Tema 2: Lenguajes libres de contexto

Autómatas y Matemáticas Discretas

Escuela de Ingeniería Informática
Universidad de Oviedo

ÍNDICE

1. Introducción

Lenguajes regulares

- palabras reservadas de un lenguaje de programación (**conjunto finito**)
- nombres de identificadores
- constantes numéricas
- símbolos de puntuación (**conjunto finito**) = ;

Lenguajes no regulares

- $\{a^n b^n | n > 0\}$
- estructuras de bloque ({ } en C++ y `begin ... end` en Pascal)
- expresiones `((a + b) - c)`

Lenguajes libres de contexto y lenguajes regulares

- Los lenguajes libres de contexto son la siguiente clase de lenguajes tras los lenguajes regulares
- Todos los lenguajes regulares son libres de contexto, pero no todos los lenguajes libres de contexto son regulares
- Se definen mediante gramáticas libres de contexto
- Son aceptados por autómatas con pila

2. GRAMÁTICAS LIBRES DE CONTEXTO

- Las gramáticas definen lenguajes mediante reglas que nos permiten generar o producir cadenas válidas del lenguaje.
- Un ejemplo de regla de una gramática:

  `<sentence> -> <noun-phrase> <verb-phrase>`
- Estas reglas a veces se denominan reglas de reescritura o reglas de substitución: `<sentence>` puede ser reescrito como `<noun-phrase>` seguido por `<verb-phrase>`.

Ejemplo

```
<sentence>->  <noun-phrase> <verb-phrase>(1)
<noun-phrase>   ->  <proper-noun>(2)
<noun-phrase>   ->  <determiner> <common-noun>(3)
<proper-noun>   ->  John (4)
<common-noun>   ->  car (5)
<common-noun>   ->  hamburger (6)
<determiner>   ->  a (7)
<determiner>->  the (8)
<verb-phrase>   ->  <verb> <adverb>(9)
<verb-phrase>   ->  <verb>(10)
<verb>   ->  drives (11)
<verb>   ->  eats (12)
<adverb>   ->  slowly (13)
<adverb>   ->  frequently (14)
```

Gramáticas libres de contexto

Variables, V Conjunto finito y no vacío de no terminales, es decir, aquellos símbolos que pueden ser substituidos por otros. Cada variable representa un estado diferente en la generación de cadenas.

Terminales, T Conjunto finito y no vacío de símbolos (alfabeto) que pueden aparecer en las cadenas del lenguaje.

Axioma, S (también llamada variable de inicio) $S \in V$, es el símbolo al que se comienzan a aplicar las reglas de substitución para obtener las cadenas del lenguaje.

Produccciones, P Reglas de reescritura o de substitución o producciones que especifican qué símbolos pueden substituir a qué otros.

En una gramática libre de contexto (GLC) son de la forma $A \to \alpha$ donde $A \in V$ y $\alpha \in (V \cup T)^*$ es una cadena de símbolos de V y T.

Se dice que la gramática es libre de contexto porque las substituciones se pueden realizar en cualquier sitio, independientemente de qué haya alrededor de A.

Generación de cadenas mediante una gramática

- Para generar una cadena del lenguaje, comenzamos con una cadena que sólo contiene a la variable de inicio.
- Aplicamos las producciones a la cadena, reemplazando una aparición de la variable de la cabeza de la regla por la cadena de símbolos del cuerpo de la regla.
- Repetimos el proceso hasta que ya no queden más variables (sólo queden terminales).
- Cada secuencia particular de reglas aplicadas al axioma **produce** una cadena del lenguaje.

Ejemplo

```
<sentence>  => <noun-phrase> <verb-phrase> por (1)
            => <proper-noun> <verb-phrase> por (2)
            => Jill <verb-phrase>          por (5)
            => Jill <verb> <adverb>        por (10)
            => Jill drives <adverb>        por (12)
            => Jill drives frequently      por (15)
```

Derivación de una cadena

$G = (V, T, S, P)$ con $V = \{\texttt{E}\}$, $T = \{+, -, *, /, (,), \texttt{id}\}$, $S = \texttt{E}$ y P:

```
E -> E + E (1) | E - E (2) | ( E ) (3)
E -> E * E (4) | E / E (5) | id    (6)
```

Derivación directa. Denotada por $\Rightarrow$.
Aplicación de exactamente una regla a una cadena para conseguir otra.

- Si $A \to \gamma$ es una producción, entonces $\alpha A \beta \Rightarrow \alpha\gamma\beta$.
- Es decir, $\alpha A \beta$ deriva directamente $\alpha\gamma\beta$, o $\alpha\gamma\beta$ se sigue directamente de $\alpha A \beta$ aplicando exactamente una producción de P.
- Por ejemplo, $E + E \Rightarrow id + E$

Derivación de una cadena

```
E -> E + E (1) | E - E (2) | ( E ) (3)
E -> E * E (4) | E / E (5) | id    (6)
```

Derivación. Denotada por $\Rightarrow^*$, corresponde a **0 o más derivaciones directas consecutivas**.

- Si $\alpha_1, \alpha_2, \ldots, \alpha_m$ son palabras en $(VUT)^*$ y $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$ entonces $\alpha_1 =>^* \alpha_m$
- Es decir, α_m se sigue de α_1 o α_1 deriva α_m aplicando cero o más producciones.
- Por ejemplo, $E + E \Rightarrow^* id + (E)$

Derivaciones más a la izquierda y más a la derecha

En general, en un paso de una derivación puede haber varias variables que sustituir.

```
E -> E + E (1) | E - E (2) | ( E ) (3)
E -> E * E (4) | E / E (5) | id    (6)
```

- En una derivación más a la izquierda siempre elegimos la variables que se encuentra más a la izquierda.

  $E \Rightarrow E + E \Rightarrow (E) + E \Rightarrow (E * E) + E \Rightarrow (id * E) + E \Rightarrow$
  $(id * id) + E \Rightarrow (id * id) + id$

- Si siempre elegimos la variable que está más a la derecha, tendremos una derivación más a la derecha.

  $E \Rightarrow E + E \Rightarrow E + id \Rightarrow (E) + id \Rightarrow (E * E) + id \Rightarrow$
  $(E * id) + id \Rightarrow (id * id) + id$

Sentencias y lenguajes

Sentencia w es una sentencia si $S \Rightarrow^* s$ y $w \in T^*$

Lenguaje generado por una gramática Dada una GLC $G = (V, X, P, S)$, el *lenguaje generado por* G es el conjunto de todas las sentencias que pueden ser derivadas partiendo del axioma.

$$L(G) = \{w \in X^* \mid S \Rightarrow^* w\}.$$

Lenguaje libre de contexto Un lenguaje $L \subseteq X^*$ es libre de contexto (o LLC) si $L = L(G)$ para alguna gramática libre de contexto G.

Ejercicios

1. $G = (V, T, S, P)$ con $V = \{\texttt{E}\}$, $T = \{+, -, *, /, (,), \texttt{id}\}$, $S = \texttt{E}$ y P:

   ```
   E -> E + E (1) | E - E (2) | ( E ) (3) | id (4) | num (5)
   ```

   Escribir una derivación para `id * (id+num) * id`

2. Dar GLCs que generen los siguientes lenguajes sobre el alfabeto $X = \{0, 1\}$

   a) $L_1 = \{0^*1^*\}$

   b) $L_2 = \{0^n 1^n | n \geq 0\}$

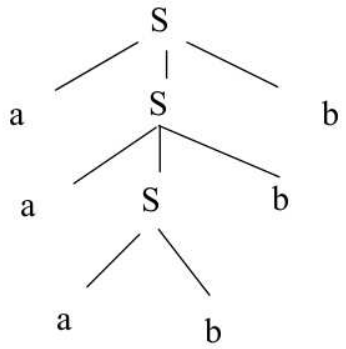
   c) $L_3 = \{w | w = w^R\}$, es decir, w es un palíndromo.

3. Árboles de derivación

Cada derivación puede ser vista como un árbol de derivación, con variables en los nodos internos y terminales en las hojas. Dada una GLC $G = (V, T, P, S)$, un **árbol de derivación** en G es un árbol etiquetado y con raíz tal que:

1. Cada vértice tiene una etiqueta de $V \cup T \cup \{\lambda\}$. Los nodos internos se etiquetan con variables y las hojas con terminales o λ
2. La raíz tiene S como etiqueta.
3. Si un vértice de etiqueta A tiene hijos etiquetados $X_1, X_2, \ldots, X_n$, de izquierda a derecha, entonces $A \to X_1, X_2, \ldots, X_n$ debe ser una producción de P
4. Si un nodo tiene etiqueta λ, entonces debe ser una hoja y ser el único hijo de su padre.

