas se distinguen dos elementos:
  - **Operadores.**- Acciones a realizar.
  - **Operandos**.- Datos sobre los cuales se realiza las acciones.

TIPOS DE FORMAS INTERNAS

Formas internas orientada a variables temporales

Los resultados intermedios se almacena en variables temporales que se generan en la compilación. La notación más utilizada es la notación de cuádruplos. En ella cada elemento de la forma interna tienen la siguiente estructura:

(op, a1, a2, R)

Donde: op : Operador.
a1 : Argumento 1.
a2 : Argumento 2.
R : Resultado.

La expresión: A * B + C * D queda:

(*, A, B, R1)
(*, C, D, R2)
(+, R1, R2, R)

TIPOS DE FORMAS INTERNAS

Formas internas orientada a pila

Es una técnica conocida como notación polaca. En ella aparecen los operandos antes que los operadores. Cada elemento de la forma interna es un operando o un operador.

Ejm.:

La expresión: A * B + C * D
queda: A, B, *, C, D, *, +

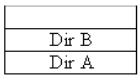
TIPOS DE FORMAS INTERNAS

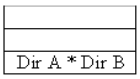
Operador Carga.

Fuente: A * B + C

F.I.: A, B, *, C, +

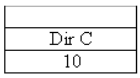
F.I.: Dir A, Dir B, *, Dir C, +

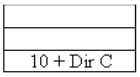




Si A=2 y B=5

Valor 10





No se puede ejecutar un valor con una referencia, pero sí dos referencias o valores juntos.

F.I.: CARGA, Dir A, CARGA, Dir B, *, CARGA, Dir C, +

SEMÁNTICA DE UN OPERADOR

- La semántica de un operador son las acciones que hay que realizar en el momento de la ejecución, es decir cuando es un operador la siguiente acción a realizar.

- Para formalizar la semántica de los operadores se define:
  - »CODE.- Es un arreglo que contiene la forma interna.
  - »IP.- Es el índice por el cual estamos ejecutando la forma interna.

- Además para la forma interna orientada a la pila se define:
  - »STACK.- Arreglo que el contiene en la forma interna.
  - »SP.- Índice correspondiente al tipo de stack.

SEMÁNTICA DE UN OPERADOR

- **Operador "+":**

  stack [SP - 1] = stack [SP - 1] + stack [SP]

  SP = SP – 1

  IP = IP + 1

- **Operador "*":**

  stack [SP - 1] = stack [SP - 1] * stack [SP]

  SP = SP – 1

  IP = IP + 1

- **Operador "CARGA":**

  SP = SP + 1

  stack [SP] = CODE [IP + 1]

  IP = IP + 2

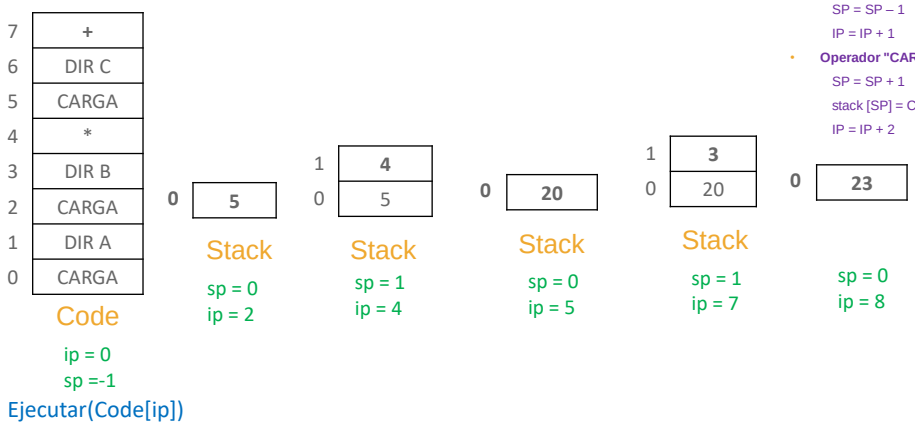
SEMÁNTICA DE UN OPERADOR

- Ejemplo:

  Mostrar la ejecución de : A* B + C

  donde los valores que tomarán las variables serán: A = 5, B = 4, C = 3.

