Ejemplo Recursividad a Izquierdas

$E \to E * T \mid E + T \mid T$

(α_1 = $*T$, α_2 = $+T$, β = T)

$E \to TE'$

$E' \to *TE' \mid +TE' \mid \lambda$

$\Sigma_T \; \Sigma_{NT}$

$r \to r\alpha \mid \beta$

$r \to \beta r'$

$r' \to \alpha r' \mid \lambda$

$E \to E * T \mid T \to E \to TE'$

$E' \to *TE' \mid \lambda$

$E \to E + T \mid T \to E \to TE'$

$E' \to +TE' \mid \lambda$

Ejemplo de Factorizacion

$S \to \text{if } C \text{ then } S \text{ else } S$

$S \to \text{if } C \text{ then } S$

$\Big\}$ $S \to \text{if } C \text{ then } S \; A'$

$A' \to \text{else } S \mid \lambda$

$S \to \text{repeat } S \text{ until } C$

$S \to \text{repeat } S \text{ forever}$

$\Big\}$ $S \to \text{repeat } S \; B'$

$B' \to \text{until } C \mid \text{forever}$

EJEMPLO GRAMATICA

Se desea realizar un traductor para un lenguaje orientado a las operaciones booleanas.
El lenguaje tiene dos sentencias: asignación y declaración

La sentencia de asignación realiza operaciones sobre constantes y variables de cadenas para asignar el resultado de la evaluación a una variable. Por ejemplo:

a = "0001" OR b AND "110"

La instrucción anterior da como resultado la asignación a la variable a (se supone que a y b han sido declaradas previamente), si b tiene el valor "010" entonces la variable a toma el valor "011".

AND 010 / 110 / 010

OR 0001 / 011

Para declarar una variable se utiliza la sentencia:

variable = BASE (numero, cadena_inicial)

El número determina la base numérica (un número natural hasta 32). La cadena_inicial es el valor de inicialización de la variable

CONSIDERACIONES

- La longitud máxima de una secuencia es de 100 dígitos
- Las operaciones lógicas se realizan sin precedencia entre los operadores y evaluandose de izquierda a derecha
- Una operación con una variable no declarada debe producir un error
- Si en una operación, las longitudes de las secuencias de los operandos son distintas, entonces las secuencias se truncan a la de menor longitud (se eliminan simbolos de derecha a izquierda).
- Las operaciones entre operandos de distinta base deben producir un error
- Para realizar operaciones entre cadenas que no están en base binaria, estas deben pasarse previamente a binario y obtenido el resultado se transforma a la base de origen.
- Las operaciones lógicas válidas son: AND, XOR, OR, NOT
- La función NOT es unaria y realiza la negación del argumento de su derecha.

EJEMPLO DE PROGRAMA VALIDO

1 → a = base (2, 0001)

2 → a = a AND NOT 0010

E → E O E

↳ operador

SOLUCION

$S \to AS \mid DS \mid \lambda$

$A \to var = E \mid var = D$

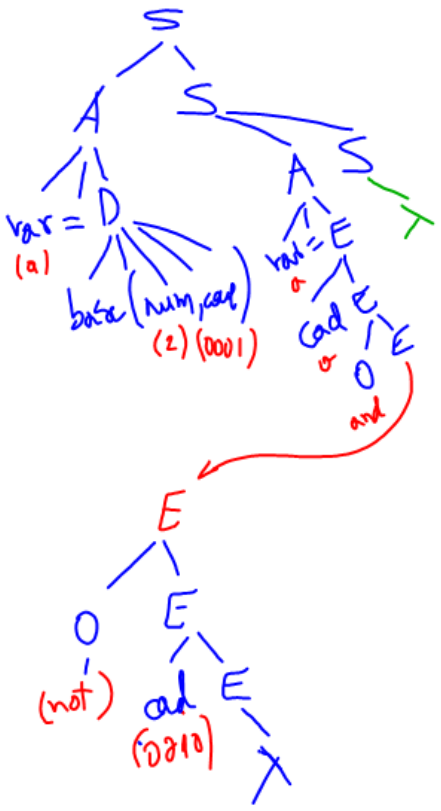
$D \to base\ (num, cad)$

$E \to cad\ E \mid var\ E \mid O\ E \mid \lambda$

$O \to not \mid xor \mid and \mid or$

D ⇒ a = base (2, 0001)