- Para cada palabra $w \in L(G)$, con G una GLC, hay **al menos** un árbol de derivación .
- Consideramos $S \to ab|aSb$
- La derivación de $w = aaabbb$ es $S \to aSb \to aaSbb \to aaabbb$
- Un árbol de derivación para w es



- Y w es el resultado del árbol, es decir, la cadena obtenida al leer sus hojas de izquierda a derecha.

Considerar la GLC de producciones:

```
arithExp ->NUMBER (1) | arithExp PLUS arithExp (2)
| arithExp MULT arithExp (3) | ( arithExp ) (4)
```

- Cada derivación corresponde a un árbol de derivación
  1. Escribir una derivación de

     `NUMBER MULT (NUMBER MULT NUMBER)`
  2. Dibujar el árbol asociado

- Cada árbol de derivación está asociado con una única derivación más a la izquierda y con una única derivación más a la derecha, y viceversa.
  1. Dibujar el árbol de derivación de
     `NUMBER MULT (NUMBER MULT NUMBER)`
  2. Escribir las derivaciones más a la derecha y más a la izquierda correspondientes
- Cada árbol corresponde a una o más derivaciones.

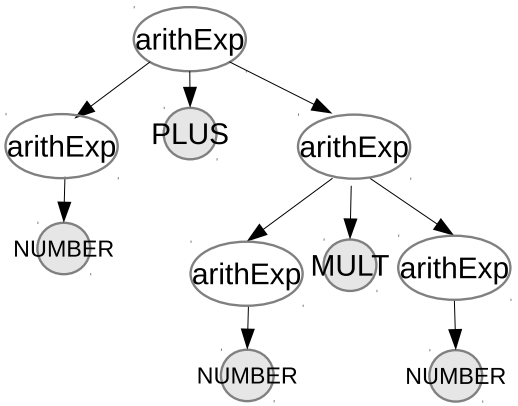
Ambigüedad

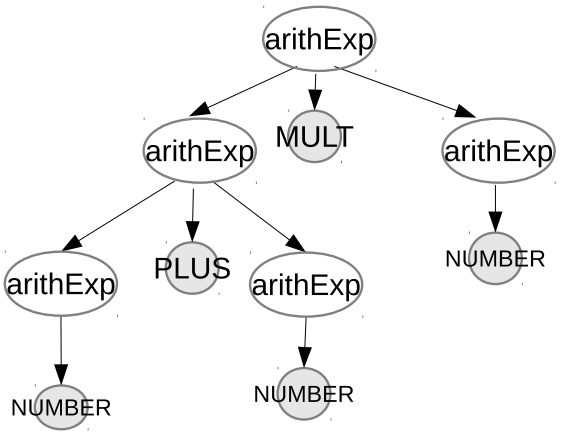
Considerar la GLC de producciones:

```
arithExp ->NUMBER (1) | arithExp PLUS arithExp (2)
| arithExp MULT arithExp (3) | ( arithExp ) (4)
```

- Puede existir una cadena w en $L(G)$ con más de una derivación más a la derecha (o más a la izquierda). Esa cadena también tendrá más de un árbol de derivación.

Por ejemplo: `NUMBER PLUS NUMBER MULT NUMBER`





Gramática ambigua

Una GLC es **ambigua** si existe $w \in T^*$ para la cual **hay al menos dos árboles de derivación** (o dos derivaciones más la izquierda o dos derivaciones más a la derecha).

- Ejemplo: `arithExp ->arithExp PLUS arithExp (1)`
  `| arithExp MULT arithExp (2) | NUMBER (3)`
- G es ambigua porque `NUMBER PLUS NUMBER MULT NUMBER` tiene dos árboles de derivación.
- **Algunas** gramáticas ambiguas pueden tener equivalentes no ambiguas.

Lenguaje inherentemente ambiguo Un LLC es inherentemente ambiguo si todas sus GLCs son ambiguas.

Gramática no ambigua para las expresiones aritméticas

Algunas veces (no siempre), se puede eliminar la ambigüedad añadiendo variables que introducen estructuras adicionales.

Gramática no ambigua para las expresiones aritméticas

- Definimos una variable distinta para cada nivel de precedencia de operadores y una variable extra para las subexpresiones más pequeñas.
- `factor` produce cualquier constante numérica
- `term` es una multiplicación de `factor`
- `arithExp` es una suma de `term`

4. Simplificación de GLCs

Dada una GLC G, queremos encontrar otra GLC G' equivalente pero que sea más *sencilla* en algún sentido. Algunas de las cosas que podemos eliminar:

Símbolos inútiles Aquellos que no se usan en ninguna derivación $S \Rightarrow^* w$, con w una sentencia y S el axioma

λ-producciones Reglas de la forma $A \to \lambda$

Producciones unidad Reglas de la forma $A \to B$, con A y B variables.

Finalmente, queremos escribir cada producción en Forma Normal de Chomsky (FNC), es decir en una de las formas:

$$A \to BC$$

$$A \to a$$

4.1 Limpieza de gramáticas

Símbolos inútiles

- Decimos que una variable es **útil** si aparece en alguna derivación de una palabra del lenguaje. En caso contrario, decimos que es inútil.
- Formalmente, una variable A es útil en una gramática G si tenemos: $S \Rightarrow \ldots \Rightarrow xAy \Rightarrow \ldots \Rightarrow w$, con w una cadena de terminales.
- Un terminal a es útil en una GLC G si alguna cadena de $L(G)$ lo contiene. Si no, a es inútil.
- Una producción es inútil si contiene algún símbolo inútil.

$S -> aSb$

$S -> \lambda$

$S -> A$

$A -> aA$ Inútil

Algunas derivaciones no terminan nunca...

$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \ldots \Rightarrow aa \ldots aA \Rightarrow \ldots$

Por tanto, A no sirve para generar ninguna cadena del lenguaje.

$S -> A$

$A -> aA$

$A -> \lambda$

$B -> bA$ Inútil

B no es accesible desde S

Por tanto, B no sirve para generar ninguna cadena del lenguaje.

Definición de los símbolos útiles

- Una variable A es **viva** si es capaz de generar una cadena de terminales. Formalmente: $A \Rightarrow^* w$, para alguna $w \in T^*$.
  En caso contrario, decimos que A es muerta.

- Una variable o terminal A es **accesible** si aparece en alguna derivación que comienza en S. Formalmente: $S \Rightarrow^* \alpha A \beta$, para $\alpha, \beta \in (V \cup T)^*$

Teorema: Sea $G = (V, T, S, P)$ una GLC con $L(G) \neq \varnothing$. Entonces, existe una gramática equivalente que no tiene símbolos ni producciones inútiles.

Algoritmo para detectar símbolos muertos

- Sea $G = (V, T, P, S)$ una GLC
  1. Marcar una variable A como viva si tiene una producción $A \to w$ donde w es una cadena que no tiene variables.
  2. Marcar una variable A como viva si tiene una producción $A \to w$ donde w es una cadena cuyas variables han sido todas marcadas como vivas.
  3. Repetir 2 hasta que ya no se marquen más variables como vivas.
- Todas aquellas variables que no han sido marcadas como *vivas* son muertas.

1. Encontrar los símbolos muertos de la siguiente gramática

   $S \to aaB \mid ABC \mid AaE \mid \lambda$

   $A \to bb \mid aA \mid BaB \mid D$

   $B \to aD \mid bD$

   $C \to a \mid bD$

   $D \to DD \mid ED$

   $E \to a \mid b$

2. Encontrar los símbolos muertos de la siguiente gramática

   $S \to aS \mid A \mid CD$

   $A \to a$

   $B \to aa$

   $C \to aCb$

   $D \to aDbb \mid add$

Algoritmo para encontrar una GLC equivalente sin variables muertas

$G = (V, T, S, P) \rightarrow G' = (V', T, S, P')$

1. Encontrar las variables muertas de V. El nuevo conjunto de variables V' contendrá sólo las variables vivas de V.
2. P': Eliminar de P todas las producciones que contengan a alguna variable muerta.

Algoritmo para encontrar los símbolos inaccesibles

- Consideramos una GLC $G = (V, T, P, S)$
  1. S es accesible
  2. Si A es accesible y $A \to \alpha$, entonces también son accesibles todos los símbolos (variables y terminales) que aparecen en α.
  3. Repetir el paso 2 hasta que no se añadan más variables accesibles.
- Todos los símbolos no marcados como accesibles son inaccesibles.

Algoritmo para obtener una GLC equivalente sin símbolos inaccesibles

$G = (V, T, S, P) \rightarrow G' = (V', T', S, P')$

1. Encontrar los símbolos inaccesibles
2. V' contendrá todas las variables de V excepto las inaccesibles
3. T' contendrá todos los terminales de T excepto los inaccesibles
4. P' contendrá todas las producciones de P excepto aquellas que tengan variables o terminales inaccesibles.