F.I.: CARGA, Dir A, CARGA, Dir B, *, CARGA, Dir C, +



- **Operador "+":**
  stack [SP - 1] = stack [SP - 1] + stack [SP]
  SP = SP – 1
  IP = IP + 1
- **Operador "*":**
  stack [SP - 1] = stack [SP - 1] * stack [SP]
  SP = SP – 1
  IP = IP + 1
- **Operador "CARGA":**
  SP = SP + 1
  stack [SP] = CODE [IP + 1]
  IP = IP + 2

SEMÁNTICA DE UN OPERADOR

- Ejemplo: Modificar el ejemplo, agregando **escribir**

  Mostrar la ejecución de : escribir A* B + C

  donde los valores que tomarán las variables serán: A = 5, B = 4, C = 3.

F.I.: CARGA, Dir A, CARGA, Dir B, *, CARGA, Dir C, +, ESCRIBIR

	Code
8	ESCRIBIR
7	+
6	DIR C
5	CARGA
4	*
3	DIR B
2	CARGA
1	DIR A
0	CARGA

ip = 0
sp =-1
Ejecutar(Code[ip])

	Stack
0	5

sp = 0
ip = 2

	Stack
1	4
0	5

sp = 1
ip = 4

	Stack
0	20

sp = 0
ip = 5

	Stack
1	3
0	20

sp = 1
ip = 7

	Stack
0	23

sp = 0
ip = 8

Stack
sp = -1
ip = 9

- **Operador "+":**
  stack [SP - 1] = stack [SP - 1] + stack [SP]
  SP = SP – 1
  IP = IP + 1
- **Operador "*":**
  stack [SP - 1] = stack [SP - 1] * stack [SP]
  SP = SP – 1
  IP = IP + 1
- **Operador "CARGA":**
  SP = SP + 1
  stack [SP] = CODE [IP + 1]
  IP = IP + 2

- **Operador "ESCRIBIR":**
  ESCRIBIR EN PANTALLA (Stack[SP])
  SP = SP – 1
  IP = IP + 1

CONCLUSIONES

- Para un programa fuente a código objeto, un compilador puede construir una o más representaciones intermedias, las cuales pueden tener una variedad de formas.
- El código intermedio es código abstracto independiente de la máquina para la que se generará el código objeto.

PREGUNTAS FINALES

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Para la siguiente secuencia A + B > C

donde los valores que tomarán las variables serán: A = 2, B = 5, C = 1

1. Escribir su forma interna
2. Mostrar la ejecución de la forma interna
3. Escribir la semántica del operador >


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE

UPN

UNIVERSIDAD PRIVADA DEL NORTE

COMPILADORES Y LENGUAJES DE PROGRAMACIÓN

UNIDAD 4

SESIÓN N° 15

GENERACIÓN DE CÓDIGO OBJETO

Carlos Enrique Castillo Diestra

¿Sabes como se genera el código objeto?

LOGRO

Al finalizar la Sesión, el estudiante genera el código objeto; a partir de un código en ensamblador; presentando el programa objeto con orden y precisión

CONTENIDO

EL COMPUTADOR SIC (COMPUTADOR HIPOTÉTICO DE ENSEÑANZA)

Memoria

- Consta de bytes de 8 bits. Tres bytes consecutivos forman una palabra de 24 bits.
- Todas las direcciones son de bytes. Las palabras se direccionan por la posición de su byte menor.
- El computador tiene un total de 2^{15} bytes de memoria principal.

Registros

- SIC cuenta con cinco registros cada uno con un uso especial, cada registro tiene una longitud de 24 bits.

Nemónico	Número	Uso
A	0	El registro A, acumulador para operaciones aritméticas.
X	1	Registro índice para direccionar.
L	2	Registro de enlace, la instrucción de salto a sub rutina JSUB almacena la dirección de retorno en este registro.
PC	8	Contador de programa; contiene la dirección de la siguiente instrucción que se va a traer para su instrucción.
SW	9	Palabras de estados; contiene diversa información incluido un código de dirección.

