Análisis Sintáctico Descendente

Objetivo: Construir el árbol de análisis desde la raíz hasta las hojas mediante la **Secuencia de Derivación a Izquierdas**

➢ Primeros: $(N \cup \Sigma)^* \to \wp(\Sigma \cup \{\epsilon\})$

$$\texttt{PRI}(\alpha) = \{a \in \Sigma \mid \alpha \stackrel{*}{\Rightarrow} a\beta\} \cup \{\epsilon \mid \alpha \stackrel{*}{\Rightarrow} \epsilon\}; \qquad \alpha, \beta \in (N \cup \Sigma)^*$$

➢ Siguientes: $N \to \wp(\Sigma \cup \{\$\})$

$$\texttt{SIG}(A) = \{a \in \Sigma \mid S \stackrel{*}{\Rightarrow} \alpha A a \beta\} \cup \{\$ \mid S \stackrel{*}{\Rightarrow} \alpha A\}; \qquad \alpha, \beta \in (N \cup \Sigma)^*; A \in N$$

ASD: Condición LL(1)

➢ Condición LL(1):

Una gramática incontextual es **LL(1)** si, para cualquier par de producciones $(A \to \alpha \quad \text{y} \quad A \to \beta)$, se cumple:

$$\boxed{\texttt{PRI}(\alpha \cdot \texttt{SIG}(A)) \cap \texttt{PRI}(\beta \cdot \texttt{SIG}(A)) = \emptyset}$$

➢ Propiedades:

P1.- Si una gramática es **LL(1)**, entonces no es ambigua

P2.- Si una gramática es **LL(1)**, entonces no es recursiva a izquierdas

P3.- Si una gramática es **LL(1)**, entonces no presenta problemas de factorización por la izquierda

Conjunto de Primeros

PRIMEROS : $(N \cup \Sigma)^* \to \wp(\Sigma \cup \{\epsilon\})$

input $\alpha \in (N \cup \Sigma)^*$ **and** $G = (\Sigma,\ N,\ S,\ P)$;
output $\texttt{PRI}(\alpha)$;

$\texttt{PRI}(\alpha) = \emptyset$; $i = 1$; FIN = false;
if ($\alpha == \epsilon$) **then** $\texttt{PRI}(\alpha) = \{\epsilon\}$;
if ($\alpha == x \in \Sigma$) **then** $\texttt{PRI}(\alpha) = \{x\}$;
if ($\alpha == B \in N$) **then for** ($(B \to \beta) \in P$) **do** $\texttt{PRI}(\alpha) = \texttt{PRI}(\alpha) \cup \texttt{PRI}(\beta)$;
if ($\alpha == X_1 X_2 \ldots X_m$ **with** $m > 1$) **then**
  repeat
    $\texttt{PRI}(\alpha) = \texttt{PRI}(\alpha) \cup (\texttt{PRI}(X_i) - \{\epsilon\})$;
    if ($\epsilon \notin \texttt{PRI}(X_i)$) **then** FIN = true; **else** $i = i + 1$;
  until (FIN $\vee$ $(i > m)$);
  if (! FIN $\wedge$ $(i > m)$ **then** $\texttt{PRI}(\alpha) = \texttt{PRI}(\alpha) \cup \{\epsilon\}$;

Conjunto de Siguientes

SIGUIENTES : $N \to \wp(\Sigma \cup \{\$\})$

input $G = (\Sigma,\ N,\ S,\ P)$;
output $\texttt{SIG}(A)$; $\forall A \in N$;

for all $A \in N$ **do** $\texttt{SIG}(A) = \emptyset$;
$\texttt{SIG}(S) = \{\$\}$;
repeat
  for ($(A \in N) \wedge \forall (B \to \alpha A \beta) \in P$)
    $\texttt{SIG}(A) = \texttt{SIG}(A) \cup (\texttt{PRI}(\beta) - \{\epsilon\})$;
    if ($(\beta == \epsilon) \vee (\epsilon \in \texttt{PRI}(\beta))$ **then** $\texttt{SIG}(A) = \texttt{SIG}(A) \cup \texttt{SIG}(B)$;
until no se modifique $\texttt{SIG}(A)$, $\forall A \in N$

