Más sobre gramáticas independientes de contexto o incontextuales

Elvira Mayordomo, Gregorio de Miguel

Universidad de Zaragoza

29 de noviembre de 2014

Contenido de este tema

- Árboles de derivación
- Gramáticas ambiguas
- Aplicaciones: el lenguaje natural y los lenguajes de programación
- Simplificación de gramáticas y forma normal de Chomsky
- Los analizadores sintácticos y bison

Árboles de derivación

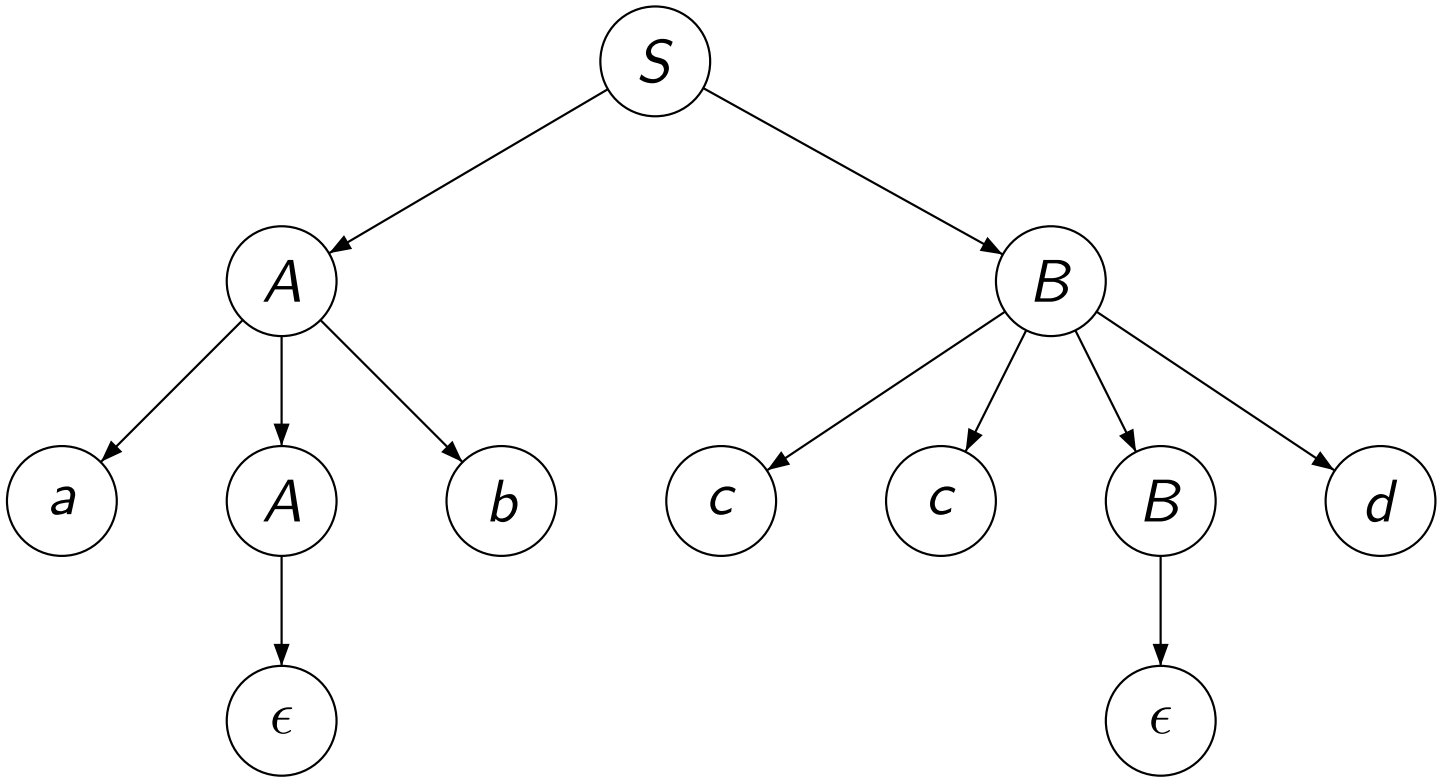
- Dada una gramática para cada palabra que genera tenemos la correspondiente derivación
- Un árbol de derivación representa una derivación
- Ejemplo:

$$\begin{aligned} S &\to AB \mid \epsilon \\ A &\to aAb \mid ab \\ B &\to ccBd \mid \epsilon \end{aligned}$$