EL COMPUTADOR SIC (COMPUTADOR HIPOTÉTICO DE ENSEÑANZA)

Formato de datos. Soporta:

- ***Enteros***, 24 bits, Se usa la representación de complemento A2 para valores negativos.
- ***Carácter.-*** Se almacenan por medio del código ASCII

Formato de instrucciones

- Todas las instrucciones de máquinas tienen el formato de 24 bits.

Código de la Operación	X	Dirección
8 bits	1 bit	15 bits

X : Modo de direccionamiento.

Modos de direccionamiento

Modo	Indicación	Dirección Objetivo
Directo	X = 0	TA = dirección
Indirecto	X = 1	TA = dirección + (X)

EL COMPUTADOR SIC (COMPUTADOR HIPOTÉTICO DE ENSEÑANZA)

Conjunto de Instrucciones.

- ***Instrucciones de carga y almacenamiento.***
  LDA, LDX, STA, STX, etc.

- ***Instrucciones de operaciones aritméticas.***
  ADD, SUB, MUL, DIV, etc.
  Todas las operaciones aritméticas se relacionan con el registro A y una palabra en memoria y dejan el resultado en el registro.

- ***Comparación***.
  COMP
  Compara el valor del registro con una palabra en memoria; esta instrucción establece un código de condición para indicar el resultado.

- ***Salto incondicional.***
  JLT, JEQ, JGT
  Prueba el código de condición y realiza el saldo correspondiente.

EL COMPUTADOR SIC (COMPUTADOR HIPOTÉTICO DE ENSEÑANZA)

Conjunto de Instrucciones (continuación)

- ***Enlace de subrutinas***

  JSUB: Salta a la subrutina y coloca la dirección de retorno en el registro L.

  RSUB: Regresa saltando a la dirección contenida en el registro L

- ***Entrada y salida.***
  - ✓ Se realiza transfiriendo un byte cada vez, hacia o desde los 8 bits situados más a la derecha del registro A. Cada dispositivo tiene asignada un código único de 8 bits.
  - ✓ Hay tres instrucciones de entrada/salida:

    TD, RD, WD.
  - ✓ TD, Prueba si el dispositivo seleccionado está listo para enviar o recibir un byte de datos. Cuando el dispositivo esté listo se utiliza RD la lectura de datos o WD para escritura de datos.

ENSAMBLADOR PARA EL COMPUTADOR SIC

Además de las instrucciones de máquina se emplean las siguientes instrucciones para el ensamblador:

Start.- Especifica nombre y dirección de inicio del programa.

End.- Indica el fin del programa fuente y opcionalmente especifica la primera instrucción ejecutable del programa.

Byte.- Generado una constante de caracteres o hexadecimal que ocupa tantos bytes como sean necesarios para representarla.

Word.- Genera una constante entera de una palabra.

RESB.- Reserva el número indicado de bytes para un área de datos.

RESW.- Reserva el número indicado de palabras para una área de datos.

ENSAMBLADOR PARA EL COMPUTADOR SIC

Para efectuar la traducción del programa a código objeto es necesario realizar las siguientes funciones (no necesariamente en el orden que se indica)

1. Conversión de los códigos de operación nemónicas a sus equivalentes en lenguaje de máquina.
2. Conversión de los operandos simbólicos a sus direcciones de máquina y equivalente.
3. Construcción de las instrucciones de máquina en el lugar adecuado.
4. Conversión de las constantes de datos especificadas en el programa fuente a sus representaciones internas de máquinas.
5. Escritura del programa objeto.

FORMATO BÁSICO DEL PROGRAMA OBJETO

Los formatos utilizados son los siguientes:

a. Registro de encabezamiento.

Col 1	H
Col 2 – 7	Nombre del programa.
Col 8 – 13	Dirección de inicio del programa objeto.
Col 14 – 19	Longitud en bytes que el programa objeto.

b. Registro de texto.

