Gramáticas y Lenguajes de Contexto Libre

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Backus Naur Form

Las gramáticas de contexto libre son parte fundamental de la tecnología de los lenguajes de programación modernos. Suelen describirse mediante BNF (por *Backus-Naur Form*), que usa `<no-terminal>` para indicar no-terminales (ayuda a explicar lo que representa), los terminales se indican como `"terminal"`, el símbolo `::=` es lo que escribimos $\rightarrow$ y se usa `|` para indicar alternativas. Comúnmente se usan extensiones de BNF, conocidas como EBNF (*Extended BNF*), que agregan construcciones como usar `{ ... }` para agrupar y cuantificadores como $*$ (cero o más de lo anterior), + (una o más de lo anterior) y ? (lo anterior es opcional).

Ejemplo de BNF

```
<command-line> ::= <list>
                 |  <list> ";"
                 |  <list> "&"

<list> ::=  <conditional>
          |   <list> ";" <conditional>
          |   <list> "&" <conditional>

<conditional> ::=  <pipeline>
                |   <conditional> "&&" <pipeline>
                |   <conditional> "||" <pipeline>
```

Ejemplo de BNF (cont.)

```
<pipeline> ::=  <command>
             |  <pipeline> "|" <command>

<command>  ::=  <word>
             |  <redirection>
             |  <command> <word>
             |  <command> <redirection>

<redirection>  ::=  <redirectionop> filename
<redirectionop>  ::=  "<"  |  ">"  |  "2>"
```

Ejemplo de BNF (cont.)

Algunos ejemplos:

```
gcc -O2 -Wall -o pgm abc.c xyz.c -lm

grep '#include␣"' *.c | wc -l

./pgm 2> /tmp/errors | less

pgm file | sort | uniq | more
```

Backus Naur Form

Esta gramática es incompleta, no explica qué es `<word>` ni `<filename>`.
Es común trabajar con gramáticas incompletas como ésta, completando con explicaciones (o formalismos) adicionales.

Motivación

Si en una gramática de contexto libre (comúnmente abreviada CFG, por *Context Free Grammar* en inglés; los respectivos lenguajes son CFL, *Context Free Language*) en una derivación aparecen dos o más no-terminales, podemos elegir cuál de ellos expandir. Al ser de contexto libre la gramática, cuál elegimos no afecta las posibilidades de los demás. Pero formalmente las derivaciones son diferentes, a pesar que intuitivamente «son la misma».

Motivación

Tenemos varias salidas a este entuerto:

- Definir un orden en que se consideran los no-terminales: *derivación de extrema izquierda* o *canónica*, siempre se reemplaza primero el no-terminal más a la izquierda, *derivación de extrema derecha*, siempre se reemplaza primero el no-terminal más a la derecha.
- Simplemente obviar el orden, representar la derivación mediante un *árbol de derivación* que para cada no-terminal tiene de hijos el lado derecho de la producción a que se expande.

Ejemplo

Consideremos nuestra gramática regalona:

$$E \to E + T$$
$$E \to T$$
$$T \to T * F$$
$$T \to F$$
$$F \to (E)$$
$$F \to a$$

Ejemplo

Derivación cualquiera de $a * (a + a)$

$$
\begin{aligned}
\underline{E} &\Rightarrow \underline{T} && E \to T \\
&\Rightarrow T * \underline{F} && T \to T * F \\
&\Rightarrow T * (\underline{E}) && F \to (E) \\
&\Rightarrow T * (\underline{E} + T) && E \to E + T \\
&\Rightarrow T * (T + \underline{T}) && E \to T \\
&\Rightarrow T * (T + \underline{F}) && T \to F \\
&\Rightarrow \underline{T} * (T + a) && F \to a \\
&\Rightarrow F * (\underline{T} + a) && T \to F \\
&\Rightarrow F * (\underline{F} + a) && T \to F \\
&\Rightarrow \underline{F} * (a + a) && F \to a \\
&\Rightarrow a * (a + a) && F \to a
\end{aligned}
$$