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbccBd \Rightarrow aabbccd$$

Árboles de derivación

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbccBd \Rightarrow aabbccd$$



Derivación a izquierda

- El mismo árbol puede corresponder a varias derivaciones

$$S \Rightarrow AB \Rightarrow AccBd \Rightarrow aAbccBd \Rightarrow aabbccBd \Rightarrow aabbccd$$

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbccBd \Rightarrow aabbccd$$

- Pero son esencialmente la misma excepto el orden en que se han sustituído las variables
- Se suele elegir la "primera por la izquierda" (sustituir siempre la primera variable por la izda)

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbccBd \Rightarrow aabbccd$$

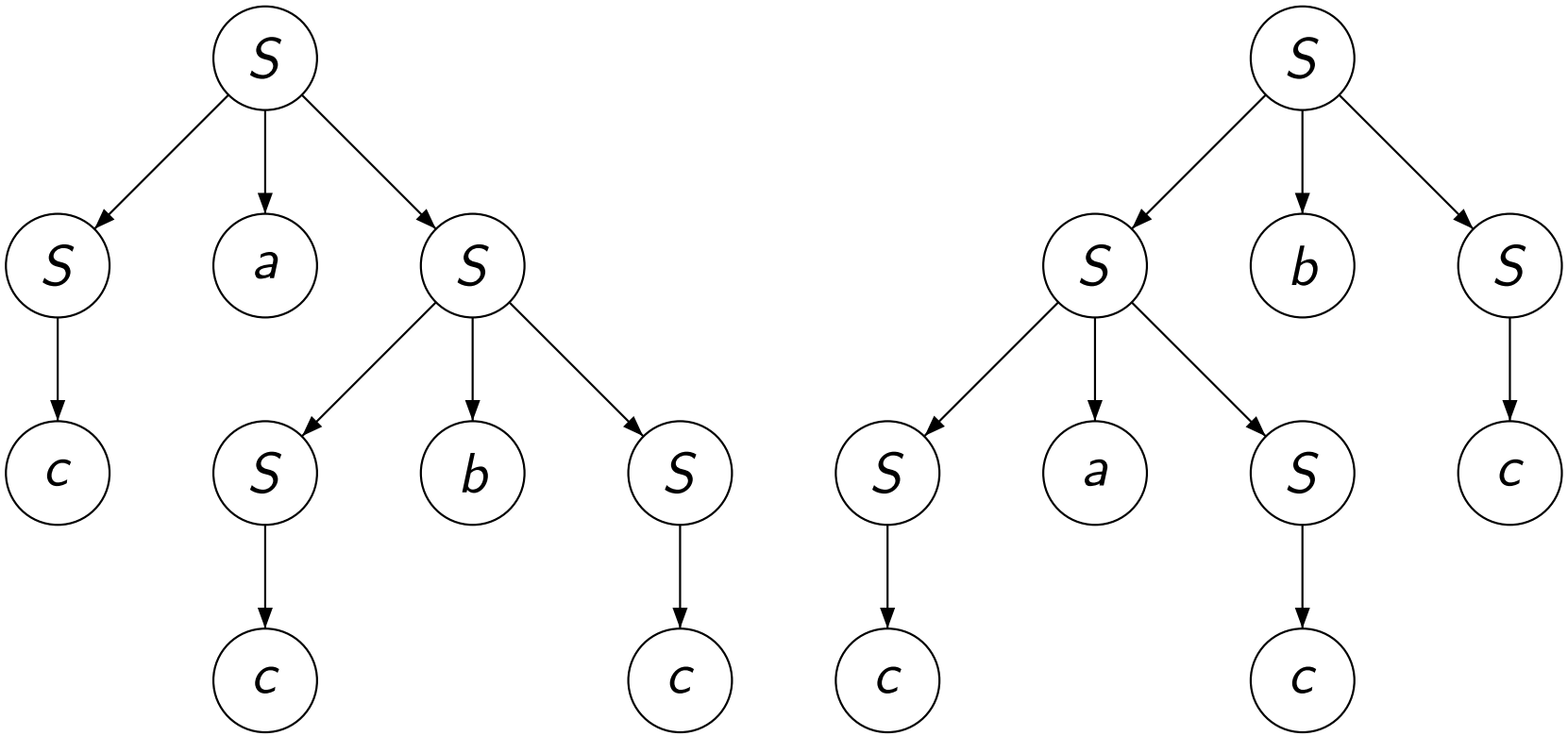
Contenido de este tema

- Árboles de derivación
- **Gramáticas ambiguas**
- Aplicaciones: el lenguaje natural y los lenguajes de programación
- Simplificación de gramáticas y forma normal de Chomsky
- Los analizadores sintácticos y bison

Ambigüedad

- La misma cadena puede tener varios árboles de derivación

$$S \to SaS \mid SbS \mid c$$



Gramática ambigua

- Una gramática se llama ambigua si existe una cadena con dos árboles de derivación distintos
- Esto es malo para las aplicaciones que veremos a continuación.