Algoritmo para obtener una GLC equivalente sin símbolos inútiles

- Una GLC carece de símbolos inútiles si y sólo todos sus símbolos son vivos y accesibles
- Por tanto, podemos eliminar los símbolos inútiles con el siguiente procedimiento:
  1. Eliminar símbolos muertos
  2. Eliminar símbolos inaccesibles

- Es importante realizar el proceso en el orden indicado, porque si lo hiciéramos al revés podría no funcionar.
- Ejemplo:

$$S \to AB \mid a$$

$$A \to aA$$

$$B \to b$$

- Aquí A es muerta y tras eliminarla (junto con la producción $S \to AB$) la variable B se vuelve inaccesible y, por tanto, inútil.
- Sin embargo, si comprobamos en primer lugar la accesibilidad, no detectaremos ninguna variable inaccesible. Al eliminar después las variables muertas, en la GLC quedaría B, que es inútil.

Ejercicios

Encontrar GLCs equivalentes sin símbolos inútiles

1. $S \to aaB \mid ABC \mid AaE \mid \lambda$
   $A \to bb \mid aA \mid BaB \mid D$
   $B \to aD \mid bD$
   $C \to a \mid bD$
   $D \to DD \mid ED$
   $E \to a \mid b$

2. $S \to aS \mid A \mid CD$
   $A \to a$
   $B \to aa$
   $C \to aCb$
   $D \to aDbb \mid add$

3. $S \to Bb \mid SaS \mid \lambda$
   $A \to BB \mid AS \mid Dbb$
   $B \to D \mid a \mid AA$
   $C \to AS \mid cCC \mid c$
   $D \to abB$
   $E \to A \mid \lambda$

Eliminación de λ-producciones

- Una λ-producción es una producción de la forma $A \to \lambda$.
- Para eliminar las λ-producciones, primero tenemos que encontrar las **variables anulables**, es decir, variables A tales que $A \Rightarrow^* \lambda$.
- **Algoritmo**

1. Si existe una producción $A \to \lambda$, marcamos A como anulable.
2. Si existe una producción $A \to \alpha$ y todos los símbolos de α son anulables, entonces A marcamos como anulable.

Es decir: una variable A es anulable si podemos derivar la palabra vacía a partir de A.

Teorema: Si L es un LLC, entonces $L - \{\lambda\}$ puede ser generado por una GLC sin λ-producciones.

Cuidado:

- Si λ está en L, entonces necesitaremos $S \to \lambda$ para generarla.
- El algoritmo que describiremos servirá para generar $L - \{\lambda\}$.

Algoritmo para eliminar λ-producciones

Consideramos una GLC $G = (V, T, S, P)$. Si $\lambda \notin L(G)$, entonces podemos encontrar $G' = (V, T, S', P')$ una GLC equivalente **sin** λ-producciones. La hallaremos con el siguiente algoritmo:

1. Encontrar las **variables anulables**.
   $S \to aAB$
   $A \to aAA \mid \lambda$
   $B \to bBB \mid \lambda$
   - A y B son ambas anulables porque $A \to \lambda$ y $B \to \lambda$

2. Añadimos al nuevo conjunto de producciones (P') todas las producciones de P salvo las λ-producciones.
   - Añadimos a P':
     $S \to aAB$
     $A \to aAA$
     $B \to bBB$

3. Para cada producción $A \to \alpha B \beta$ de P' con B anulable, añadimos a P' la producción $A \to \alpha\beta$ (excepto si se trata de una λ-producción).

4. Repetimos el paso 3 hasta que no se añadan nuevas producciones.
   - Para $S \to aAB$
     - Como A es anulable, añadimos a P' $S \to aB$
     - Como B es anulable, añadimos a P' $S \to aA$
   - Para $S \to aB$
     - Como B es anulable, añadimos a P' $S \to a$
   - Para $S \to aA$
     - Como A es anulable, añadimos a P' $S \to a$

- Para $A \to aAA$
  - Como A es anulable, añadimos a P' $A \to aA$
- Para $B \to bBB$
  - Como B es anulable, añadimos a P' $B \to bB$
- Para $A \to aA$
  - Como A es anulable, añadimos a P' $A \to a$
- Para $B \to bB$
  - Como B es anulable, añadimos a P' $B \to b$

La GLC resultante será:

$$S \to aAB \mid aA \mid aB \mid a$$

$$A \to aAA \mid aA \mid a$$

$$B \to bBB \mid bB \mid b$$

Producciones unidad

Una producción unidad es una producción de la forma $A \to B$ donde A y B son variables.

Teorema: Si $G = (V, T, P, S)$ es una GLC, entonces existe una GLC equivalente que no tiene producciones unidad.

Algoritmo para eliminar las producciones unidad

1. Para cada producción unidad $A \to B$, buscar las producciones $B \to \alpha_1 | \alpha_2 | \cdots \alpha_n$, y reemplazar $A \to B$ con $A \to \alpha_1 | \alpha_2 | \cdots \alpha_n$
2. Eliminar cualquier producción de la forma $A \to A$ que aparezca.
3. Repetir 1 y 2 hasta que no queden producciones unidad.

Ejemplo:

$$S \to S + T \mid T$$

$$T \to T * F \mid F$$

$$F \to (S) \mid a$$

Limpieza de GLCs

Teorema: Sea L un LLC que no contiene λ. Entonces, existe existe una GLC para L que no contiene símbolos inútiles, producciones unidad ni λ-produccciones.

Demostración:

1. Eliminar λ-producciones
2. Eliminar producciones unidad
3. Eliminar símbolos inútiles

El orden es muy importante:

- Eliminar λ-producciones puede crear producciones unidad y variables muertas

  $S \to AB$ $\qquad$ $A \to \lambda$ $\qquad$ $B \to b$

- Eliminar producciones unidad puede dar lugar a símbolos inaccesibles

  $S \to A$ $\qquad$ $A \to a$

- Eliminar variables muertas puede crear símbolos inaccesibles.

  $S \to AB$ $\qquad$ $A \to a$ $\qquad$ $B \to BB$

Orden recomendado:

1. **Eliminar símbolos muertos**
2. **Eliminar símbolos inaccesibles**
3. Eliminar λ-producciones
4. Eliminar producciones unidad
5. Eliminar símbolos muertos
6. Eliminar símbolos inaccesibles

Los pasos 1 y 2 no son estrictamente necesarios, pero suelen ahorrar tiempo en la aplicación del resto de métodos.

Ejemplos

Limpiar las siguientes GLCs:

1. $S \to ABaC$
   $A \to BC$
   $B \to b \mid \lambda$
   $C \to D \mid \lambda$
   $D \to d$

2. $S \to Aa \mid B$
   $B \to A \mid bb$
   $A \to a \mid bc \mid B$

3. $S \to 0A \mid 1AB$
   $A \to 10A \mid AA \mid \lambda$
   $B \to AA \mid 1S \mid 0$

4. $S \to 0A \mid 1B$
   $A \to 10AB \mid BB$
   $B \to BAB \mid 1 \mid \lambda$
   $C \to AB \mid SC\ 0$

5. $S \to Dbh \mid D$
   $A \to aaC$
   $B \to Sf \mid ggg$
   $C \to cA \mid d\,C$
   $D \to E \mid SABC$
   $E \to be$

4.2 Forma normal de Chomsky (FNC)

- Algunos algoritmos requieren que las producciones se encuentren en un formato especial, lo que se llama una **forma normal**.
- El ejemplo principal es la Forma normal de Chomsky (FNC).
- La FNC requiere que toda producción sea de una de las dos siguientes formas:
  - $A \to BC$, con A, B y C variables
  - $A \to a$, con A una variable y a un terminal

La GLC

$$S \to AS \mid a$$

$$A \to SA \mid b$$

está en FNC.

Pero la GLC

$$S \to AS \mid AAS$$

$$A \to SA \mid aa$$

no está en FNC.

Teorema: Cualquier GLC que no genera λ puede ser transformada en una GLC equivalente en FNC.

Algoritmo

1. Eliminar λ-producciones
2. Eliminar producciones unidad
3. Eliminar símbolos inútiles (no es estrictamente necesario, pero sí conveniente)

Tras estos tres pasos, cualquier producción de la forma $A \to x_1$ ya está en CNF porque x_1 ha de ser un terminal (no hay producciones unidad).

4. Consideremos las producciones de la forma
   $A \to x_1 x_2 \cdots x_m$ con $m \geq 2$
   - Si x_i es un terminal a, creamos una nueva variable X_a, añadimos una nueva producción $X_a \to a$ y reemplazamos a por X_a
   - Repetimos con cada producción y terminal en esas condiciones

Ejemplo: $S \to AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b$
$A \to aAA \mid aA \mid a$
$B \to bBB \mid bB \mid b$
Nótese que todos los símbolos son útiles y que no hay λ-producciones ni producciones unidad.

