Escritura de gramáticas libres de contexto

Plan de la clase

1. Escritura de gramáticas
   - Repaso
   - Nociones de asociatividad y precedencia
   - Otros ejemplos
2. Reducción
   - Definiciones
   - Algoritmos
     - Símbolos activos
     - Símbolos alcanzables
     - Símbolos anulables
   - Reglas de renombramiento
   - Recursividad a izquierda
   - Factorización
   - Orden

- "Practicamente todos los lenguajes que se usan en computación son independientes de contexto."
- Como vimos, reconocer si una cadena pertenece a un lenguaje libre de contexto de manera determinista puede resultar no eficiente. (Los autómatas de pila no deterministas no siempre pueden pasarse a deterministas.)
- Vamos a ver distintas técnicas de parsing. Es decir, dada una gramática, tener un algoritmo que nos diga si una determinada cadena pertenece al lenguaje de esa gramática, y obtener un árbol de derivación de la misma, de manera que además comprendamos su estructura sintáctica.
- Y despues vamos a ver como agregar acciones a los parsers de manera de aprovechar que tenemos la estructura para procesar la cadena de entrada en forma sistemática.

Repaso

Ambigüedad: por ejemplo, la gramática

$$\mathrm{S} \rightarrow \mathrm{aS}|\mathrm{Sa}|\lambda$$

es ambigua (ver dos árboles de derivación para a) pero es equivalente a la gramática no ambigua:

$$\mathrm{S} \rightarrow \mathrm{aS}|\lambda$$

Recordar: árboles de derivación sólo se aplican a gramáticas libres de contexto (y no a las sensibles al contexto o sin restricciones).

Asociatividad y precedencia

Supongamos que queremos definir un lenguaje de la lógica de primer orden (sólo predicados, sin funciones ni constantes). Una primera idea podría ser:

$$\begin{array}{lcl} \mathrm{E} & \to & \mathrm{A} \mid \mathrm{E} \ \wedge \ \mathrm{E} \mid \mathrm{E} \ \vee \ \mathrm{E} \mid \mathrm{E} \ \Rightarrow \ \mathrm{E} \mid \neg \ \mathrm{E} \mid (\ \mathrm{E}\ ) \mid \forall \ \mathrm{id_V} \ \mathrm{E} \mid \exists \ \mathrm{id_V} \ \mathrm{E} \\ \mathrm{A} & \to & \mathrm{id_P} \ (\ \mathrm{L}\ ) \\ \mathrm{L} & \to & \mathrm{id_V} \mid \mathrm{L} \ , \ \mathrm{L} \end{array}$$

Donde $\mathrm{id_V}$ representa algún símbolo de variable (como $\mathrm{x}_1, \mathrm{x}_2, \ldots$, y $\mathrm{id_P}$ es algún símbolo de relación (predicado), como $=$, $\leq$, R, etc.)

Asociatividad y precedencia

Una gramática no sólo nos sirve para definir un lenguaje, sino también para entender su estructura. En el proceso de reconocimiento de una cadena, obtenemos el *árbol de derivación*, que si la gramática es apropiada nos puede ayudar a entender el significado de la expresión.

Asociatividad y precedencia

La gramática definida es bastante ambigua...
Ejemplo:

$$\mathrm{A} \vee \mathrm{A} \wedge \mathrm{A}$$

Hay dos árboles de derivación que producen esta cadena. Uno puede interpretarse como $(\mathrm{A} \vee \mathrm{A}) \wedge \mathrm{A}$, y el otro como $\mathrm{A} \vee (\mathrm{A} \wedge \mathrm{A})$. Queremos una gramática equivalente en la cual únicamente sea posible esta última derivación-interpretación. Es decir, queremos representar la precedencia mayor del $\wedge$ por sobre el $\vee$ de alguna manera en la gramática. Se puede ver una intuición de que uno quiere que el $\vee$ aparezca primero más arriba en el árbol de derivación. Surge la idea de jerarquizar niveles en la gramática dependiendo de la precedencia.

Asociatividad y precedencia

Por otro lado, si tenemos:

$$\mathrm{A} \wedge \mathrm{A} \wedge \mathrm{A}$$

elegimos según asociatividad (digamos que a izquierda), de modo que se interprete:

$$(\mathrm{A} \wedge \mathrm{A}) \wedge \mathrm{A}$$

Idea de cómo hacer esto con la gramática.