Ejemplo de ASD - LL(1) 1/3

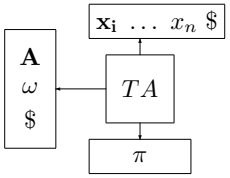
			⇒					
E	::=	E + T		E	::=	T E'	→	PRI(T E' · SIG(E)) ={ a, (}
E	::=	T		E'	::=	+ T E'	→	PRI(+ T E' · SIG(E')) ={ + }
T	::=	T * F		E'	::=	ϵ	→	PRI(SIG(E')) ={), $ }
T	::=	F		T	::=	F T'	→	PRI(F T' · SIG(T)) ={ a, (}
F	::=	(E)		T'	::=	*F T'	→	PRI(* F T' · SIG(T')) ={ * }
F	::=	a		T'	::=	ϵ	→	PRI(SIG(T')) ={ +,), $ }
				F	::=	(E)	→	PRI((E) · SIG(F)) ={ (}
				F	::=	a	→	PRI(a · SIG(F)) ={ a }

SIG(E') = SIG(E) = { $,) }; SIG(T') = SIG(T) = { +, $,) }; SIG(F) = {*, +, $,) };

Construcción de analizadores LL(1)

Dada una gramática $G = (\Sigma,\ N,\ S,\ P)$ que cumple la condición LL(1):

$$S \overset{*}{\Rightarrow} x_1 \dots x_{i-1}\ A\ \omega\ \$ \qquad x_1 \dots x_{i-1}\ x_i\ \dots x_n\ \$$$



Configuración $(x_i \dots x_n\ \$,\ A\ \omega\ \$,\ \pi) \vdash$

$(x_i \dots x_n\ \$,\ \alpha\ \omega\ \$,\ \pi \cdot k)$ **sii** $TA[A, x_i] = k : (A \to \alpha)$

ó

$(x_{i+1} \dots x_n\ \$,\ \omega\ \$,\ \pi)$ **sii** $TA[A, x_i] =$ `consumir`

si $A \in N\ (k : A \to \alpha) \in \mathcal{P}$ `derivar` **sii** $x_i \in$ PRI$(\alpha \cdot$ SIG$(A))$

si $A \in \Sigma$ `consumir` los símbolos (pila y cadena de entrada) **sii** $A = x_i$

Construcción de analizadores LL(1)

Algorithm Construcción de la TA-LL(1)

input $G = (\Sigma,\ N,\ S,\ P)$;

output $TA : ((N \cup \Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})) \to \{k : (A \to \alpha), consumir, aceptar, error\}$;

Inicializar TA con la acción $error$;

for $(k : A \to \alpha) \in P$

  for $a \in$ PRI$(\alpha \cdot$ SIG$(A))$ **do** $TA[A, a] = k : (A \to \alpha)$;

for $a \in T$ **do** $TA[a, a] = consumir$;

$TA[\$, \$] = aceptar$;

Ejemplo de ASD - LL(1) 2/3

			⇒					
E	::=	E + T		E	::=	T E'	→	PRI(T E' · SIG(E)) ={ a, (}
E	::=	T		E'	::=	+ T E'	→	PRI(+ T E' · SIG(E')) ={ + }
T	::=	T * F		E'	::=	ϵ	→	PRI(SIG(E')) ={), $ }
T	::=	F		T	::=	F T'	→	PRI(F T' · SIG(T)) ={ a, (}
F	::=	(E)		T'	::=	*F T'	→	PRI(* F T' · SIG(T')) ={ * }
F	::=	a		T'	::=	ϵ	→	PRI(SIG(T')) ={ +,), $ }
				F	::=	(E)	→	PRI((E) · SIG(F)) ={ (}
				F	::=	a	→	PRI(a · SIG(F)) ={ a }

