

Introducción a la Lógica y la Computación

Parte III: Lenguajes y Autómatas

Clase del 12 de Noviembre de 2014

Lenguajes Regulares vs. Lenguajes No Regulares

- ▶ Hasta ahora estudiamos la teoría de los lenguajes regulares, y diferentes formalismos para definirlos...(cuales?).
- ▶ Pero también mencionamos que había diferentes lenguajes que no son regulares.
- ▶ Sabemos cómo definir LR, pero no sabemos cómo decidir cuando un lenguaje no es regular.
- ▶ Vamos a ver un lema que nos va a ayudar.

Pumping Lemma

Proposición

Sea L un lenguaje regular. Entonces existe un número $k > 0$ tal que para toda cadena $\alpha \in L$ con $|\alpha| \geq k$, existen cadenas β_1, β_2, γ con $|\beta_1\gamma| \leq k$, $|\gamma| > 0$ que satisfacen:

1. $\alpha = \beta_1\gamma\beta_2$,
2. $\beta_1\gamma^n\beta_2 \in L$, para todo $n \geq 1$.

Demostración: ...al pizarrón.

Pumping Lemma vs. Lenguajes No Regulares

- ▶ Vimos el Pumping Lemma como una propiedad de los lenguajes regulares.
- ▶ Dijimos que lo íbamos a introducir como una herramienta para demostrar que un lenguaje **no** es regular.
- ▶ Entonces, ¿cómo lo usamos? Fácil! haciendo demostraciones por el absurdo, suponiendo que un lenguaje es regular.

Pumping Lemma vs. Lenguajes No Regulares - Ejemplo

Ejemplo

Demostremos que el lenguaje $L = \{a^n b^n \mid n > 1\}$ no es regular.

Demostración: Vamos a hacer la demostración por el absurdo, suponiendo que L es regular, y usando el Pumping Lemma para llegar a una contradicción. Tenemos que elegir $\alpha \in L$ de tal forma de obtener la contradicción que buscamos.

Elegimos $\alpha = a^k b^k$. Por el Pumping Lemma, existen β_1, γ, β_2 con $|\beta_1 \gamma| \leq k$, $|\gamma| > 0$, que satisfacen $\alpha = \beta_1 \gamma \beta_2$ y $\beta_1 \gamma^n \beta_2 \in L$ para todo $n \geq 1$.

Como $|\beta_1 \gamma| \leq k$, γ es de la forma a^s , con $s > 0$, luego $\alpha = a^r a^s a^t b^k$, con $r + s + t = k$ y $s > 0$. Además, $a^r a^{s \cdot n} a^t b^k \in L$, para todo $n \geq 1$.

Pero $r + s \cdot n + t > k$, para $n > 1$, entonces $a^r a^{s \cdot n} a^t b^k \notin L$! Lo cual es una contradicción (que vino de suponer que L era regular).

Pumping Lemma - Esquema de demostración

El Pumping Lemma es utilizado para detectar si un lenguaje es regular con la siguiente estrategia:

1. Seleccionar el lenguaje L que se desea verificar que no es regular.
2. Suponer que es regular y por lo tanto existe k como en el lema.
3. Seleccionar una cadena $\alpha \in L$ de longitud mayor que k . La elección de la cadena no es arbitraria, y depende del lenguaje dado.
4. Descomponer la cadena de acuerdo al Pumping Lemma y generar nuevas cadenas en L , inflando en el centro (es decir, las cadenas $\beta_1\gamma^n\beta_2$).
5. Verificar que las cadenas “infladas” no pertenecen a L . Esto genera una contradicción proveniente de suponer que L era regular.

Observación

Nótese que la estrategia anterior puede no funcionar! De todas maneras, que no podamos aplicar el Pumping Lemma a un lenguaje, no implica que éste sea regular.

Lenguajes no regulares

- ▶ Vimos como demostrar que un lenguaje no es regular.
- ▶ Antes vimos también, que hay diferentes tipos de lenguajes no regulares (recordar jerarquía de Chomsky).
- ▶ Entonces, ¿nos interesa algún tipo particular de lenguajes?
- ▶ **Si!** Queremos bajar un poco las restricciones del lenguaje, pero no tanto...

Lenguajes libres de contexto

¿Por qué nos interesan estos lenguajes en particular?

- ▶ La respuesta es que, hay lenguajes interesantes en las ciencias de la computación, que pueden ser definidos en esta clase.
- ▶ Son lenguajes que poseen el equilibrio (que buscamos, al menos) entre **expresividad** y **buen comportamiento**.
- ▶ Vamos a ver ejemplos de lenguajes muy simples (pero a su vez muy útiles para nosotros) que no son regulares, pero sí son libres de contexto.

Gramáticas libres de contexto - Ejemplo

Consideremos por ejemplo los números enteros. La gramática que define a los enteros puede ser:

$$\begin{aligned}\langle \text{entero} \rangle &\rightarrow \langle \text{entero_con_signo} \rangle \mid \langle \text{entero_sin_signo} \rangle \\ \langle \text{entero_con_signo} \rangle &\rightarrow +\langle \text{entero_sin_signo} \rangle \mid -\langle \text{entero_sin_signo} \rangle \\ \langle \text{entero_sin_signo} \rangle &\rightarrow \langle \text{digito} \rangle \mid \langle \text{digito} \rangle \langle \text{entero_sin_signo} \rangle \\ \langle \text{digito} \rangle &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

Esta gramática claramente no es regular. Pero es **libre de contexto**, como buscábamos.

BNF (Backus-Naur Form)

Consideremos por ejemplo los números enteros. La gramática que define a los enteros puede ser:

$$\begin{aligned}\langle \text{entero} \rangle & ::= \langle \text{entero_con_signo} \rangle \mid \langle \text{entero_sin_signo} \rangle \\ \langle \text{entero_con_signo} \rangle & ::= +\langle \text{entero_sin_signo} \rangle \mid -\langle \text{entero_sin_signo} \rangle \\ \langle \text{entero_sin_signo} \rangle & ::= \langle \text{digito} \rangle \mid \langle \text{digito} \rangle \langle \text{entero_sin_signo} \rangle \\ \langle \text{digito} \rangle & ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

Las BNF son usualmente utilizadas para definir, por ejemplo, la sintaxis de los lenguajes de programación (C, Pascal, Java, etc).

Gramáticas libres de contexto - Ejemplo 2

Consideremos el lenguaje visto en la parte de “Lógica” de la materia. Una gramática libre de contexto que define ese lenguaje puede ser:

$$\begin{aligned} F &\rightarrow (F \wedge F) \mid (F \vee F) \mid (F \leftrightarrow F) \mid (F \rightarrow F) \mid \neg(F) \mid A \\ A &\rightarrow \perp \mid p_1 \mid \dots \mid p_n. \end{aligned}$$

Podemos derivar la expresión $((p_1 \vee p_2) \wedge \perp)$:

$$\begin{aligned} F &\Rightarrow (F \wedge F) \\ &\Rightarrow (F \wedge A) \\ &\Rightarrow (F \wedge \perp) \\ &\Rightarrow ((F \vee F) \wedge \perp) \\ &\Rightarrow ((F \vee A) \wedge \perp) \\ &\Rightarrow ((F \vee p_2) \wedge \perp) \\ &\Rightarrow ((A \vee p_2) \wedge \perp) \\ &\Rightarrow ((p_1 \vee p_2) \wedge \perp) \end{aligned}$$