Ambigüedad inherente

- A veces se puede escribir otra gramática no ambigua para el mismo lenguaje. En lugar de $S \to SaS \mid SbS \mid c$

$$S \to Sac \mid Sbc \mid c$$

- Pero a veces todas las gramáticas que generan el lenguaje son ambiguas, se dice que en un lenguaje inherentemente ambiguo
- Por ejemplo el siguiente lenguaje es inherentemente ambiguo y el problema está en generar palabras de la forma $a^n b^n c^n$

$$L = \left\{ a^i b^j c^k \mid i = j \text{ ó } j = k \right\}$$

Contenido de este tema

- Árboles de derivación
- Gramáticas ambiguas
- **Aplicaciones: el lenguaje natural y los lenguajes de programación**
- Simplificación de gramáticas y forma normal de Chomsky
- Los analizadores sintácticos y bison

Aplicaciones de las gramáticas

- ¿Por qué nos interesan las gramáticas?
- ¿No nos basta con los autómatas de pila?
- En muchas aplicaciones no sólo queremos saber si la gramática genera una palabra sino cómo se ha generado (es decir, el árbol de derivación)

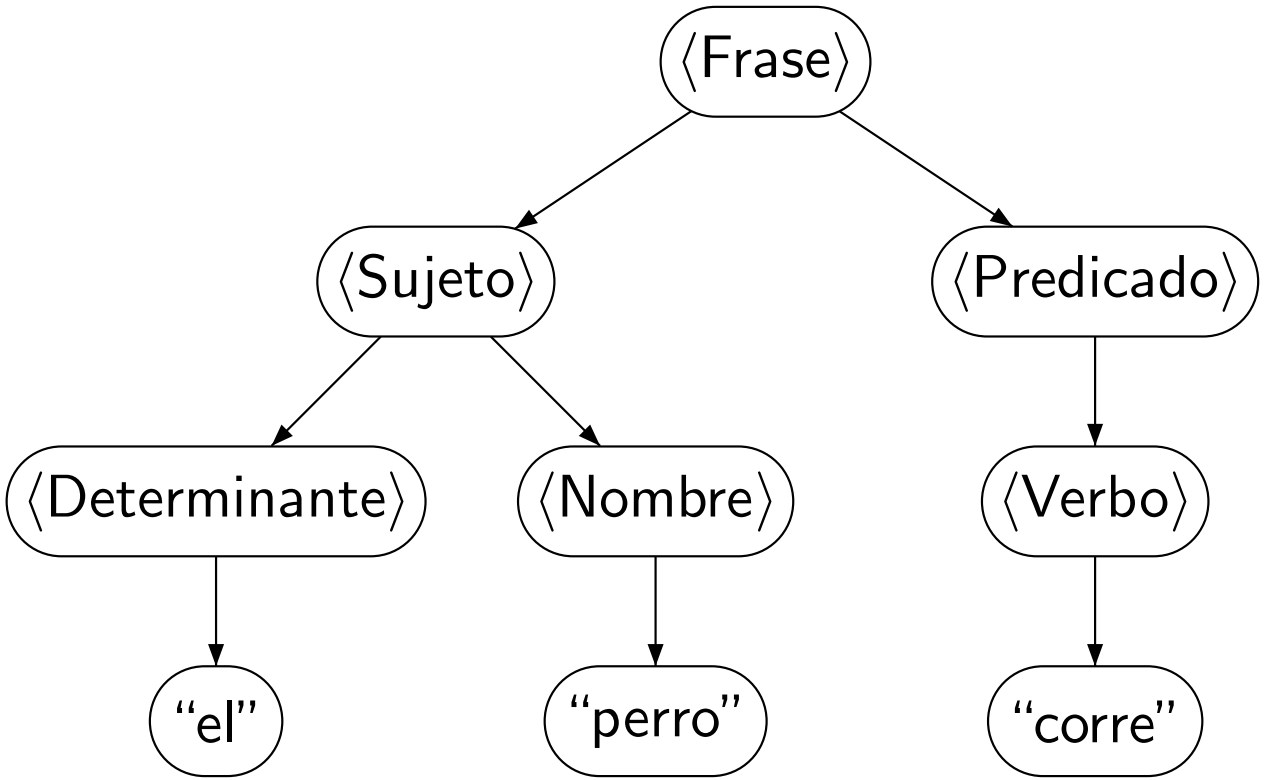
Lenguaje natural

- Las frases correctas en español se pueden representar por una gramática Un poco más complicada que incontextual, aquí la simplificamos
- Si las variables o no terminales las escribimos entre $\langle\ \rangle$ y los terminales o entrada entre " "

$$
\begin{aligned}
\langle \text{Frase} \rangle &\rightarrow \langle \text{Sujeto} \rangle \langle \text{Predicado} \rangle \\
\langle \text{Sujeto} \rangle &\rightarrow \langle \text{Determinante} \rangle \langle \text{Nombre} \rangle \\
\langle \text{Determinante} \rangle &\rightarrow \text{“el”} \mid \text{“este”} \\
\langle \text{Predicado} \rangle &\rightarrow \langle \text{Verbo} \rangle \\
\langle \text{Nombre} \rangle &\rightarrow \text{“perro”} \mid \text{“gato”} \\
\langle \text{Verbo} \rangle &\rightarrow \text{“corre”} \mid \text{“juega”}
\end{aligned}
$$

Lenguaje natural

- Para analizar la corrección, el significado, etc es necesario reconstruir el árbol de derivación



Lenguajes de programación

- La sintaxis de un lenguaje de programación se expresa con una gramática

$$
\begin{aligned}
\langle \text{Programa} \rangle &\rightarrow \text{“Begin”}\ \langle \text{ListaInstrucc.} \rangle\ \text{“End.”} \\