Col 1	T
Col 2 – 7	Dirección de inicio de código objeto en este registro.
Col 8 – 9	Longitud en bytes del código objeto en este registro.
Col 10 – 69	Código objeto.

c. Registro de fin.

Col 1	E
Col 2 – 7	Dirección de la primera instrucción ejecutada del programa objeto.

EJEMPLO

Genere el código objeto para el siguiente programa:

UNO	START	1536
INICIO	STL	RETDIR
	JSUB	BORRA
	LDA	QUINCE
	STA	SEPARA, X
QUINCE	WORD	15
RETDIR	BYTE	c ‘ABC’
SEPARA	RESW	2000
CERO	WORD	0
BORRA	LDA	CERO
	RSUB	
	END	INICIO

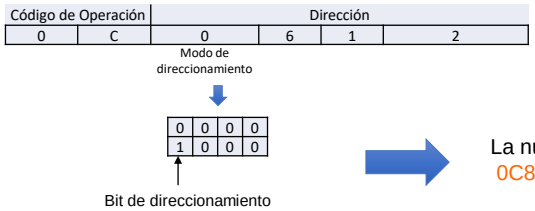
Códigos de operación nemónicos

Nemónico	Código
JSUB	48
LDA	00
RSUB	4C
STA	0C
STL	14

EJEMPLO

	UNO	START	1536	
0600	INICIO	STL	RETDIR	14060F
0603		JSUB	BORRA	481D85
0606		LDA	QUINCE	00060C
0609		STA	SEPARA, X	0C0612
060C	QUINCE	WORD	15	00000F
060F	RETDIR	BYTE	c ‘ABC’	414243
0612	SEPARA	RESW	2000	
1D82	CERO	WORD	0	000000
1D85	BORRA	LDA	CERO	001D82
1D88		RSUB		4C0000
1D8B		END	INICIO	

En el direccionamiento directo tenemos 0C0612, pero como es un direccionamiento indirecto debemos de modificar el 9 bit comenzando por la izquierda:



La nueva dirección sería:
0C8612

EJEMPLO

H ^ UNO_ _ _ ^ 000600 ^ 00178B
T ^ 000600 ^ 12 ^ 14060F ^ 481D85 ^ 00060C ^ 0C8612 ^ 00000F ^ 414243
T ^ 001D82 ^ 09 ^ 000000 ^ 001D82 ^ 4C0000
E ^ 000600

T ^ 000600 ^ 12 ^ 14060F ^ 481D85 ^ 00060C ^ 0C8612 ^ 00000F ^ 414243
(1) (2) (3) (4) (5) (6)

Cada columna es igual a ½ Byte por lo que el tamaño sería la cantidad de columnas divididas entre 2 para obtener el número de bytes en decimal luego tenemos que convertirlo a hexadecimal.

6 (grupos) * 6 (columnas de cada grupo) = 36
36 (columnas) / 2 = 18 bytes
18 (decimal) = 12 (hexadecimal)

CONCLUSIONES

El generador de código objeto se encarga de tomar como entrada el código intermedio generado por el front-end, y producir código objeto de la arquitectura destino para luego entrar en la fase de optimización de código.

BIBLIOGRAFIA

Aho, A., Lam, M., Sethi, R., Ullman, J. (2008). Compiladores: Principios, Técnicas y Herramientas. México: Person Education.

TAREA EN LINEA

Genere el código objeto para el siguiente programa:

DOS	START	1800
INICIO	LDA	VALOR
	ADD	NUEVO
	STA	SEGURO
	JSUB	SALTO
VALOR	WORD	28
NUEVO	WORD	22
SEGURO	RESB	700
SALTO	WORD	12
	RSUB	
	END	INICIO

Códigos de operación nemónicos

Nemónico	Código
JSUB	48
LDA	00
RSUB	4C
STA	0C
ADD	10

Gracias

UPN
UNIVERSIDAD
PRIVADA
DEL NORTE