Asociatividad y precedencia

También con:

$$\mathrm{A} \Rightarrow \mathrm{A} \Rightarrow \mathrm{A}$$

En este caso elegimos asociatividad a derecha

$$\mathrm{A} \Rightarrow (\mathrm{A} \Rightarrow \mathrm{A})$$

(Otro ejemplo de asociatividad a derecha sería la exponenciación: 2^{3^4} se interpreta como $2^{(3^4)}$)

Asociatividad y precedencia

Recordar que asociatividad a izquierda y a derecha no tiene vínculo con derivación más a la izquierda o más a la derecha.
Armamos entonces la tabla de precedencia (las primeras líneas tienen la menor precedencia y las últimas la mayor):

$\Rightarrow$	(Asoc. a derecha)
$\vee$	(Asoc. a izq.)
$\wedge$	(Asoc. a izq.)
unarios:	($\forall$ $\exists$ $\neg$)

Sí, la precedencia de los operadores unarios también debe ser especificada; tienen la mayor precedencia. Ejemplo:

$$\neg \, \mathrm{A} \ \wedge \ \mathrm{A}$$

Ambigüedad

¿Cómo eliminar la ambigüedad y tener el árbol que queremos? No siempre se puede: hay lenguajes inherentemente ambiguos, y en general no se puede determinar si una gramática es ambigua o no. En casos como este, si empezamos por los operadores de menor precedencia podemos pensar que nos subdividen la entrada en subárboles.
Ejemplo:

$$\mathrm{A} \vee \mathrm{A} \Rightarrow \mathrm{A} \wedge \mathrm{A} \Rightarrow \mathrm{A} \wedge \mathrm{A} \vee \mathrm{A}$$

Ambigüedad

Ahora, separamos de menor a mayor precedencia, poniendo símbolos no terminales y reglas de producción para cada nivel de precedencia. Nos queda la siguiente gramática:

$$
\begin{aligned}
\mathrm{E} &\rightarrow \mathrm{D} \Rightarrow \mathrm{E} \mid \mathrm{D} \\
\mathrm{D} &\rightarrow \mathrm{D} \vee \mathrm{C} \mid \mathrm{C} \\
\mathrm{C} &\rightarrow \mathrm{C} \wedge \mathrm{U} \mid \mathrm{U} \\
\mathrm{U} &\rightarrow \forall\ \mathrm{id_V}\ \mathrm{U} \mid \exists\ \mathrm{id_V}\ \mathrm{U} \mid \neg\ \mathrm{U} \mid \mathrm{A} \\
\mathrm{A} &\rightarrow \mathrm{id_P}\ (\ \mathrm{L}\ ) \mid (\ \mathrm{E}\ ) \\
\mathrm{L} &\rightarrow \mathrm{id_V} \mid \mathrm{L}\ ,\ \mathrm{L}
\end{aligned}
$$

Ambigüedad

Todavía podemos mejorar esto: en realidad, no hace falta separar U y A (de alguna forma, son todas operaciones unarias):

$$\mathrm{U} \ \rightarrow \ \forall \ \mathrm{id_V} \ \mathrm{U} \mid \exists \ \mathrm{id_V} \ \mathrm{U} \mid \neg \ \mathrm{U} \mid \mathrm{id_P} \ (\ \mathrm{L}\ ) \mid (\ \mathrm{E}\ )$$

(Esto motiva la escritura de gramáticas, que veremos a continuación.)
Además, la parte de la lista de variables sigue siendo ambigua, así que hacemos el siguiente cambio:

$$\mathrm{L} \ \rightarrow \ \mathrm{id_V} \mid \mathrm{L} \ , \ \mathrm{id_V}$$

Otros ejemplos

Con aritmética:

$$\begin{array}{lcl} \mathrm{E} & \rightarrow & \mathrm{E} + \mathrm{T} \mid \mathrm{T} \\ \mathrm{T} & \rightarrow & \mathrm{T} * \mathrm{F} \mid \mathrm{F} \\ \mathrm{F} & \rightarrow & \mathrm{id} \mid (\ \mathrm{E}\ ) \end{array}$$

es una gramática no ambigua

Tampoco hay problema en que haya dos operadores con la misma precedencia:

$$\begin{array}{lcl} \mathrm{E} & \rightarrow & \mathrm{E} + \mathrm{T} \mid \mathrm{E} - \mathrm{T} \mid \mathrm{T} \\ \mathrm{T} & \rightarrow & \mathrm{T} * \mathrm{F} \mid \mathrm{T} / \mathrm{F} \mid \mathrm{F} \\ \mathrm{F} & \rightarrow & \mathrm{id} \mid (\ \mathrm{E}\ ) \end{array}$$

Sigue siendo una gramática no ambigua. Observar que la resta es asociativa a izquierda: $3 - 2 - 1$ interpretamos como $(3 - 2) - 1$ y no como $3 - (2 - 1)$.

