

## Traducción dirigida por sintaxis

El objetivo es especificar la traducción de una construcción de un lenguaje de origen en términos de atributos de sus componentes sintácticos. La idea básica es usar las producciones para especificar un procedimiento (típicamente recursivo) para la traducción. Por ejemplo, considere la producción

```
stmt-list → stmt-list ; stmt
```

Para procesar la `stmt-list` izquierda,

1. Se hace una llamada recursiva para procesar la `stmt-list` derecha (que es más pequeña). Esto, por ejemplo, generará código para todas las declaraciones en la `stmt-list` derecha.
2. Se invoca al procedimiento para `stmt`, generando código para `stmt`.
3. Se procesa la `stmt-list` izquierda combinando los resultados de los dos primeros pasos, así como lo que se necesita para el punto y coma (un terminal, por lo que no delegamos más sus acciones). En este caso, probablemente concatenamos el código para la `stmt-list` derecha y para `stmt`.

Para evitar tener que decir la `stmt-list` derecha y la `stmt-list` izquierda, escribimos la producción como

```
stmt-list → stmt-list1 ; stmt
```

donde el subíndice se usa para distinguir las dos instancias de `stmt-list`.

### Notación post-fija (un ejemplo)

Esta notación se llama **postfija** porque la regla es un operador **después** de los operandos. No se necesitan paréntesis. La notación que usamos normalmente se llama **infija** porque las reglas son un operador entre operandos. Si se comienza con una expresión de infijo, el siguiente algoritmo dará la expresión de sufijo equivalente.

1. Las variables y constantes no se cambian.
2.  $E \text{ op } F$  se convierte en  $E' F' \text{ op}$ , donde  $E'$  y  $F'$  son el post-fijo de  $E$  y  $F$  respectivamente.
3.  $(E)$  se convierte en  $E'$ , donde  $E'$  es el post-fijo de  $E$ .

Una pregunta es, por ejemplo en  $1 + 2 - 3$ , ¿qué son E, F y op? ¿E = 1 + 2, F = 3 y op = -? ¿O E = 1, F = 2 - 3 y op = +? Este es el tema de la precedencia y la asociatividad mencionado anteriormente. Para simplificar la presente discusión, comenzaremos con expresiones infijas entre paréntesis.

**Ejemplo:**  $1 + 2 / 3 - 4 * 5$

1. Comenzando con  $1 + 2 / 3 - 4 * 5$
2. Paréntesis (usando precedencia estándar) para obtener  $(1+(2/3))-(4*5)$
3. Aplique las reglas anteriores para calcular  $P\{(1+(2/3))-(4*5)\}$ , donde P{X} significa convertir la expresión infija X en postfijo.
  - A.  $P\{(1+(2/3))-(4*5)\}$
  - B.  $P\{(1+(2/3))\} P\{(4*5)\} -$
  - C.  $P\{1+(2/3)\} P\{4*5\} -$
  - D.  $P\{1\} P\{2/3\} + P\{4\} P\{5\} * -$
  - E.  $1 P\{2\} P\{3\} / + 4 5 * -$
  - F.  $1 2 3 / + 4 5 * -$

**Ejemplo:** Ahora hacer  $(1+2)/3-4*5$

1. Paréntesis para obtener  $((1+2)/3)-(4*5)$
2. Calcular  $P\{((1+2)/3)-(4*5)\}$ 
  - A.  $P\{((1+2)/3) P\{(4*5)\} -$
  - B.  $P\{(1+2)/3\} P\{4*5\} -$
  - C.  $P\{(1+2)\} P\{3\} / P\{4\} P\{5\} * -$
  - D.  $P\{1+2\} 3 / 4 5 * -$
  - E.  $P\{1\} P\{2\} + 3 / 4 5 * -$
  - F.  $1 2 + 3 / 4 5 * -$

## Atributos sintetizados

Queremos decorar los árboles de análisis que construimos con anotaciones que den el valor de ciertos *atributos* del nodo correspondiente del árbol.