- Definimos nuevas producciones $X_a \to a$ y $X_b \to b$ y reemplazamos
  $S \to AB \mid \underline{aAA} \mid \underline{aA} \mid a \mid \underline{bBB} \mid \underline{bB} \mid b$
  $A \to \underline{aAA} \mid \underline{aA} \mid a$
  $B \to \underline{bBB} \mid \underline{bB} \mid b$

- Quedaría:
  $S \to AB \mid X_aAA \mid X_aA \mid a \mid X_bBB \mid X_bB \mid b$
  $A \to X_aAA \mid X_aA \mid a$
  $B \to X_bBB \mid X_bB \mid b$
  $X_a \to a\ X_b \to b$

Tras los 4 primeros pasos, todas las producciones son:

- $A \to a$ con A variable y a terminal
- $A \to x_1 x_2 \cdots x_m$ con $m \geq 2$ y cada x_i es una variable.

5. Si $m = 2$, la producción ya está en FNC.
6. Cada producción $A \to x_1 x_2 \cdots x_m$ con $m > 2$ se sustituye por una serie de producciones cada una en FNC. Por ejemplo, $A \to BCDBCE$ se sustituye por:

$$A \to BY_1$$

$$Y_1 \to CY_2$$

$$Y_2 \to DY_3$$

$$Y_3 \to BY_4$$

$$Y_4 \to CE$$

- En el ejemplo que teníamos

$S \to AB \mid \underline{X_aAA} \mid X_aA \mid a \mid \underline{X_bBB} \mid X_bB \mid b$
$A \to \underline{X_aAA} \mid X_aA \mid a$
$B \to \underline{X_bBB} \mid X_bB \mid b$
$X_a \to a$
$X_b \to b$

- Añadimos las producciones

$Y_1 \to AA$
$Y_2 \to BB$

- Y el resultado final sería:

$$S \to AB \mid X_aY_1 \mid X_aA \mid a \mid X_bY_2 \mid X_bB \mid b$$
$$A \to X_aY_1 \mid X_aA \mid a$$
$$B \to X_bY_2 \mid X_bB \mid b$$
$$X_a \to a$$
$$X_b \to b$$
$$Y_1 \to AA$$
$$Y_2 \to BB$$

Ejemplos

1. Encontrar una GLC en FNC equivalente a:

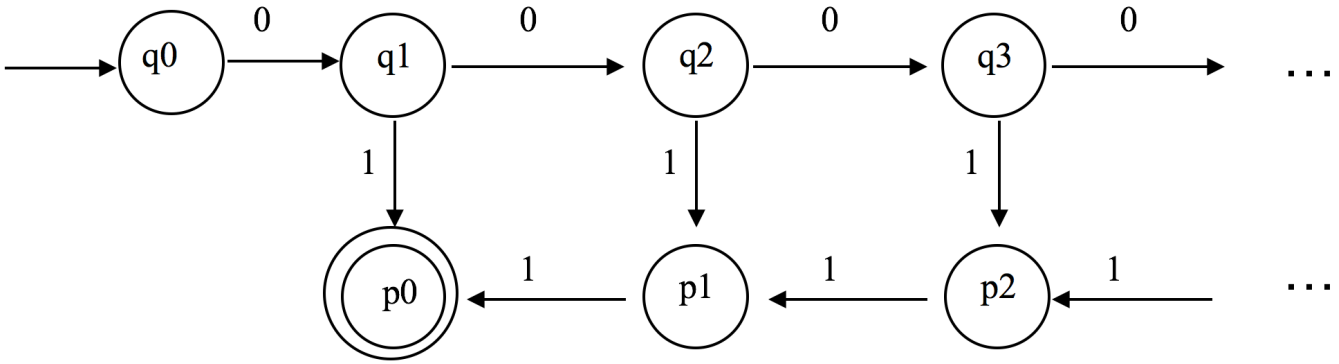
$$S \to bA \mid aB \mid ABaS$$

$$A \to bAA \mid aS \mid a \mid ab$$

$$B \to aBB \mid bS \mid b \mid aBa$$

2. Dar una GLC en FNC que genere $L = \{a^n b^n : n > 0\}$.

5. AUTÓMATAS CON PILA



¿Por qué no podemos reconocer las palabras de la forma $0^n 1^n$ con un autómata finito?

- Necesitamos almacenar el número de ceros para compararlo con el de unos
- Sólo podemos distinguir un **número finito** de situaciones, pero la longitud de las palabras no está acotada

Conclusión: Los autómatas finitos no son capaces de reconocer las palabras de la forma $0^n 1^n$

¿Cómo podríamos solucionar el problema?

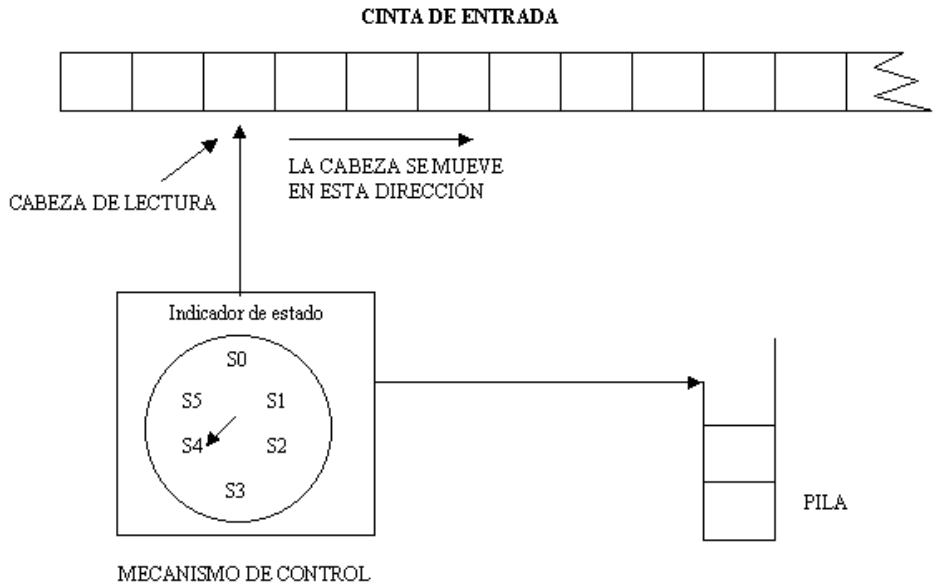
- Añadiendo un dispositivo de almacenamiento de capacidad no acotada

¿Cuál es el dispositivo más sencillo de este tipo?

- Una pila (estructura LIFO).

Descripción informal de los autómatas con pila (AP)

- Los AP tienen una **cinta** de entrada dividida en casillas en la que se coloca la palabra a analizar. Se lee un símbolo de cada vez, de izquierda a derecha.
- La **unidad de control** del AP es una máquina de estados finitos.
- También tienen una **pila** de la que se puede **sacar** o en la que se puede (**meter**) información, siempre por la parte superior.



- Los AP son máquinas que reconocen LLCs.
- Dada una GLC, se puede construir un AP que reconoce el lenguaje generado por la gramática (y viceversa).

5.1 Funcionamiento de los AP

Ejemplo

Veamos con un ejemplo como podríamos reconocer el lenguaje $\{a^i\ b^i : i \geq 0\}$. Procesaríamos cada palabra del siguiente modo:

1. Por cada a que se lee en la cinta se añade una A en la pila.
2. Cuando se llega a las b's se saca una A de la pila por cada b que se lea.

Se aceptan exactamente aquellas cadenas que hacen que la pila se vacíe al leer la última b.

5.2 Definición formal

Un AP M es una séptupla $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ donde

- Q es un conjunto finito no vacío de **estados**.
- Σ es el **alfabeto de entrada** (conjunto finito y no vacío de símbolos).
- Γ es el **alfabeto de la pila** (conjunto finito y no vacío de símbolos).
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ es la **función de transición**
- $q_0 \in Q$ es el **estado inicial**
- $Z \in \Gamma$ es el **símbolo inicial de la pila**
- $F \subseteq Q$ es el conjunto de **estados de aceptación** (podría ser vacío)

Un AP realiza **transiciones** en función del **símbolo de entrada**, el **estado** actual y el símbolo de la **cima de la pila**.

- Puede usar o no el símbolo de la cinta:

  Dependiente de la entrada Lee el símbolo de la cinta y avanza hacia la derecha.

  $\delta : state \times input \times stack \rightarrow state \times stack^*$

  Independiente de la entrada Semejante a una (λ-transición) en un AF. No lee símbolo y no avanza en la cinta.