\langle \text{ListaInstrucc.} \rangle &\rightarrow \langle \text{ListaInstrucc.} \rangle \langle \text{Instrucción} \rangle \mid \langle \text{Instrucción} \rangle \\
\langle \text{Instrucción} \rangle &\rightarrow \langle \text{Asignación} \rangle \mid \langle \text{Bucle} \rangle \mid \langle \text{Condicional} \rangle \\
&\quad \mid \text{“Begin”}\ \langle \text{ListaInstrucc.} \rangle\ \text{“End;”} \\
\langle \text{Asignación} \rangle &\rightarrow \langle \text{Identificador} \rangle\ \text{“:=”}\ \langle \text{Expresión} \rangle \\
\langle \text{Bucle} \rangle &\rightarrow \text{“while”}\ \langle \text{Condición} \rangle\ \text{“do”}\ \langle \text{Instrucción} \rangle
\end{aligned}
$$

...

Lenguajes de programación

- Dado un programa podemos comprobar que es sintácticamente correcto
- Pero también queremos compilarlo, es decir, transformarlo en un ejecutable (o en un código intermedio)
- Y aun más, queremos señalar los errores si no es sintácticamente correcto

- Para todo ello necesitamos reconstruir el árbol de derivación, y anotarlo con información adicional (cómo afecta cada parte del programa al valor de las variables, por ejemplo)

Analizadores sintácticos o parsers

- En las dos aplicaciones anteriores hemos visto que dada una cadena es necesario reconstruir su árbol de derivación
- Un analizador sintáctico o parser es un programa capaz de:
  - reconocer si una cadena es generada por una gramática
  - reconstruir su árbol de derivación
  - conforme reconstruye la derivación puede realizar otras acciones
- Cuanto más simples sean las reglas de la gramática más eficiente será el parser
- A continuación vemos cómo simplificar una gramática

Contenido de este tema

- Árboles de derivación
- Gramáticas ambiguas
- Aplicaciones: el lenguaje natural y los lenguajes de programación
- **Simplificación de gramáticas y forma normal de Chomsky**
- Los analizadores sintácticos y bison

Forma normal de Chomsky de una gramática

- Una gramática G está en forma normal de Chomsky si todas las reglas de la gramática tienen una de las dos formas siguientes, para A, B, C no terminales y a terminal:

$$\begin{aligned} A &\to BC \\ A &\to a \end{aligned}$$

Además puede aparecer la regla $S \to \epsilon$, sólo en el caso de que no haya reglas con S en la parte derecha

