Introducción a la Compilación: Parsing Descendente

Construyendo un parser descendente

Recordemos la gramática de expresiones "natural".

```
E -> E + E
   | E - E
   | E * E
   | E / E
   | ( E )
   | i
```

> ¿Qué problema fundamental tiene esta gramática?

. . .

> **No modela la precedencia de los operadores.**

Precedencia de operadores

Resolvemos la precedencia poniendo un no-terminal por cada "nivel".

```
E -> T + E
   | T - E
   | T
T -> F * T
   | F / T
   | F
F -> ( E )
   | i
```

> ¿Y ahora qué problema tiene?

. . .

> **Asocia "incorrectamente" hacia la derecha.**

Asociatividad

¿Por qué no cambiar entonces las producciones para asociar "correctamente"?

```
E -> E + T
   | E - T
   | T
T -> T * F
   | T / F
   | F
F -> ( E )
   | i
```

> ¿Esta sí?

. . .

> **No se puede *parsear* recursivamente.**

Eliminando la recursión izquieda

> **Definición:** Una gramática libre del contexto $G =< S, N, T, P >$ se dice recursiva izquierda si y solo si $S \rightarrow^* Sw$ (donde $w \in \{N \cup T\}^*$ es una forma oracional).

Ejemplo (recursión directa):

```
S -> Sa | b
```

¿Se puede arreglar?

. . .

```
S -> bX
X -> aX | epsilon
```

¿Cómo queda la gramática de expresiones?

```
E -> T X
X -> + T X
   | - T X
   | epsilon
T -> F Y
Y -> * F Y
   | / F Y
   | epsilon
F -> ( E )
   | i
```

> ¿Sigue mantienendo la asociatividad "correcta"?

Otra forma de llegar a lo mismo

```
E -> T + E
   | T - E
   | T
T -> F * T
   | F / T
   | F
F -> ( E )
   | i
```

> ¿Qué *otro* problema tiene al parsear?

. . .

> **Ambigüedad para escoger la siguiente producción.**

Factorización a la izquierda

Definición: Diremos que una gramática $G =< S, N, T, P >$ está *factorizada a la izquierda* si para todo par de producciones $A \to \theta_i w_i$ y $A \to \theta_j w_j$ se cumple $\theta_i \neq \theta_j$ ($\theta_i, \theta_j \in N \cup T$).

```
S -> cA | cB
A -> aA | a
B -> bB | b
```

Se puede arreglar, ¿verdad?

. . .

```
S -> cX
X -> A  | B
A -> aA | a
B -> bB | b
```

Volviendo a la gramática de expresiones

```
E -> T X
X -> + E
   | - E
   | epsilon
T -> F Y
Y -> * T
   | / T
   | epsilon
F -> ( E )
   | i
```

> Y luego es solo un pasito para llegar a la misma gramática...

Vamos a parsear

Por vagancia, usemos una gramática más sencilla...

```
E -> T X
X -> + E | epsilon
T -> i Y | ( E )
Y -> * T | epsilon
```

Y veamos cómo queda el *parse* de:

```
w = i * ( i + i )
```

Parsing final

```
E -> T X
  -> i Y X
  -> i * T X
  -> i * ( E ) X
  -> i * ( T X ) X
  -> i * ( i Y X ) X
  -> i * ( i X ) X
  -> i * ( i + E ) X
  -> i * ( i + T X ) X
  -> i * ( i + i Y X ) X
  -> i * ( i + i X ) X
  -> i * ( i + i ) X
  -> i * ( i + i )
```

¿Qué hemos hecho en cada paso?

- Saber si alguna de las producciones de X puede derivar en una forma oracional cuyo primer símbolo sea el terminal que toca generar.
- Si $X \rightarrow \epsilon$, saber si esta derivación puede potencialmente redundar en que "lo que sea que venga detrás" de X genere el terminal que toca.

> Tratemos de definir formalmente estas ideas.