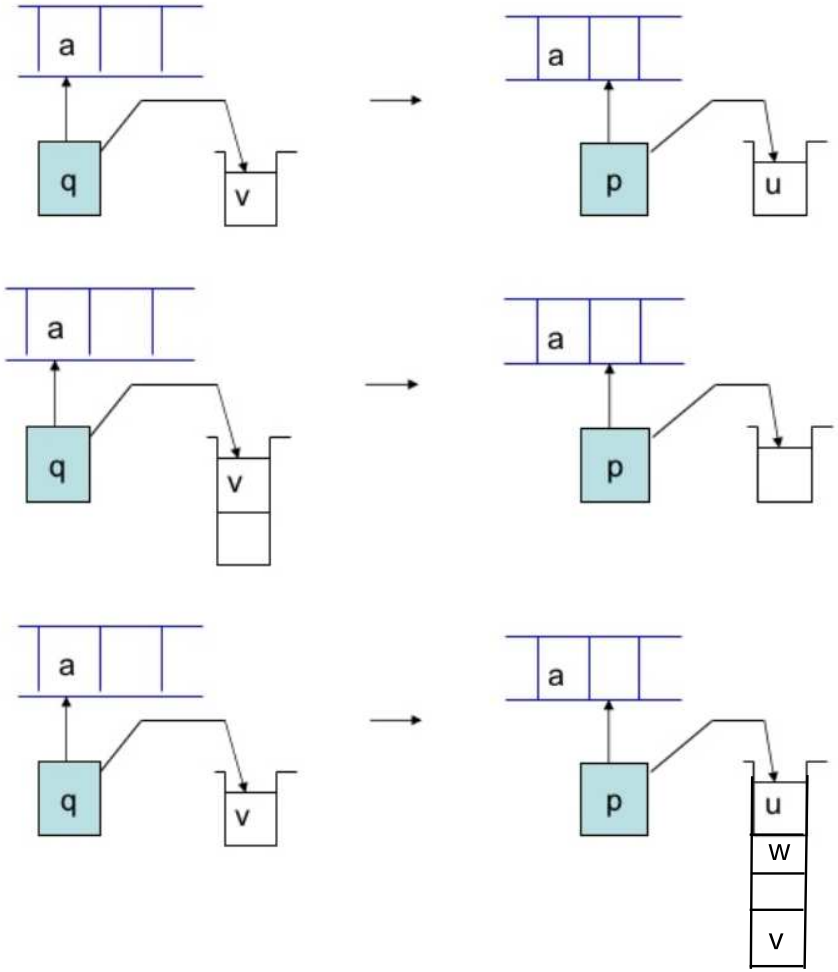
  $\delta : state \times \{\lambda\} \times stack \rightarrow state \times stack^*$

Combinando ambas opciones:

$$f : state \times (input \cup \{\lambda\}) \times stack \rightarrow state \times stack^*$$

En una **transición** un AP puede realizar lo siguiente:

1. Cambiar de estado (podría ser el mismo que el anterior).
2. Reemplazar el símbolo de la cima de la pila por otros símbolos:
   - Por λ, que equivale a **sacar** el símbolo de la pila.
   - Por el mismo símbolo que había (equivale a no hacer nada).
   - Por uno o más símbolos iguales o distintos


a
q
v
a
p
u
a
q
v
a
p
a
q
v
a
p
u
w
v

5.3 Computación

Configuración instantánea

La **configuración instantánea** de un AP en un momento del procesamiento de una cadena es la tupla
$(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ donde

- q es el **estado actual**
- w es la parte de palabra que queda por procesar
  - el primer símbolo de w es el que se encuentra bajo la cabeza lectora
  - Si $w = \lambda$, entonces la palabra ha sido leída completamente

- α es el contenido actual de la pila
  - el primer símbolo de α es el que se encuentra en la cima de la pila
  - Si $\alpha = \lambda$ entonces la pila se ha vaciado
- Si w es la cadena de entrada, la **configuración inicial** es (q_0, w, Z), siendo q_0 el estado inicial y Z el símbolo inicial de la pila.

Un **movimiento** o transición de M

$$(q, aw, A\alpha) \vdash (q', w, \gamma\alpha)$$

se produce si

$$(q', \gamma) \in \delta(q, a, A)$$

con $q' \in Q$, $\gamma \in \Gamma^*$, $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $A \in \Gamma$

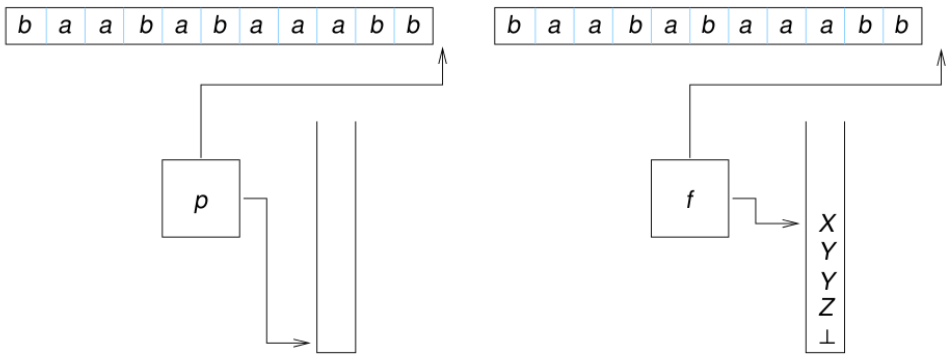
Utilizaremos $\vdash^*$ para representar cero o más transiciones o movimientos.

5.4 Aceptación

Un AP puede funcionar de dos modos distintos:

Por vaciado de pila Se aceptan las palabras que se consumen totalmente y vacían la pila: $(q, \lambda, \lambda), q \in Q$

Por estados finales Se aceptan las palabras que llevan al AP a un estado final: $(q, \lambda, \alpha), q \in F, \alpha \in \Gamma^*$



Vaciado de pila Estados

Si $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, \emptyset)$ es un AP, entonces su

lenguaje aceptado por vaciado de pila es

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z) \vdash^* (p, \lambda, \lambda) \text{ con } p \in Q\}$$

lenguaje aceptado por estados finales es

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, Z) \vdash^* (p, \lambda, \alpha) \text{ con } p \in F \text{ y } \alpha \in \Gamma^*\}$$

5.5. Ejemplo de AP

AP que reconoce el lenguaje $\{a^n b^n : n \geq 1\}$ por vaciado de pila

- La pila no tiene un tamaño acotado
  - Siempre podemos añadir un símbolo más
  - Gracias a ello, el AP puede contar hasta números arbitrariamente grandes
- Funcionamiento del AP
  - Por cada a que se lee en la cinta se añade una A en la pila.
  - Cuando se llega a las b's se saca una A de la pila por cada b que se lea.
  - Se aceptan exactamente aquellas cadenas que hacen que la pila se vacíe al leer la última b.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$
$$Q = \{q_0, q_1\}$$
$$\Sigma = \{a, b\}$$
$$\Gamma = \{A, Z_0\}$$
$$F = \emptyset$$

δ definida por:

$$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$$
$$\delta(q_0, a, A) = \{(q_0, AA)\}$$
$$\delta(q_0, b, A) = \{(q_1, \lambda)\}$$
$$\delta(q_1, b, A) = \{(q_1, \lambda)\}$$
$$\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$$

Configuración inicial $(q_0, aaabbb, Z_0)$

$\vdash (q_0, aaabbb, Z_0)$ // $\delta(q_0, a, Z_0) \to (q_0, AZ_0)$

$\vdash (q_0, aabbb, AZ_0)$ // $\delta(q_0, a, A) \to (q_0, AA)$

$\vdash (q_0, abbb, AAZ_0)$ // $\delta(q_0, a, A) \to (q_0, AA)$

$\vdash (q_0, bbb, AAAZ_0)$ // $\delta(q_0, b, A) \to (q_1, \lambda)$

$\vdash (q_1, bb, AAZ_0)$ // $\delta(q_1, b, A) \to (q_1, \lambda)$

$\vdash (q_1, b, AZ_0)$ // $\delta(q_1, b, A) \to (q_1, \lambda)$

$\vdash (q_1, \lambda, Z_0)$ // $\delta(q_1, \lambda, Z_0) \to (q_1, \lambda)$

$\vdash (q_1, \lambda, \lambda)$ //Pila vacía - Se acepta la palabra

Configuración inicial $(q_0, aabbb, Z_0)$

$\vdash (q_0, aabbb, Z_0)$ // $\delta(q_0, a, Z_0) \to (q_0, AZ_0)$

$\vdash (q_0, abbb, AZ_0)$ // $\delta(q_0, a, A) \to (q_0, AA)$

$\vdash (q_0, bbb, AAZ_0)$ // $\delta(q_0, b, A) \to (q_1, \lambda)$

$\vdash (q_1, bb, AZ_0)$ // $\delta(q_1, b, A) \to (q_1, \lambda)$

$\vdash (q_1, b, Z_0)$ // $\delta(q_1, \lambda, Z_0) \to (q_1, \lambda)$

$\vdash (q_1, b, \lambda)$ // No hay más movimientos - Se rechaza la palabra

AP deterministas)