Cómo obtener la forma normal de Chomsky de una gramática

Para convertir G a forma normal de Chomsky:

- Añadimos una nueva variable S_0 que es el nuevo símbolo inicial y añadimos la regla $S_0 \to S$ (esto sólo es necesario si existe la regla $S \to \epsilon$)
- Quitamos las ϵ-producciones, es decir, las reglas de la forma $A \to \epsilon$
- Quitamos las producciones unarias, es decir, las reglas de la forma $A \to B$
- Convertimos las reglas que quedan a las formas

$$\begin{aligned} A &\to BC \\ A &\to a \end{aligned}$$

Eliminación de ϵ-producciones

Repetir mientras quede alguna regla $A \to \epsilon$ que no sea $S_1 \to \epsilon$:

1. Quitar la regla $A \to \epsilon$
2. Añadir A al conjunto Quitadas
3. Para cada regla que tenga A en la parte derecha

$$B \to BAb$$

añadir una regla igual pero sin A

$$B \to BAb \,|\, Bb$$

4. Si A aparece varias veces, añadir también el resultado de quitarla una vez, dos veces, etc

$$B \to BAbAa$$

$$B \to BAbAa \,|\, BbAa \,|\, BAba \,|\, Bba$$

5. En los pasos 3. y 4., no añadir nunca una regla $A \to \epsilon$ para $A \in$ Quitadas

Eliminación de producciones unarias

Repetir mientras quede alguna regla $A \to B$:

1. Quitar la regla $A \to B$
2. Añadir $A \to B$ al conjunto Quitadas
3. Para cada regla $B \to u$ añadir $A \to u$
4. En el paso 3, no añadir nunca una regla $A \to B$ de Quitadas

Conversión final

Para cada regla $A \to \alpha_1 \dots \alpha_k$ con $\alpha_1 \dots \alpha_k$ terminales y variables:

1. Quitar la regla $A \to \alpha_1 \dots \alpha_k$
2. Añadir nuevas variables $A_1, \dots, A_{k-2}$
3. Añadir las reglas

$$\begin{aligned} A &\to \alpha_1 A_1 \\ A_1 &\to \alpha_2 A_2 \\ &\dots \\ A_{k-2} &\to \alpha_{k-1} A_{k-1} \\ A_{k-1} &\to \alpha_{k-1} \alpha_k \end{aligned}$$

4. Si α_i es un terminal, añadir una nueva variable U_i y sustituir $A_{i-1} \to \alpha_i A_i$ por las reglas

$$A_{i-1} \to U_i A_i, \qquad U_i \to \alpha_i$$

Ejemplo de conversión a forma normal de Chomsky

$$\begin{aligned} S &\to ASA \,|\, aB \\ A &\to B \,|\, S \\ B &\to b \,|\, \epsilon \end{aligned}$$

- Añadimos una nueva variable S_0 que es el nuevo símbolo inicial y la regla $S_0 \to S$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \,|\, aB \\ A &\to B \,|\, S \\ B &\to b \,|\, \epsilon \end{aligned}$$

Ejemplo de conversión a forma normal de Chomsky

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \,|\, aB \\ A &\to B \,|\, S \\ B &\to b \,|\, \epsilon \end{aligned}$$

- Quitamos las ϵ-producciones

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \,|\, aB \,|\, a \,|\, SA \,|\, AS \,|\, S \\ A &\to B \,|\, S \\ B &\to b \end{aligned}$$

Ejemplo de conversión a forma normal de Chomsky

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

- Quitamos las producciones unarias $S_0 \to S$, $S \to S$, $A \to B$, y $A \to S$

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to b \mid ASA \mid aB \mid a \mid SA \mid AS \\
B &\to b
\end{aligned}
$$

Ejemplo de conversión a forma normal de Chomsky

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to b \mid ASA \mid aB \mid a \mid SA \mid AS \\
B &\to b
\end{aligned}
$$

- Convertimos las reglas que quedan a $A \to BC$ y $A \to a$

$$
\begin{aligned}
S_0 &\to & AA_1 \mid UB \mid a \mid SA \mid AS \\
A_1 &\to SA & \\
U &\to a & \\
S &\to & AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\to b \mid & AA_1 \mid UB \mid a \mid SA \mid AS \\
B &\to b &
\end{aligned}
$$