Definiciones

$\mathrm{X} \in \mathrm{V_N}$ es *útil* si participa en alguna derivación que genere una cadena de terminales. Es decir, si hay una derivación:

$$\mathrm{S} \Rightarrow^* \alpha\mathrm{X}\beta \Rightarrow^* \alpha\gamma\beta$$

con $\alpha, \beta, \gamma \in \mathrm{V_T^*}$.
Notar que X es útil si suceden dos cosas:

- $\mathrm{X} \in \mathrm{V_N}$ es *alcanzable* si aparece en alguna forma sentencial, es decir, si hay $\alpha, \beta \in \mathrm{V}^*$ tales que:

$$\mathrm{S} \Rightarrow^* \alpha\mathrm{X}\beta$$

- $\mathrm{X} \in \mathrm{V_N}$ es *activo* si puede generar una cadena de terminales, es decir, si existe alguna derivación:

$$\mathrm{X} \Rightarrow^* \gamma$$

con $\gamma \in \mathrm{V_T^*}$.

Definiciones

Como uno esperaría, se define que X es *inalcanzable* si no es alcanzable, X es *inactivo* si no es activo, y es *inútil* si no es útil.

Ejemplo

Ejemplos de gramáticas con símbolos...

- alcanzables pero inactivos
- activos pero inalcanzables
- inalcanzables inactivos
- alcanzables y activos (útiles)

Una gramática puede reducirse a otra equivalente sin símbolos inútiles.

Algoritmos

Es fácil determinar si hay símbolos inútiles, calculando los conjuntos de símbolos activos y de símbolos alcanzables, y viendo que ambos conjuntos sean iguales a $\mathrm{V_N}$.
Los algoritmos que damos calculan los símbolos que tienen cierta propiedad, para luego eliminar las producciones que los incluyen.
Los algoritmos que damos terminan cuando en una iteración el conjunto que obtenemos es el mismo que en el paso anterior. No son eficientes, sólo muestran que los conjuntos se pueden calcular.

Símbolos activos

$\mathrm{N} = \emptyset$

repetir hasta que N no cambie

$$\mathrm{N} = \mathrm{N} \cup \{\mathrm{A} \mid \mathrm{A} \to \alpha \in \mathrm{P} \wedge \alpha \in (\mathrm{N} \cup \mathrm{V_T})^*\}$$

Gramática ejemplo:

$$\begin{aligned} \mathrm{S} &\to \mathrm{a} \mid \mathrm{aA} \mid \mathrm{aC} \\ \mathrm{A} &\to \mathrm{A\ B} \\ \mathrm{B} &\to \mathrm{b} \\ \mathrm{C} &\to \mathrm{c\ C} \end{aligned}$$

Símbolos alcanzables

$\mathrm{N} = \mathrm{S}$

repetir hasta que N no cambie

$$\mathrm{N} = \mathrm{N} \cup \{\mathrm{X} \mid \mathrm{A} \to \alpha \mathrm{X} \beta \in \mathrm{P} \wedge \mathrm{A} \in \mathrm{N}\}$$

Gramática ejemplo:

$$\begin{aligned} \mathrm{S} &\to \mathrm{a} \mid \mathrm{aA} \mid \mathrm{aC} \\ \mathrm{A} &\to \mathrm{A\ B} \\ \mathrm{B} &\to \mathrm{b} \\ \mathrm{C} &\to \mathrm{c\ C} \end{aligned}$$

Símbolos inútiles

Para eliminar inútiles:

- Eliminar inactivos.
- Eliminar inalcanzables.

Hay que hacerlo en este orden. ¿Por qué?
Porque tal vez un símbolo era alcanzable a través de uno inactivo.
En cambio, si un símbolo para ser activo usa otros inalcanzables es porque él mismo era inalcanzable...

Símbolos inútiles

Para eliminar inútiles:

- Eliminar inactivos.
- Eliminar inalcanzables.