Un AP $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es **determinista** si y sólo si cumple las dos siguientes condiciones

1. Si $\delta(q, \lambda, A) \neq \emptyset$ entonces $\delta(q, a, A) = \emptyset$
   Es decir:
   - Se puede realizar una λ-transición **sólo** si no hay otro tipo de movimiento para la misma situación.
   - **NO** se cumple en el siguiente ejemplo:

$$\delta(q, b, A) = \{(q, AA)\}$$
$$\delta(q, \lambda, A) = \{(q, S)\}$$

2. $|\delta(q, a, A)| \leq 1$ para cada q, a y A.
   - Hay **a lo sumo** una transición para cada posible situación.
   - **NO** se cumple en el siguiente ejemplo:

$$\delta(q, a, S) = \{(q, A), (q, AS)\}$$

Ejemplo de AP determinista

$L_1 = \{z2z^R \mid z \in \{0,1\}^*\}$

$P = (\{0,1,2\}, \{Z,C,U\}, \{p,q\}, Z, p, f, \emptyset)$ con f dada por:

1. **Primera mitad de la cadena**

   a) El primer símbolo puede ser 0, 1 ó 2

$$f(p,0,Z) = \{(p,CZ)\}$$
$$f(p,1,Z) = \{(p,UZ)\}$$
$$f(p,2,Z) = \{(q,Z)\}$$

   b) Si el símbolo es igual que el anterior

$$f(p,0,C) = \{(p,CC)\}$$
$$f(p,1,U) = \{(p,UU)\}$$

c) Si el símbolo es diferente al anterior

$$f(p, 0, U) = \{(p, CU)\}$$
$$f(p, 1, C) = \{(p, UC)\}$$

2. **Mitad de la cadena**. El símbolo anterior puede ser 0 ó 1.

$$f(p, 2, U) = \{(q, U)\}$$
$$f(p, 2, C) = \{(q, C)\}$$

3. **Segunda parte de la cadena**. Cada símbolo debe corresponder debe corresponderse con su simétrico con respecto al centro

$$f(q, 1, U) = \{(q, \lambda)\}$$
$$f(q, 0, C) = \{(q, \lambda)\}$$

4. **Vaciamos la pila**.

$$f(q, \lambda, Z) = \{(q, \lambda)\}$$

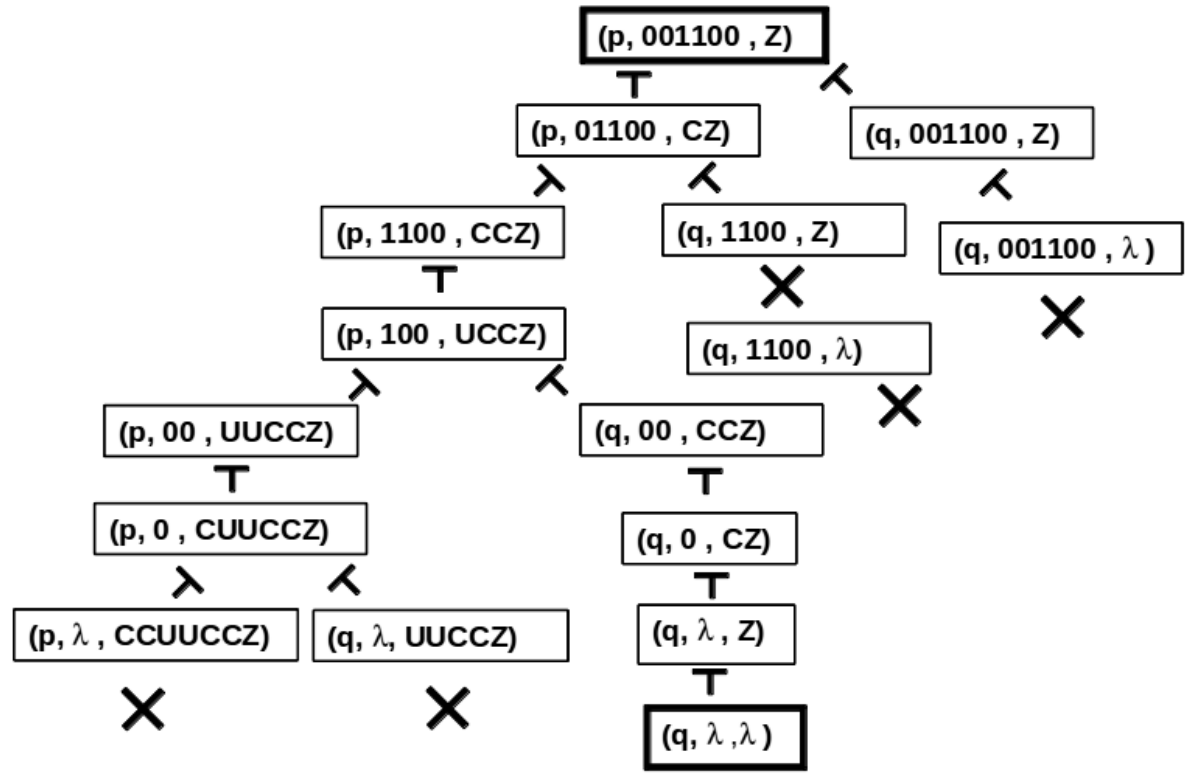
Ejemplo de AP no determinista

$L_2 = \{zz^R \mid z \in \{0,1\}^*\}$

- Nos falta el marcador de mitad de palabra
- No sabemos cuándo tenemos que dejar de apilar y comenzar a desapilar
- Un AP determinista no podría aceptar, por vaciado de pila, la palabra 11 y también la palabra 110011.

$P = (\{0,1\}, \{Z, C, U\}, \{p, q\}, Z, p, f, \emptyset)$ con f dada por:

$$f(p, 0, Z) = \{(p, CZ)\}$$
$$f(p, 1, Z) = \{(p, UZ)\}$$
$$f(p, \lambda, Z) = \{(q, \lambda)\}$$
$$f(p, 1, Z) = \{(p, UZ)\}$$
$$f(p, 0, U) = \{(p, CU)\}$$
$$f(p, 1, C) = \{(p, UC)\}$$
$$f(p, 0, C) = \{(p, CC), (q, \lambda)\}$$
$$f(p, 1, U) = \{(p, UU), (q, \lambda)\}$$
$$f(q, 0, C) = \{(q, \lambda)\}$$
$$f(q, 1, U) = \{(q, \lambda)\}$$
$$f(q, \lambda, Z) = \{q, \lambda\}$$


(p, 001100 , Z)
(p, 01100 , CZ)
(q, 001100 , Z)
(p, 1100 , CCZ)
(q, 1100 , Z)
(q, 001100 , λ)
(p, 100 , UCCZ)
(q, 1100 , λ)
(p, 00 , UUCCZ)
(q, 00 , CCZ)
(p, 0 , CUUCCZ)
(q, 0 , CZ)
(p, λ , CCUUCCZ)
(q, λ, UUCCZ)
(q, λ , Z)
(q, λ ,λ)

Capacidad expresiva de los APs deterministas y no deterministas

APs no deterministas :

- Reconocen TODOS los LLCs
- Con backtracking
- Ineficientes

APs deterministas :

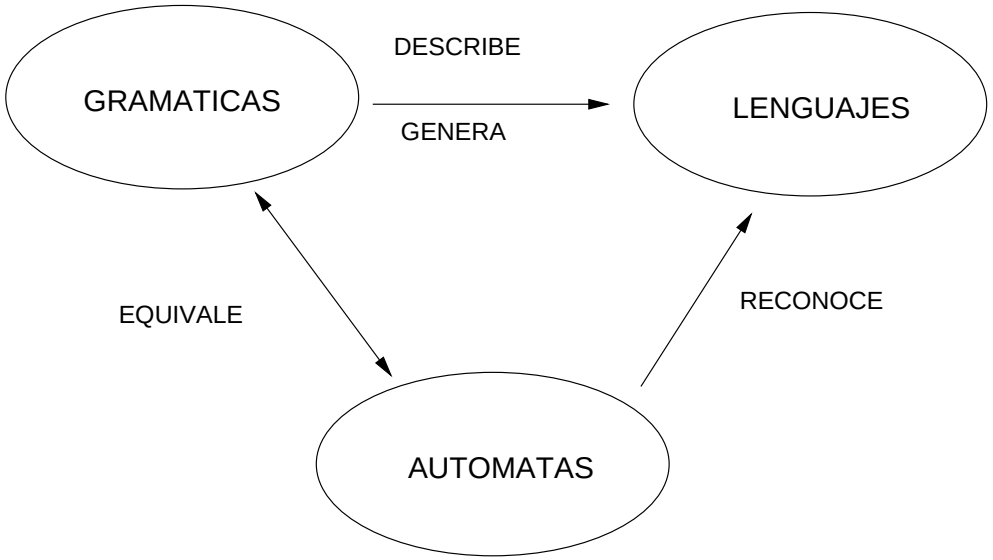
- Un subconjunto de los LLCs (y todos los LRs)
- Sin backtracking
- Eficientes

El modelo equivalente a las GLCs son los APs no deterministas (o, simplemente, APs).