A ⇒ a = a and not 0010



Otra solución mas optimizada

$S \to id = A \mid id = AS$

$A \to E \mid base\ (num, cad)$

$E \to O \mid O\ P\ E$

$O \to N\ cad \mid N\ id$

$N \to not\ N \mid \lambda$

$P \to and \mid xor \mid or$

Resolucion de Ambigüedades

$S \to id = A \mid id = A\,S$

$A \to E \mid base\,(num, cad)$

$E \to O \mid O\,P\,E$

$O \to N\,cad \mid N\,id$

$N \to not\ N \mid \lambda$

$P \to and \mid xor \mid or$

FACTORIZAR

$S \to id = A\,S'$

$S' \to S \mid \lambda$

$E \to O\,E'$

$E' \to P\,E \mid \lambda$

$O \to N\,O'$

$O' \to cad \mid id$

Calculo de Conjuntos Primero → Conjunto de terminales que inician las cadenas

$E \to TE'$

$E' \to +TE' \mid \lambda$

$T \to FT'$

$T' \to *FT' \mid \lambda$

$F \to (E) \mid id$

$PRI(E) = PRI(T) = PRI(F) = \{(, id\}$

$PRI(T) = PRI(F) = \{(, id\}$

$PRI(E') = \{+, \lambda\}$

$PRI(T') = \{*, \lambda\}$

$PRI(F) = \{(, id\}$

$SIG(E) = \{\$\} \cup PRI()) = \{\$,)\}$

$SIG(E') = SIG(E) = \{\$,)\} \cup SIG(E') = \{\$,)\}$

$SIG(T) = PRI(E') = \{+, \not{*}\} \cup SIG(E) = \{+, \$,)\}$

$SIG(T') = SIG(T) = \{+, \$,)\}$

$SIG(F) = PRI(T') = \{*, \not{*}\} \cup SIG(T) = \{*, +, \$,)\}$

¿CUMPLE LAS CONDICIONES LL(1)?

$E' \to +TE' \mid \lambda$

$T' \to *FT' \mid \lambda$

$F \to (E) \mid id$

EJEMPLO DE CONFLICTO PRI/PRI:

$T' \to \bar{F}\,\bar{T}' \mid \lambda$

$\bar{F} \to (E) \mid id \mid \lambda$

$PRI(F) = \{(, id, \lambda\}$

1.: No puede ser ambigua

2.: No puede ser recursiva por la izquierda

3.- Cuando $A \to \alpha \mid \beta$ debemos comprobar

3.1. No puede haber conflictos PRI/PRI

$E' \to +TE' \mid \lambda \Rightarrow PRI(+TE') \neq PRI(\lambda)$

$+ \neq \lambda$ OK

$T' \to *FT' \mid \lambda \Rightarrow PRI(*FT') \neq PRI(\lambda)$

$* \neq \lambda$ OK

$F \to (E) \mid id \Rightarrow PRI((E)) \neq PRI(id)$

$(\neq id$ OK

3.2 No puede haber multiples alternativas nulas

OK

3.3: No puede haber conflicto PRI/SIG

Quitamos las producciones donde hay alternativas con λ

$E' \to +TE' \mid \lambda \Rightarrow SIG(E') \neq PRI(+TE')$

$\{\$,)\} \neq \{+\}$ OK

$T' \to *FT' \mid \lambda$

$SIG(T') \neq PRI(*FT')$

$\{+, \$,)\} \neq \{*\}$ OK

CONSTRUCCION DE LA TABLA LL(1)

Una vez verificadas las condiciones LL(1) se observa que las cumple, por tanto construimos la tabla LL(1).

NO TERMINAL (Σ_{NT})	SIMBOLOS DE ENTRADA $\Sigma_T \cup \$$					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→λ	E'→λ
T	T→FT'			T→FT'		
T'		T'→λ	T'→*FT'		T'→λ	T'→λ
F	F→id			F→(E)		

1: $E \rightarrow TE'$

vamos a $PRI(T) = \{id, (\}$

2: $E' \rightarrow +TE' \mid \lambda$

$E' \rightarrow +TE'$ $\quad E' \rightarrow \lambda$

$SIG(E') = \{\$,)\}$

3: $T \rightarrow FT'$

$PRI(F) = \{(, id\}$

4: $T' \rightarrow *FT' \mid \lambda$

$PRI(*FT') = \{*\}$ $\quad SIG(T') = \{\$, +,)\}$

5: $F \rightarrow (E) \mid id$

$PRI((E)) = \{(\}$ $\quad PRI(id) = id$