SIG(E') = SIG(E) = { $,) }; SIG(T') = SIG(T) = { +, $,) }; SIG(F) = {*, +, $,) };

	a	+	*	(	)	$
E	(TE',1)			(TE',1)		
E'		(+TE',2)			(ϵ,3)	(ϵ,3)
T	(FT',4)			(FT',4)		
T'		(ϵ,6)	(*FT',5)		(ϵ,6)	(ϵ,6)
F	(a,8)			((E),7)		
a	sacar					
+		sacar				
*			sacar			
(				sacar		
)					sacar	
$						aceptar

ASD: basado en la tabla LL(1)

Algorithm ASD-LL(1)

input $G = (\Sigma,\ N,\ S,\ P)$; $\omega \in \Sigma^*$

$TA : ((N \cup \Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})) \to \{k : (A \to \alpha), consumir, aceptar, error\}$;

output **if** $\omega \in L(G)$ **then** π **else** $MenError(\cdot)$;

push($\$S$); sym = getsym; $\pi = \epsilon$; ok = FALSE;

repeat

  switch $TA[top, sim]$ **do**

  case $k : (A \to \alpha)$: pop; push(α); $\pi = \pi \cdot k$;

  case $consumir$: pop; sym = getsym;

  case $aceptar$: ok = TRUE;

  case $error$: ok = TRUE; $MenError(\cdot)$;

until ok

Ejemplo de ASD - LL(1) 3/3

E	::=	E + T		E	::=	T E'	→	PRI(T E' · SIG(E))	={ a, (}
E	::=	T		E'	::=	+ T E'	→	PRI(+ T E' · SIG(E'))	={ + }
T	::=	T * F	⇒	E'	::=	ϵ	→	PRI(SIG(E'))	={), $ }
T	::=	F		T	::=	F T'	→	PRI(F T' · SIG(T))	={ a, (}
F	::=	(E)		T'	::=	*F T'	→	PRI(* F T' · SIG(T'))	={ * }
F	::=	a		T'	::=	ϵ	→	PRI(SIG(T'))	={ +,), $ }
				F	::=	(E)	→	PRI((E) · SIG(F))	={ (}
				F	::=	a	→	PRI(a · SIG(F))	={ a }

SIG(E') = SIG(E) = { $,) }; SIG(T') = SIG(T) = { +, $,) }; SIG(F) = {*, +, $,) };

	a	+	*	(	)	$
E	(TE',1)			(TE',1)		
E'		(+TE',2)			(ϵ,3)	(ϵ,3)
T	(FT',4)			(FT',4)		
T'		(ϵ,6)	(*FT',5)		(ϵ,6)	(ϵ,6)
F	(a,8)			((E),7)		
a	sacar					
+		sacar				
*			sacar			
(				sacar		
)					sacar	
$						aceptar

$(a * a\$,\ E\$,\ \epsilon)$ ⊢ $(a * a\$,\ TE'\$,\ 1)$
⊢ $(a * a\$,\ FT'E'\$,\ 14)$
⊢ $(a * a\$,\ aT'E'\$,\ 148)$
⊢ $(*a\$,\ T'E'\$,\ 148)$
⊢ $(*a\$,\ *FT'E'\$,\ 1485)$
⊢ $(a\$,\ FT'E'\$,\ 1485)$
⊢ $(a\$,\ aT'E'\$,\ 14858)$
⊢ ($, T'E'$, 14858)
⊢ ($, E'$, 148586)
⊢ ($, $, 1485863)

Modificación de gramáticas no LL(1)

- Recursividad a izquierdas (directa)

  $A \to A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

  ⇨ Eliminación de la recursividad a izquierdas

  $A \to \beta_1 A' \mid \ldots \mid \beta_m A'$

  $A' \to \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \epsilon$

- Factorización por la izquierda

  $A \to \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_m$

  ⇨ Eliminación de la factorización por la izquierda

  $A \to \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m$

  $A' \to \beta_1 \mid \ldots \mid \beta_n$