Teorema: El conjunto de lenguajes aceptados por APs funcionando por estados finales es igual al conjunto de lenguajes aceptados por APs por vaciado de pila.

Teorema: Para cada CFL G existe un AP A tal que $L(G) = L(A)$ y viceversa.

Teorema: El conjunto de lenguajes aceptados por APs es igual al conjunto de todos los LLCs.



7. Construcción de un AP a partir de una GLC

Posibilidades

- Directamente (lo que hemos estado haciendo)
- A partir de una GLC (análisis sintático - CUP)

Teorema Para cada CFL G existe un AP A tal que $L(G) = L(A)$.

Idea Reproducir derivaciones más a la izquierda en la pila del AP.

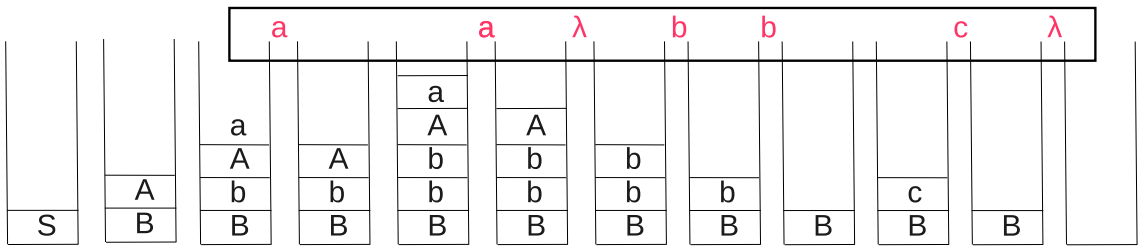
Ejemplo: $S \to AB, A \to aAb \mid \lambda, B \to cB \mid \lambda$

- $L(G) = \{w \in \{a,b,c\}^* \mid w = a^i b^i c^j\}$
- Derivación más a la izquierda para $aabbc$:
  $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aaAbbB \Rightarrow aabbB \Rightarrow aabbcB \Rightarrow aabbc$

Funcionamiento informal

1. Usar el axioma como símbolo inicial de la pila.
2. Usar las producciones de la GLC para sustituir (no determinísticamente) las variables de la cima de la pila.
3. Si en la cima de la pila hay un terminal, compararlo con el contenido de la cinta.

- Derivación más a la izquierda para $aabbc$:
  $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aaAbbB \Rightarrow aabbB \Rightarrow aabbcB \Rightarrow aabbc$
- En la pila:



Algoritmo formal

Dada una GLC $G = (V, T, S, P)$, construimos un AP equivalente del siguiente modo:

$$P = (\Sigma, \Gamma, Q, Z, q_0, f, \emptyset)$$

$\Sigma = T$, $\Gamma = V \cup T$, $Q = \{q\}$, $Z = S$, $q_0 = q$

con f dada por:

$$\begin{aligned} f(q, a, a) &= \{(q, \lambda)\} \\ f(q, \lambda, A) &= \{(q, \alpha) : A \to \alpha \in P\} \end{aligned}$$

Ejemplo

Consideremos la GLC (V, T, S, P) con $V = \{E, T, F\}$,
$T = \{id, +, *, [,]\}$, $S = E$
$P = \{E \to E + T \mid T, T \to T * F \mid F, F \to [E] \mid id\}$

El AP equivalente tendrá f dada por:

$f(q, id, id) = \{(q, \lambda)\}$ $\quad$ $f(q, +, +) = \{(q, \lambda)\}$
$f(q, *, *) = \{(q, \lambda)\}$ $\quad$ $f(q, [, [) = \{(q, \lambda)\}$
$f(q, \lambda, E) = \{(q, E + T), (q, T)\}$ $\quad$ $f(q, \lambda, T) = \{(q, T * F), (q, F)\}$
$f(q, \lambda, F) = \{(q, [E]), (q, id)\}$ $\quad$ $f(q,],]) = \{(q, \lambda)\}$

Ejercicio: Procesar las siguientes palabras

- $id * [id + id]$
- $[id]$
- $[id * [id + id]]$

8. Propiedades de los lenguajes libres de contexto

Teorema: Si L_1 y L_2 son LLCs entonces también lo son $L_1 \cup L_2$, $L_1 L_2$ y L_1^*. Además, si L_3 es un lenguaje regular entonces $L_1 \cap L_3$ es LLC.

Sin embargo, los LLCs no son cerrados para:

Intersección La intersección de dos LLCs no siempre es un LLC

Complemento El complementario de un LLC no siempre es un LLC

Unión

Teorema: Consideramos dos LLCs, L_1 reconocido por $G_1 = (V_1, T_1, S_1, P_1)$ y L_2 reconocido por $G_2 = (V_2, T_2, S_2, P_2)$. Entonces, $L_1 \cup L_2$ es un lenguaje libre de contexto.

Demostración:

- Renombramos variables para asegurarnos de que las GLCs no tienen ninguna en común ($V_1 \cap V_2 = \emptyset$) y que $S \notin V_1 \cup V_2$.
- $V = V_1 \cup V_2 \cup \{S\}$
- $T = T_1 \cup T_2$
- $P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$

Concatenación

Teorema: Consideramos dos LLCs, L_1 reconocido por $G_1 = (V_1, T_1, S_1, P_1)$ y L_2 reconocido por $G_2 = (V_2, T_2, S_2, P_2)$. Entonces, $L_1 L_2$ es un lenguaje libre de contexto.

Demostración:

- Renombramos variables para asegurarnos de que las GLCs no tienen ninguna en común ($V_1 \cap V_2 = \emptyset$) y que $S \notin V_1 \cup V_2$.
- $V = V_1 \cup V_2 \cup \{S\}$
- $T = T_1 \cup T_2$
- $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$

Clausura de Kleene

Teorema: Consideramos un LLC L_1 reconocido por $G_1 = (V_1, T_1, S_1, P_1)$. Entonces, L_1^* es un lenguaje libre de contexto.

Proof:

- Renombramos variables para asegurarnos de que $S \notin V_1$.
- $V = V_1 \cup \{S\}$
- $T = T_1$
- $P = P_1 \cup \{S \to S_1 S \mid \lambda\}$

PROPIEDADES DE DECISIÓN DE LOS LLCS

- Existen algoritmos para decidir si:
  1. Un LLC dado es vacío
  2. Un LLC dado es infinito
  3. Una cadena es generada por una GLC dada
- Sin embargo, no existen algoritmos para resolver los siguientes problemas:
  - Dadas dos GLCs, ¿generan el mismo lenguaje?
  - Dado un LLC, ¿es su complemento también LLC?
  - ¿Es ambigua una GLC dada?

Comprobar si un LLC es vacío

- Ya lo hemos comprobado, indirectamente
- Sabemos cómo detectar y eliminar variables muertas
- Si el axioma es muerto, entonces el lenguaje generado por la GLC es vacío. En caso contrario, no lo es.

Comprobar si un LLC es infinito

Dada una GLC construir un grafo como sigue:

1. Convertir la GLC a FNC **sin símbolos inútiles** (la palabra vacía podría perderse en el proceso, pero eso no afecta a la infinitud del lenguaje).
2. Tomar cada variable como un nodo del grafo
3. Añadir un eje de A a B si hay una producción de la forma $A \to BC$ o $A \to CB$

El LLC generado por la gramática es infinito si el grafo anterior **tiene algún ciclo** y finito en caso contrario.

¿Son infinitos los lenguajes generados por las siguientes GLCs?