Ejemplo de conversión a forma normal de Chomsky

$$\begin{aligned} S &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \epsilon \end{aligned}$$

- Es equivalente a la siguiente, en forma normal de Chomsky

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ A_1 &\to SA \\ U &\to a \\ S &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ A &\to b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ B &\to b \end{aligned}$$

Parsers para gramáticas en forma normal de Chomsky

- El parser CYK (Cocke, Younger y Kasami) construye el árbol de derivación para gramáticas en forma normal de Chomsky
- Dada una cadena $w = w_1 \dots w_n$ el parser construye árboles correspondientes a derivar $w_i \dots w_j$ a partir de la variable A
  - Si existe la regla $A \to w_i$ entonces construye el árbol de

$$A \Rightarrow w_i$$

  - Si existe la regla $A \to BC$ y ya tiene dos árboles para $B \Rightarrow^* w_i \dots w_j$ y $C \Rightarrow^* w_{j+1} \dots w_k$ entonces construye el árbol de

$$A \Rightarrow^* w_i \dots w_k$$

- El árbol buscado corresponde al de $S \Rightarrow^* w_1 \dots w_n$
- El parser CYK es eficiente (tiempo n^3)

Contenido de este tema

- Árboles de derivación
- Gramáticas ambiguas
- Aplicaciones: el lenguaje natural y los lenguajes de programación
- Simplificación de gramáticas y forma normal de Chomsky
- **Los analizadores sintácticos y bison**

Analizadores Sintácticos - Bison

- Un analizador sintáctico o parser es capaz de reconocer si una cadena es generada por una gramática, al ir reconstruyendo la derivación puede generar acciones
- Por ejemplo, un analizador sintáctico que comprueba que una expresión aritmética con paréntesis está bien escrita y calcula su valor
- Bison es un generador de analizadores sintácticos, a partir de las reglas de la gramática construye el analizador
- Ejemplo de fuente bison

...

$$
\begin{array}{rll}
exp: & NUM & \{\$\$ = \$1;\} \\
 & |\,\mathrm{exp}\ '+'\ \mathrm{exp} & \{\$\$ = \$1 + \$3;\} \\
 & |\,\mathrm{exp}\ '-'\ \mathrm{exp} & \{\$\$ = \$1 - \$3;\} \\
 & |\,'('\ \mathrm{exp}\ ')' & \{\$\$ = \$2;\} \\
; & &
\end{array}
$$

...

Curiosidades: cómo funciona bison

- GNU Bison es un generador de analizadores sintácticos que convierte una gramática libre de contexto (de tipo LALR(1) *"(Look-Ahead Left-to-Right)"*) en un programa C que analice esa gramática.
- Fue escrito originariamente por Robert Corbett

Curiosidades: Cómo funciona bison

- El algoritmo que utiliza el parser de bison es de tipo bottom-up
- El parser lee tokens y los apila en la pila del analizador sintáctico (desplazamiento)
- Cuando un conjunto de elementos de la pila (tokens o variables) encajan con una producción de una regla de la gramática, entonces se combinan de acuerdo con dicha regla y se sustituyen por el no terminal (reducción). Depende del tipo de token en prebúsqueda (lookahead).
- El parser intenta, mediante desplazamientos y reducciones, el reducir la entrada completa a un sólo agrupamiento, cuyo símbolo será el de la variable de la regla inicial
- Bison se ha empleado en el lenguaje de programación Ruby (YARV), PHP (Zend Parser), GCC (hasta 2004), Go (GC) y Bash (para analizar la linea de comandos)

Curiosidades: Cómo funciona bison

Ejemplo del uso del token de *lookahead*:

$$\begin{aligned} \mathit{expr} : &\quad \mathit{term}\ '+'\ \mathit{expr} \quad | \ \mathit{term}\ ; \\ \mathit{term} : &\quad '('\ \mathit{expr}\ ')' \quad | \ \mathit{term}\ '!' \ | \ \mathit{ENTERO}\ ; \end{aligned}$$

Supongamos que los tokens '1 + 2' han sido ya desplazados.

- ¿Qué debería hacerse si el siguiente token es ')'?
- Reducir los tres primeros tokens por *expr*
- ¿Y si el siguiente token fuera '!'?
- Desplazar '!' para reducir '2 !' por *term*

Bibliografía

- Sipser (2a edición), sección 2.1.
- Kelley, secciones 3.4 a 3.5.
- Manual de Bison