First y Follow

> **Definición**: Sea $G =< S, N, T, P >$ una gramática libre del contexto, $\alpha \in \{N \cup T\}^*$ una forma oracional, y $x \in T$ un terminal. Decimos que $x \in First(\alpha)$ si y solo si $\alpha \rightarrow^* x\beta$ (donde $\beta \in \{N \cup T\}^*$ es otra forma oracional).

. . .

> **Definición**: Sea $G =< S, N, T, P >$ una gramática libre del contexto, $X \in N$ un no-terminal, y $x \in T$ un terminal. Decimos que $x \in Follow(X)$ si y solo si $S \rightarrow^* \alpha X x \beta$ (donde $\alpha, \beta \in \{N \cup T\}^*$ son formas oracionales cualesquiera).

¿Para qué nos sirve esto?

Intuitivamente, los conjuntos `First` y `Follow` nos permiten decidir qué producción escoger. **¿Cómo?**

. . .

1. $X \to W$ es una producción posible si y solo si el próximo token `t` pertenece al $First(W)$.
2. $X \to \epsilon$ es una producción posible si y solo si el próximo token `t` pertenece al $Follow(X)$.

. . .

Informalmente, llamaremos a una gramática, `LL(1)` si estos conjuntos nos permiten escoger **siempre**.

LL(1): *Left-to-right* ***L****eftmost-derivation Lookahead* ***1***

Calculando el `First`

- Si $X \to W1|W2|\dots|Wn$ entonces por definición, $First(X) = \cup First(W_i)$.
- Si $X \to^* \epsilon$ entonces $\epsilon \in First(X)$.
- Si $W = xZ$ donde x es un terminal, entonces trivialmente $First(W) = x$.
- Si $W = YZ$ donde Y es un no-terminal y Z una forma oracional, entonces $First(Y) \subseteq First(W)$.
- Si $W = YZ$ y $Y \to^* \epsilon$ entonces $First(Z) \subseteq First(W)$.

> ¡A calcular todos los `First`!

Calculando el `Follow`

- $ pertenece al $Follow(S)$.
- Por definición ϵ nunca pertenece al $Follow(X)$ para todo X.
- Si $X \to WAZ$ siendo W y Z formas oracionales, y A un no-terminal cualquiera, entonces $First(Z) - \epsilon \subseteq Follow(A)$.
- Si $X \to WAZ$ y $Z \to^* \epsilon$ (o igualmente ϵ está en el $First(Z)$), entonces $Follow(X) \subseteq Follow(A)$.

> ¡A calcular todos los `Follow`!

Construyendo la tabla LL(1)

1. Si $X \to W$ y $t \in First(W)$ entonces $T[X, t] = X \to W$.
2. Si $X \to \epsilon$ y $t \in Follow(X)$ entonces $T[X, t] = X \to \epsilon$.

	i	+	*	(	)	$
E	T X			T X		
T	i Y			(E)		
X		+ E			ϵ	
Y		ϵ		* T		ϵ

Finalmente...

> **Definición:** Sea $G =< S, N, T, P >$ una gramática libre del contexto. G es LL(1) si y solo si para todo no-terminal $X \in N$, tal que $X \rightarrow W_1|W_2|\ldots|W_n$ se cumple que:
>
> - $First(Wi) \bigcap First(Wj) = \emptyset \ \forall i \neq j$
> - $\epsilon \in First(Wi) => First(Wj) \bigcap Follow(X) = \emptyset \ \forall j \neq i$

Algunas conclusiones parciales

> - Este es el primer tipo de *parser* **de verdad**, usado en la práctica, que se puede implementar a mano.
> - Pero... al tener un algoritmo de construcción de *parser*, ya no hay que hacerlo a mano.
> - Aunque la mayoría de las gramáticas interesantes no son LL(1), con un poco de esfuerzo casi siempre se pueden transformar.
> - De todas formas, las gramáticas LL(1) son bastante feas...
> - Si una gramática es ambigua, no es LL(1) (y el algoritmo dice dónde).
> - Si una gramática no es LL(1), no tiene por qué ser ambigua (hay *parsers* más potentes).

. . .

> **Pregunta interesante:** ¿Existen lenguajes no ambiguos para los que no existe ninguna gramática LL(1)?