1. $S \to AB \mid CC$
   $A \to CC|a$
   $B \to b|CC$
   $C \to AS$

2. $S \to AA \mid BB$
   $A \to BB|a$
   $B \to b$

3. $S \to AB$
   $A \to CC|a$
   $B \to b|CC$
   $C \to CC$

COMPROBAR SI UNA CADENA ES GENERADA POR UNA GLC

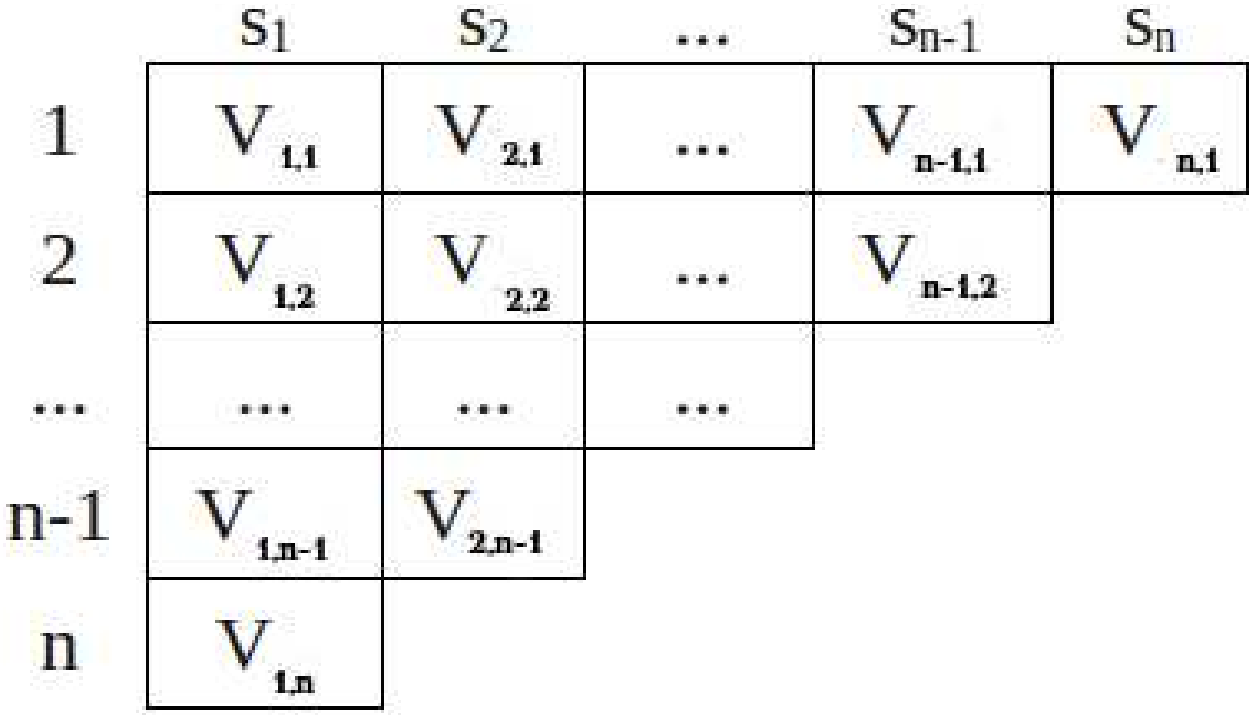
- Queremos saber si una cadena w está en $L(G)$.
- **Partimos de una GLC en FNC**. Si no, la convertimos.
- $w = \lambda$ es un caso especial que puede ser resuelto comprobando si el axioma es anulable.
- Podemos construir un AP para comprobar si w es generada por la GLC, pero es un método poco eficiente (se generan todas las posibles derivaciones).
- El algoritmo de Cocke-Younger-Kasami (CYK) resuelve el problema usando una **tabla**.

- Dada una cadena w de longitud n, construimos una tabla triangular de tamaño n.
- La casilla de columna i y fila j corresponde a la subcadena que comienza en la posición i y tiene longitud j.

	the_1	boy_2	$killed_3$	the_4	$dragon_5$
1	the	boy	killed	the	dragon
2	the boy	boy killed	killed the	the dragon	
3	the boy killed	boy killed the	killed the dragon		
4	the boy killed the	boy killed the dragon			
5	the boy killed the dragon				

CYK: idea informal

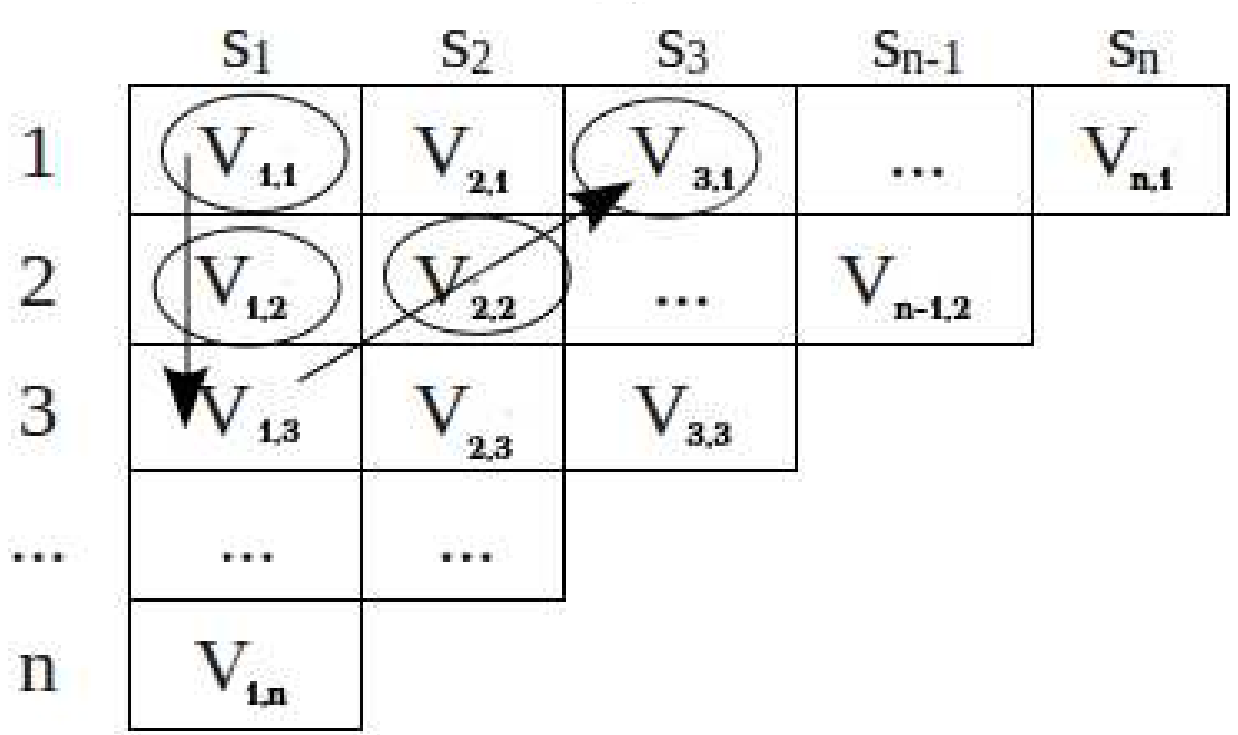
El algoritmo CYK construye una tabla con casillas que contienen las variables de la GLC que son capaces de generar las subcadenas correspondientes a esas casillas.



CYK: idea informal

Cómo hallar $V_{i,j}$, donde i es la posición de inicio y j la longitud de la subcadena.

1. La fila superior contiene las variables que generan directamente terminales: $V_{i,1} = \{A \in V \mid A \Rightarrow^* s_i\}$
2. Completamos la tabla por filas, de arriba hacia abajo: $V_{i,j} = \{A \in V \mid A \Rightarrow^* s_i \ldots s_{i+j-1}\}$.
   a) Para rellenar la casilla para la subcadena $s_1 s_2$ (longitud 2) miramos las casillas de s_1 y s_2 y buscamos variables que puedan producir pares de esas variables en orden.
   b) Para subcadenas más largas, consideramos cada posibilidad de partirlas en dos. Para cada división, consideramos las casillas correspondientes a las dos partes resultantes.



3. La última casilla de la tabla contiene las variables capaces de generar la cadena completa. Miramos si el axioma se encuentra entre ellas.

Pseudocódigo del algoritmo CYK

1: **for** $i = 1$ to n **do**
2: $\quad V_{i,1} = \{A \mid A \to a_i \in P\}$
3: **end for**
4: **for** $j = 2$ to n **do** {Longitud creciente}
5: $\quad$ **for** $i = 1$ to $n - j + 1$ **do** {Posición de inicio}
6: $\quad\quad V_{i,j} = \emptyset$
7: $\quad\quad$ **for** $k = 1$ to $j - 1$ **do** {Punto de corte}
8: $\quad\quad\quad V_{i,j} = V_{i,j} \cup \{A \mid A \to BC \in P, B \in V_{i,k}$ y $C \in V_{i+k,j-k}\}$
9: $\quad\quad$ **end for**
10: $\quad$ **end for**
11: **end for**

Ejemplo

$S \to AB \mid BC$

$A \to BA \mid a$

$B \to CC \mid b$

$C \to AB \mid a$

	b	a	a	b	a
1	{B}	{A, C}	{A, C}	{B}	{A, C}
2	{A, S}	{B}	{S, C}	{A, S}	
3	$\emptyset$	{B}	{B}		
4	$\emptyset$	{S, A, C}			
5	{A, S, C}				