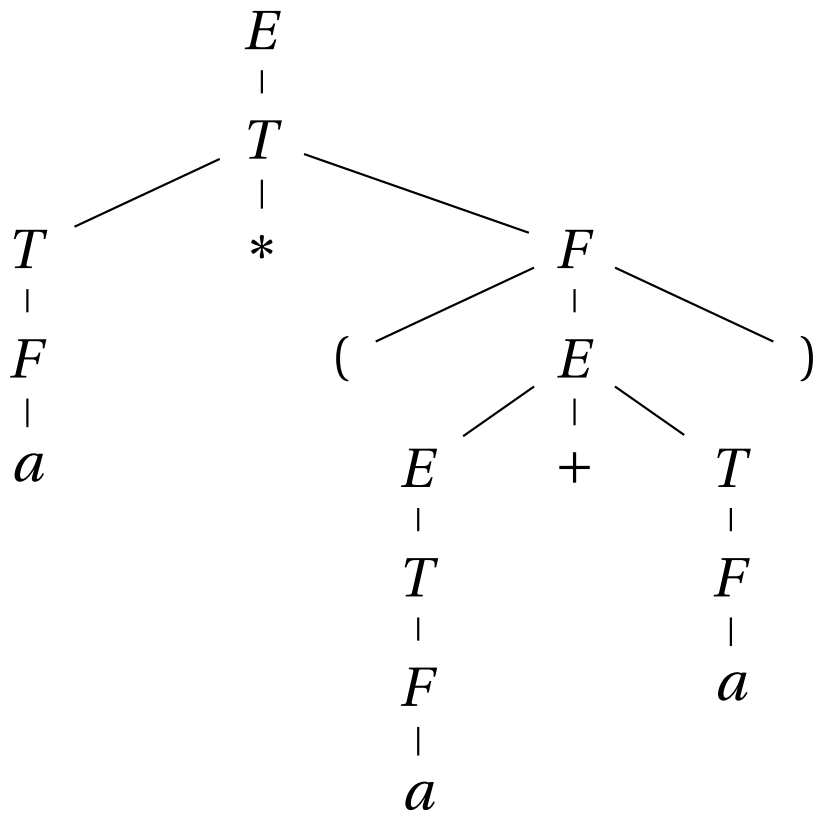
Ejemplo

Derivación canónica de $a * (a + a)$

$$
\begin{aligned}
E &\Rightarrow T && E \to T \\
&\Rightarrow T * F && T \to T * F \\
&\Rightarrow F * F && T \to F \\
&\Rightarrow a * F && F \to a \\
&\Rightarrow a * (E) && F \to (E) \\
&\Rightarrow a * (E + T) && E \to E + T \\
&\Rightarrow a * (T + T) && E \to T \\
&\Rightarrow a * (F + T) && T \to F \\
&\Rightarrow a * (a + T) && F \to a \\
&\Rightarrow a * (a + F) && T \to F \\
&\Rightarrow a * (a + a) && F \to a
\end{aligned}
$$

Ejemplo

Árbol de derivación de $a * (a + a)$



Ejemplo

Es claro que a un árbol de derivación corresponde una derivación de extrema izquierda y una de extrema derecha.
El árbol de derivación muestra gráficamente la estructura impuesta sobre la palabra por la gramática. Interesa que podemos calcular el valor de la expresión mediante un recorrido en postorden del árbol de derivación. Esta observación es lo que hace que árboles de derivación (y gramáticas de contexto libre) sean ubicuos en la descripción de lenguajes de programación. Se considera el «significado» de la palabra asociado al árbol de derivación respectivo.

Otra gramática

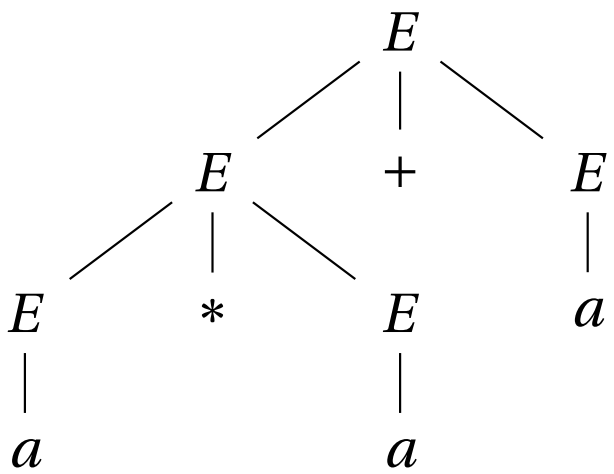
Obviamente hay otras gramáticas que generan el mismo lenguaje que nuestra gramática regalona.
Por ejemplo:

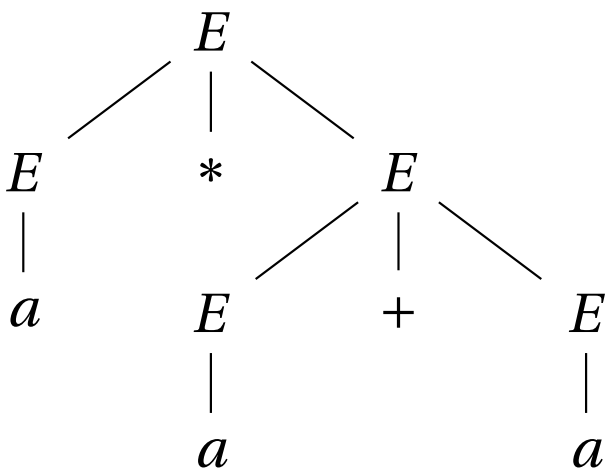
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

Genera el mismo lenguaje ya que tenemos $E \Rightarrow T \Rightarrow F$, que nos dice que todo E (o T) podemos reemplazarlo por F sin cambiar el lenguaje generado (esencialmente es resumir ramas del árbol de derivación). Hacer esto sistemáticamente da la gramática de arriba, pero con F en vez de E.

Ambigüedad

Claro que no impone la misma estructura anterior. Peor aún, hay dos árboles de derivación distintos para, por ejemplo, $a * a + a$:





No hay un significado único, la gramática es *ambigua*.
(Calcular el valor vía recorrido en postorden da $a^2 + a$ y $2a^2$, respectivamente.)

Manipular gramáticas

Es claro que si en una gramática de contexto libre hay una producción $A \to \alpha$, donde A no aparece en α, podemos «simular» esa producción creando nuevas producciones en cuyos lados derechos reemplazamos A por α y eliminando la original (si A aparece varias veces debemos reemplazar todos los subconjuntos de sus apariciones, claro está). Esto no puede introducir ambigüedad (si la gramática original no era ambigua, la gramática modificada —que esencialmente se «salta» esa producción— no puede dar lugar a árboles de derivación alternativos).

Producciones ε

Vimos que en las gramáticas regulares podíamos aceptar producciones nulas (de la forma $A \to \varepsilon$) sin afectar mayormente los lenguajes generados (solo agregamos la posibilidad de generar ε). Demostraremos lo propio para gramáticas de contexto libre.

No-terminales que dan ε

La idea es aplicar la observación precedente. Para ello debemos identificar todos los no-terminales A tales que $A \Rightarrow^* \varepsilon$ (en inglés les llaman *nullable symbols*, frase imposible de traducir limpiamente al castellano). Luego creamos nuevas producciones con lados derechos que simplemente omiten todos los subconjuntos de los no-terminales del caso, y finalmente eliminando todas las producciones nulas.

Si $S \Rightarrow^* \varepsilon$, basta reponer solo la producción $S \to \varepsilon$ para generar el lenguaje original (ε incluido).

No-terminales que dan ε

Determinar el conjunto de los no-terminales que derivan ε es bastante simple:

- Si hay una producción $A \to \varepsilon$, el no-terminal A pertenece al conjunto.
- Si $A \to \alpha$ es una producción, y α está formado solo por símbolos del conjunto, A pertenece al conjunto.

Partiendo con los no-terminales que dan ε directamente (primera regla), agregamos sistemáticamente los que la segunda regla incluye hasta que no hayan cambios.