Ejemplo:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{b} \mid \mathrm{I} \\ \mathrm{I} &\rightarrow \mathrm{A}\ \mathrm{B} \\ \mathrm{A} &\rightarrow \mathrm{b} \end{aligned}$$

Si eliminamos los inalcanzables primero, no se elimina ninguno. Entonces, al eliminar los inactivos (I, B), quedaría

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{b} \\ \mathrm{A} &\rightarrow \mathrm{b} \end{aligned}$$

Símbolos anulables

También es práctico saber cuáles son los símbolos *anulables* de una gramática, es decir, los símbolos X tales que $\mathrm{X} \Rightarrow^* \lambda$. Los podemos calcular de forma similar a los anteriores:

$\mathrm{N} = \{\mathrm{A} \mid \mathrm{A} \to \lambda \in \mathrm{P}\}$
repetir
$\mathrm{N} = \mathrm{N} \cup \{\mathrm{A} \mid \mathrm{A} \to \alpha \in \mathrm{P} \wedge \alpha \in \mathrm{N}^*\}$
hasta que N no cambie

Para eliminar las producciones λ: remplazar cada regla de la forma $\mathrm{B} \to \alpha_1\mathrm{A}_1\alpha_2\mathrm{A}_2\alpha_3 \ldots \alpha_\mathrm{k}\mathrm{A}_\mathrm{k}\alpha_{\mathrm{k}+1}$, con $\mathrm{k} \geq 1$ y $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_\mathrm{k}$ anulables, por todas las reglas de la forma $\mathrm{B} \to \alpha_1\beta_1\alpha_2\beta_2\alpha_3 \ldots \alpha_\mathrm{k}\beta_\mathrm{k}\alpha_{\mathrm{k}+1}$ tales que $\beta_\mathrm{j} = \mathrm{A}_\mathrm{j}$ o λ para todo $1 \leq \mathrm{j} \leq \mathrm{k}$, y eliminar cada regla $\mathrm{A}_\mathrm{j} \to \lambda$, excepto $\mathrm{S} \to \lambda$ si aparece, en ese caso S no debe aparecer a derecha.

Símbolos anulables

Gramática ejemplo:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{d} \mid \mathrm{aA} \\ \mathrm{A} &\rightarrow \mathrm{bA} \mid \mathrm{BBC} \\ \mathrm{B} &\rightarrow \mathrm{bB} \mid \lambda \\ \mathrm{C} &\rightarrow \mathrm{cC} \mid \lambda \end{aligned}$$

$\mathrm{N} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$
Resulta:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{d} \mid \mathrm{aA} \mid \mathrm{a} \\ \mathrm{A} &\rightarrow \mathrm{bA} \mid \mathrm{b} \mid \mathrm{BBC} \mid \mathrm{B} \mid \mathrm{BB} \mid \mathrm{C} \mid \mathrm{BC} \\ \mathrm{B} &\rightarrow \mathrm{bB} \mid \mathrm{b} \\ \mathrm{C} &\rightarrow \mathrm{cC} \mid \mathrm{c} \end{aligned}$$

Reglas de renombramiento

Una regla de renombramiento es una producción de la forma $\mathrm{A} \to \mathrm{B}$, con $\mathrm{A}, \mathrm{B} \in \mathrm{V_N}$.

1. Construir para cada A el conjunto $\mathrm{N_A} = \{\mathrm{B} \in \mathrm{V_N} \mid \mathrm{A} \Rightarrow^* \mathrm{B}\}$

   $\mathrm{N_A} = \mathrm{A}$
   repetir hasta que $\mathrm{N_A}$ no cambie
   $\mathrm{N_A} = \mathrm{N_A} \cup \{\mathrm{C} \mid \mathrm{B} \to \mathrm{C} \in \mathrm{P} \wedge \mathrm{B} \in \mathrm{N_A}\}$

2. Para cada $\mathrm{B} \to \alpha$, agregar $\mathrm{A} \to \alpha$ si $\mathrm{B} \in \mathrm{N_A}$.
3. Eliminar todas las producciones de la forma $\mathrm{A} \to \mathrm{B}$

Renombramientos

Gramática ejemplo:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{A} \\ \mathrm{A} &\rightarrow \mathrm{a} \mid \mathrm{B} \\ \mathrm{B} &\rightarrow \mathrm{A} \end{aligned}$$