En un traductor de práctica, se podría usar como entrada un lenguaje similar a C/Java/Python etc y podría existir un atributo de `codigo` para muchos nodos con la propiedad de que `codigo` contendría el código intermedio que resulta de compilar el programa correspondiente a las hojas del subárbol con este nodo como raíz. En particular `raizDelArbol.codigo` contendrá la compilación de todo el programa similar a C/Java/Python.

Sin embargo, por ahora el objetivo es más modesto. Se hará el ejercicio de traducir infijo a postfijo usando la misma gramática de infijo descrita antes. Para mayor comodidad, la gramática se repite abajo. Los nombres de los no terminales corresponden a la terminología aritmética estándar donde se multiplican y dividen factores para obtener términos, que a su vez se suman y restan para formar expresiones.

```

expr   → expr + term | expr - term | term
term   → term * factor | term / factor | factor
factor → digit | ( expr )
digit  → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

Esta gramática admite paréntesis, aunque el ejemplo  $1 + 2 / 3 - 4 * 5$  no los usa.

El atributo que asociaremos con los nodos es la forma post-fija de la cadena en las hojas debajo del nodo. En particular, el valor de este atributo en la raíz es la forma post-fija de la expresión completa.

### Definiciones Dirigidas por Sintaxis (SDD)

**Definición:** Una *definición dirigida por sintaxis* es una gramática junto con *reglas semánticas* asociadas con las producciones. Estas reglas se utilizan para calcular los valores de los atributos. Un árbol de análisis aumentado con los valores de atributo en cada nodo se denomina *árbol de análisis anotado*.

Para el enfoque de abajo hacia arriba que ilustraré ahora, anotamos un nodo **después de** haber anotado sus hijos. Por lo tanto, los valores de atributo en un nodo pueden depender de los valores de los atributos en los hijos del nodo, pero no de los atributos en el padre del nodo. A estos atributos los llamamos sintetizados de abajo hacia arriba, ya que se forman sintetizando los atributos de los hijos.

Más adelante, cuando estudiemos también las anotaciones de arriba hacia abajo, presentaremos atributos heredados que se transmiten de padres a hijos.

Especificamos cómo sintetizar atributos dando las reglas semánticas junto con la gramática. Es decir, damos la **definición dirigida por sintaxis**.

Producción	Regla semántica
$\text{expr} \rightarrow \text{expr}_1 + \text{term}$	$\text{expr.t} := \text{expr}_1.\text{t} \parallel \text{term.t} \parallel '+'$
$\text{expr} \rightarrow \text{expr}_1 - \text{term}$	$\text{expr.t} := \text{expr}_1.\text{t} \parallel \text{term.t} \parallel '-'$
$\text{expr} \rightarrow \text{term}$	$\text{expr.t} := \text{term.t}$
$\text{term} \rightarrow \text{term}_1 * \text{factor}$	$\text{term.t} := \text{term}_1.\text{t} \parallel \text{factor.t} \parallel '*'$

term → term <sub>1</sub> / factor	term.t := term <sub>1</sub> .t    factor.t    '/'
term → factor	term.t := factor.t
factor → digit	factor.t := digit.t
factor → ( expr )	factor.t := expr.t
digit → 0	digit.t := '0'
digit → 1	digit.t := '1'
digit → 2	digit.t := '2'
digit → 3	digit.t := '3'
digit → 4	digit.t := '4'
digit → 5	digit.t := '5'
digit → 6	digit.t := '6'
digit → 7	digit.t := '7'
digit → 8	digit.t := '8'
digit → 9	digit.t := '9'

## DDS para traductor de infijo a posfijo

Aplicando estas reglas de abajo hacia arriba (comenzando con las producciones geográficamente más bajas, es decir, las líneas más bajas del árbol) se obtiene el gráfico anotado que se muestra. Las anotaciones se ilustran en verde.

