Lenguajes formales y autómatas

Analizador sintáctico

- Convierte la secuencia de tokens de entrada en árboles de derivación
- Si el analizador no es capaz de generar un árbol, entonces la cadena no pertenece al lenguaje
- Se basan en autómatas de pila y gramáticas libres de contexto
- Hay distintas técnicas para obtener el AP; no todas ellas son capaces de reconocer todas las gramáticas

Técnica LL

- L -> Left to right. Los tokens se leen de izquierda a derecha
- L -> Left most. Las derivaciones se hacen lo más a la izquierda posible
- Es una técnica descendente: trata de armar el árbol desde la raíz hacia las hojas
- Predictivo: predice las reglas necesarias para llegar a las hojas
- Estando en un NT, tengo que predecir por cual derivación expandir el árbol. Cuando hay más de 1 posibilidad tengo que **ver** que token sigue en la entrada. Este token se denomina **lookahead**
- En todo momento con **k** tokens de lookahead debo poder determinar unívocamente que regla derivar: **LL(k)**

Ejemplo de gramática:

S ::= A B

A ::= a A

A ::= c

B ::= b B

B ::= d

Algoritmo

1. Escribir la gramática
2. Por cada regla escribir el **conjunto first**: Dada una regla, el conjunto First representa los posibles tokens con los que comienza esa regla.
3. Por cada NT escribir el **conjunto follow**: Dado un NT, el conjunto follow representa los posibles tokens que deberían a aparecer en la cadena una vez que el NT se haya reducido.
4. Por cada regla escribir el **conjunto select**: Idem al conjunto first pero sin lambdas
5. Si los conjuntos selects son disyuntos para cada NT, entonces la gramática es LL(k)

Cada elemento de en los conjuntos es una secuencia de tokens de largo **k**

Conjunto First (Fi). Reglas

1. $Fi(a) = \{a\};\ Fi(\lambda) = \{\lambda\}$

2.1. $Si\ S \to a\delta \Rightarrow a \in Fi(S)$

2.2. $Si\ S \to \delta\beta \Rightarrow Fi(\delta) \in Fi(S)$

3. $Si\ S \to \delta\beta\gamma \wedge \delta \to^* \lambda \Rightarrow\ reemplazar\ \lambda\ en\ Fi(S)\ por\ Fi(\beta)$

4. $Si\ S \to \delta \wedge \delta \to^* \lambda \Rightarrow lambda \in Fi(S)$

Conjunto Follow (Fo). Reglas

1. $Si\ S\ es\ distinguido \Rightarrow\ \$ \in Fo(S)$
2. $Si\ S \to \delta\alpha\beta \Rightarrow \{s | s \in \{Fi(\alpha) - \lambda\}\} \in Fo(\delta)\ ;\ \ Si\ \alpha \to^* \lambda \Rightarrow \{s | s \in \{Fi(\beta) - \lambda\}\} \in Fo(\delta)$
3. $Si\ S \to \delta\alpha \wedge \alpha \to^* \lambda \Rightarrow\ Fo(S)\ \in Fo(\delta)$

Conjunto Select (Se).

- Se calculan reemplazando en los conjuntos Fi los lambdas por los Fo del NT.

Ejemplo LL.

Regla	First	Follow	Select
S ::= X Y Z			
X ::= a X ::= b X ::= λ			
Y ::= a Y ::= d Y ::= λ			
Z ::= e Z ::= f Z ::= λ			

Ejemplo LL.

Regla	First	Follow	Select
S ::= X Y Z	{a, b, d, e, f, λ }	{$}	{a, b, d, e, f, $}
X ::= a X ::= b X ::= λ	{a} {b} { λ }	{a, d, e, f, $}	{a} {b} {a, d, e, f, $}
Y ::= a Y ::= d Y ::= λ	{a} {d} { λ }	{e, f, $}	{a} {d} {e, f, $}
Z ::= e Z ::= f Z ::= λ	{e} {f} { λ }	{$}	{e} {f} {$}

Problemas comunes

Factor común		Recursión a izquierda	
S ::= a B1 S ::= a B2 S ::= a B3 S ::= C S ::= D S ::= E	S ::= a S' S ::= C S ::= D S ::= E S'::= B1 S'::= B2 S'::= B3	S ::= S C S ::= S D S ::= S E S ::= F S ::= G S ::= H	S ::= F S' S ::= G S' S ::= H S' S'::= C S' S'::= D S' S'::= E S' S'::= λ

Ejemplo

```
E ::= E + T
E ::= E - T
E ::= T
T ::= T * F
T ::= T / F
T ::= F
F ::= num
F ::= (E)
```

Ejemplo

Original	Corregida	Selects
E ::= E + T	E ::= T E'	{num, (}
E ::= E - T	E'::= + T E'	{+}
E ::= T	E'::= - T E'	{-}
T ::= T * F	E'::= λ	{$,)}
T ::= T / F	T ::= F T'	{num, (}
T ::= F	T'::= * F T'	{*}
F ::= num	T'::= / F T'	{/}
F ::= (E)	T'::= λ	{+, -, $,)}
	F ::= num	{num}
	F ::= (E)	{(}

Implementación recursiva

1. Por cada NT hacer una rutina donde:
   - Por cada regla del NT hacer un if para que de acuerdo al lookahead elija que regla usar.
   - Por cada regla hacer llamadas para los NT e if + consumo de tokens para los T
2. El programa principal consume un token, llama a la rutina del distinguido y si el lookahead es $ la cadena pertenece al lenguaje.

```
def E():
    T()
    E1()

def E1():
    global look
    if look in ['+']:  # + T E'
        look = yylex()
        T()
        E1()
    elif look in ['-']:  # - T E'
        look = yylex()
        T()
        E1()
    elif look in ['$', ')']:  # lambda                                    .
        return
    else:
        error()
```

Implementación con tabla

	num	+	-	*	/	(	)	$
E	T E'					T E'		
E'		+T E'	-T E'				[]	[]
T	F T'					F T'		
T'		[]	[]	*F T'	/ F T'		[]	[]
F	num					(E)		

```
stack = ['E']
look = yylex()

while stack:
    s = stack.pop()
    if s in NT:
        l = table[(s, look)]
        l = l[::-1]
        stack.extend(l)
    elif s == look:
        look = yylex()
    else:
        print 'Error'

if look == '$':
    print 'ok'
else:
    print 'Error'
```

Ventajas del LL

- Asegura una derivación left most determinística
- Si una gramática es LL no hay ambigüedad
- Tiempo lineal según la longitud de la cadena

Desventajas del LL

- Abarca un pequeño conjunto de gramáticas
- A veces hay que desdibujar la gramática para aplicarlo

Bibliografía y enlaces útiles.

- Aho Alfred y Ullman Jeffrey - Compiladores, principios, técnicas y herramientas - PEARSON EDUCACION