$\mathrm{N_A} = \{\mathrm{B} \in \mathrm{V_N} \mid \mathrm{A} \Rightarrow^* \mathrm{B}\}$
Para cada $\mathrm{B} \rightarrow \alpha$, agregar $\mathrm{A} \rightarrow \alpha$ si $\mathrm{B} \in \mathrm{N_A}$.
Quedará

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{a} \\ \mathrm{A} &\rightarrow \mathrm{a} \\ \mathrm{B} &\rightarrow \mathrm{a} \end{aligned}$$

Al eliminar inútiles, quedará

$$\mathrm{S} \rightarrow \mathrm{a}$$

Ciclos

Cómo eliminar ciclos, es decir la posibilidad de $\mathrm{A} \Rightarrow^{+} \mathrm{A}$.
De no haber producciones λ, $\mathrm{A} \Rightarrow^{+} \mathrm{A}$ sólo es posible si hay al menos una regla de renombramiento que involucre a A.
Por eso conviene previamente eliminar las producciones λ.

Recursividad a izquierda

Una gramática es recursiva a izquierda si existe un no terminal A tal que $\mathrm{A} \Rightarrow^{+} \mathrm{A}\alpha$ para alguna cadena α. ¿Cómo eliminar esta condición?
Cambiar cada conjunto de producciones de la forma $\mathrm{A} \to \mathrm{A}\alpha \mid \beta$ por

$$\begin{aligned} \mathrm{A} &\to \beta\mathrm{A}' \\ \mathrm{A}' &\to \alpha\mathrm{A}' \mid \lambda \end{aligned}$$

agregando un nuevo $\mathrm{A}' \notin \mathrm{V_N}$.

Recursividad a izquierda

Gramática ejemplo:

$$\begin{aligned} \mathrm{E} &\rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T} \\ \mathrm{T} &\rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F} \\ \mathrm{F} &\rightarrow (\mathrm{E}) \mid \mathrm{id} \end{aligned}$$

Quedará

$$\begin{aligned} \mathrm{E} &\rightarrow \mathrm{TE}' \\ \mathrm{E}' &\rightarrow +\mathrm{TE}' \mid \lambda \\ \mathrm{T} &\rightarrow \mathrm{FT}' \\ \mathrm{T}' &\rightarrow *\mathrm{FT}' \mid \lambda \\ \mathrm{F} &\rightarrow (\mathrm{E}) \mid \mathrm{id} \end{aligned}$$

Después se puede hacer necesario eliminar reglas λ.

Factorización a izquierda

Cambiar cada conjunto de producciones de la forma $\mathrm{A} \to \alpha\beta_1 \mid \alpha\beta_2 \ \ldots \alpha\beta_\mathrm{n} \mid \gamma$, donde $\alpha \notin \mathrm{Ini}(\gamma)$, por

$$\begin{array}{lcl} \mathrm{A} & \to & \alpha\mathrm{A}' \mid \gamma \\ \mathrm{A}' & \to & \beta_1 \mid \beta_2 \mid \ \ldots \ \mid \beta_\mathrm{n} \end{array}$$

agregando un nuevo $\mathrm{A}' \notin \mathrm{V_N}$.

Factorización a izquierda

Gramática ejemplo:

$$\begin{aligned} \mathrm{P} &\rightarrow \mathrm{iEtP} \mid \mathrm{iEtPeP} \mid \mathrm{a} \\ \mathrm{E} &\rightarrow \mathrm{b} \end{aligned}$$

¿Qué representa en lenguajes de programación?
El **if** . . . **then** . . . **else** . . .
Quedará

$$\begin{aligned} \mathrm{P} &\rightarrow \mathrm{iEtPP'} \mid \mathrm{a} \\ \mathrm{P'} &\rightarrow \mathrm{eP} \mid \lambda \\ \mathrm{E} &\rightarrow \mathrm{b} \end{aligned}$$

Gramática reducida y propia

Una gramática es *reducida* si no tiene símbolos inútiles.
(Algunos piden también que símbolo inicial no aparezca en el lado derecho de ninguna producción. Para que ocurra esto, podría ser necesario introducir un renombramiento.)
Una gramática es *propia* si no tiene producciones λ, ni ciclos, ni símbolos inútiles.

Orden en la sala

¿En qué orden aplicar los algoritmos? Sin ser el único, un orden posible es:

1. Eliminar recursividad a izquierda
2. Eliminar producciones λ
3. Eliminar renombramientos
4. Eliminar inútiles
    1. Eliminar inactivos
    2. Eliminar inalcanzables