Ejemplo

Consideremos la gramática:

$$S \to aAbB \mid bBaA \mid cAc \mid BCB$$
$$A \to bA \mid \varepsilon$$
$$B \to aB \mid \varepsilon$$
$$C \to AB \mid cc$$

Dan ε:

$$\{A, B, C, S\}$$

Ejemplo

Reemplazando en la gramática como se describió antes:

$$S \to aAbB \mid bBaA \mid cAc \mid BCB$$
$$S \to abB \mid aAb \mid ab$$
$$S \to baA \mid bBa \mid ba$$
$$S \to cc$$
$$S \to CB \mid BB \mid BC \mid B \mid C \mid \varepsilon$$
$$A \to bA$$
$$A \to b$$
$$B \to aB$$
$$B \to a$$
$$C \to AB \mid cc$$
$$C \to B \mid A \mid \varepsilon$$

Ejemplo

Eliminando producciones nulas:

$$S \to aAbB \mid bBaA \mid cAc \mid BCB$$

$$S \to abB \mid aAb \mid ab$$

$$S \to baA \mid bBa \mid ba$$

$$S \to cc$$

$$S \to CB \mid BB \mid BC \mid B \mid C$$

$$A \to bA$$

$$A \to b$$

$$B \to aB$$

$$B \to a$$

$$C \to AB \mid cc$$

$$C \to B \mid A$$

Ejemplo

Eliminamos las producciones nulas y obtenemos una gramática de contexto libre.

Para recuperar el lenguaje original, dado que $S \Rightarrow^* \varepsilon$, agregamos:

$$S \to \varepsilon$$

(claro que la gramática resultante formalmente no es de contexto libre).

Producciones nulas

Es claro que las gramáticas pueden crecer bastante al eliminar producciones nulas.
Permitir producciones nulas es una abreviación muy bienvenida en la definición de lenguajes de programación: es común tener construcciones opcionales.

Por la construcción precedente informalmente se llaman de contexto libre gramáticas todas cuyas producciones tienen la forma $A \to \alpha$ con $A \in N$ y $\alpha \in (N \cup \Sigma)^*$.

Símbolos inútiles

Un símbolo que no aparece en ninguna derivación de una palabra en el lenguaje claramente es inútil.

Sea la gramática $G = (N, \Sigma, P, S)$.
Un símbolo X del vocabulario de G se llama *generante* si $X \Rightarrow^* \alpha$, con $\alpha \in \Sigma^*$.
Un símbolo X del vocabulario de G se llama *alcanzable* si hay una derivación $S \Rightarrow^* \alpha X \beta$.

Buscamos eliminar símbolos no generantes e inalcanzables y las producciones en que participan.

Algoritmos para determinar generantes y alcanzables son bastante obvios.

Ejemplo

Consideremos la gramática de contexto libre siguiente:

$$S \to AB \mid a$$
$$A \to b$$
$$B \to Bb \mid BA$$
$$C \to cc$$

Son generantes:

$$\{a, b, c, S, A, C\}$$

Son alcanzables:

$$\{S, a, A, B, b\} = \{a, b, S, A, B\}$$

Orden de las operaciones

Si primero eliminamos los símbolos no generantes:

$$S \to a$$
$$A \to b$$
$$C \to cc$$

En esta gramática son alcanzables $\{a, S\}$.

Eliminando ahora los no alcanzables:

$$S \to a$$

Orden de las operaciones

Si primero eliminamos los símbolos no alcanzables:

$$S \to AB \mid a$$
$$A \to b$$
$$B \to Bb \mid BA$$

Son generantes en la gramática que resulta $\{a, b, S, A\}$.

Eliminando luego los símbolos no generantes de la gramática resultante:

$$S \to a$$
$$A \to b$$

Orden de las operaciones

O sea, debemos siempre primero eliminar no generantes y luego no alcanzables.

Demostrar que efectuar las operaciones en este orden elimina los símbolos inútiles queda de ejercicio.

Relevancia práctica

Si definimos una gramática, digamos para un lenguaje de programación, presumiblemente buscamos no incorporar símbolos o producciones inútiles. El uso práctico es determinar errores de diseño de la gramática.

Resumen

- Describimos el formalismo de Backus-Naur, usado para definir lenguajes de programación.
- Discutimos operaciones generales que dan gramáticas equivalentes (que generan el mismo lenguaje).
- Usamos las anteriores para eliminar símbolos inútiles.
- Introdujimos las derivaciones de extrema izquierda y de extrema derecha como representantes de las «derivaciones equivalentes».
- Definimos árboles de derivación. En aplicaciones suelen asociarse al «significado» de la palabra generada.
- Definimos ambigüedad de gramáticas de contexto libre.