

# Gramáticas Libres de Contexto y Autómatas de Pila

Matemáticas Computacionales  
(TC2020)

M.C. Xavier Sánchez Díaz

sax@itesm.mx



# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

$$① \quad S \rightarrow aSb$$

$$② \quad S \rightarrow \varepsilon$$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

❶  $S \rightarrow aSb$

❷  $S \rightarrow \varepsilon$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Recordemos una GLC, por ejemplo, para generar palabras del lenguaje  $\{a^n b^n\}$ :

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Hacia dónde **crece** la palabra? Deriva usando  $w = \langle 1, 1, 1, 2 \rangle$ .

Sin usar una secuencia dada, al momento de derivar una palabra usando las reglas de la gramática, tenemos que **decidir** entre cuál de las reglas disponibles usar.

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

$$\textcircled{1} S \rightarrow aSb$$

$$\textcircled{2} S \rightarrow \varepsilon$$

¿Cómo sabemos qué regla utilizar?

Nosotros podemos decidir porque sabemos a la palabra a la que queremos llegar. Un compilador no puede hacer eso, por lo que necesita de algún otro mecanismo para **prever** qué regla utilizar.

Generar la función de transición completa para un autómata de pila para  $\{a^n b^n\}$ .



# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

$$\textcircled{1} \ S \rightarrow aSb$$

$$\textcircled{2} \ S \rightarrow \varepsilon$$

¿Cómo sabemos qué regla utilizar?

Nosotros podemos decidir porque sabemos a la palabra a la que queremos llegar. Un compilador no puede hacer eso, por lo que necesita de algún otro mecanismo para **prever** qué regla utilizar.

Generar la función de transición completa para un autómata de pila para  $\{a^n b^n\}$ .

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Cómo sabemos qué regla utilizar?

Nosotros podemos decidir porque sabemos a la palabra a la que queremos llegar. Un compilador no puede hacer eso, por lo que necesita de algún otro mecanismo para **prever** qué regla utilizar.

Generar la función de transición completa para un autómata de pila para  $\{a^n b^n\}$ .

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

$$\textcircled{1} \quad S \rightarrow aSb$$

$$\textcircled{2} \quad S \rightarrow \varepsilon$$

¿Cómo sabemos qué regla utilizar?

Nosotros podemos decidir porque sabemos a la palabra a la que queremos llegar. Un compilador no puede hacer eso, por lo que necesita de algún otro mecanismo para **prever** qué regla utilizar.

Generar la función de transición completa para un autómata de pila para  $\{a^n b^n\}$ .

# Crecimiento y no-determinismo en GLCs

Compiladores LL y LR

$$\textcircled{1} S \rightarrow aSb$$

$$\textcircled{2} S \rightarrow \varepsilon$$

¿Cómo sabemos qué regla utilizar?

Nosotros podemos decidir porque sabemos a la palabra a la que queremos llegar. Un compilador no puede hacer eso, por lo que necesita de algún otro mecanismo para **prever** qué regla utilizar.

Generar la función de transición completa para un autómata de pila para  $\{a^n b^n\}$ .

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Tape	Pop	Move	Push
<i>aabb</i>	\$	<i>R</i>	<i>A</i>
<i>abb</i>	<i>A</i>	<i>R</i>	<i>AA</i>
<i>bb</i>	<i>A</i>	<i>R</i>	$\varepsilon$
<i>b</i>	<i>A</i>	<i>R</i>	$\varepsilon$
■	\$	<i>N</i>	$\varepsilon$

Hay muchas reglas que no estamos utilizando, y nos guiamos por el hecho de ver hacia *adelante*. Sabiendo que busco *aabb*, entonces lo más lógico es usar  $S \rightarrow aSb$  dos veces, para poder *finalizar* con  $S \rightarrow \varepsilon$ .

Este tipo de compiladores, que leen de izquierda a derecha, y que tienen una ventana de “predicción” se conocen como *Left to right Leftmost derivation*, o **LL** por sus siglas en inglés.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Tape	Pop	Move	Push
<i>aabb</i>	\$	<i>R</i>	<i>A</i>
<i>abb</i>	<i>A</i>	<i>R</i>	<i>AA</i>
<i>bb</i>	<i>A</i>	<i>R</i>	$\varepsilon$
<i>b</i>	<i>A</i>	<i>R</i>	$\varepsilon$
■	\$	<i>N</i>	$\varepsilon$

Hay muchas reglas que no estamos utilizando, y nos guiamos por el hecho de ver hacia *adelante*. Sabiendo que busco *aabb*, entonces lo más lógico es usar  $S \rightarrow aSb$  dos veces, para poder *finalizar* con  $S \rightarrow \varepsilon$ .

Este tipo de compiladores, que leen de izquierda a derecha, y que tienen una ventana de “predicción” se conocen como *Left to right Leftmost derivation*, o **LL** por sus siglas en inglés.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Tape	Pop	Move	Push
<i>aabb</i>	\$	<i>R</i>	<i>A</i>
<i>abb</i>	<i>A</i>	<i>R</i>	<i>AA</i>
<i>bb</i>	<i>A</i>	<i>R</i>	$\varepsilon$
<i>b</i>	<i>A</i>	<i>R</i>	$\varepsilon$
■	\$	<i>N</i>	$\varepsilon$

Hay muchas reglas que no estamos utilizando, y nos guiamos por el hecho de ver hacia *adelante*. Sabiendo que busco *aabb*, entonces lo más lógico es usar  $S \rightarrow aSb$  dos veces, para poder *finalizar* con  $S \rightarrow \varepsilon$ .

Este tipo de compiladores, que leen de izquierda a derecha, y que tienen una ventana de “predicción” se conocen como *Left to right Leftmost derivation*, o **LL** por sus siglas en inglés.

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Por el contrario, hay compiladores que leen *al revés*, de derecha a izquierda (con respecto a las reglas de la gramática, primero  $aSB$  y luego  $S \rightarrow$ ). De este modo, el árbol de derivación se recorre hacia atrás.

Este tipo de compiladores se conoce como *Left to right Rightmost derivation*, o LR por sus siglas en inglés.

¿Pero cómo convertimos de una GLC a un AP y viceversa?



# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Por el contrario, hay compiladores que leen *al revés*, de derecha a izquierda (con respecto a las reglas de la gramática, primero  $aSB$  y luego  $S \rightarrow$ ). De este modo, el árbol de derivación se recorre hacia atrás.

Este tipo de compiladores se conoce como *Left to right Rightmost derivation*, o **LR** por sus siglas en inglés.

¿Pero cómo convertimos de una GLC a un AP y viceversa?

# Crecimiento y no-determinismo en GLCs

## Compiladores LL y LR

Por el contrario, hay compiladores que leen *al revés*, de derecha a izquierda (con respecto a las reglas de la gramática, primero  $aSB$  y luego  $S \rightarrow$ ). De este modo, el árbol de derivación se recorre hacia atrás.

Este tipo de compiladores se conoce como *Left to right Rightmost derivation*, o **LR** por sus siglas en inglés.

¿Pero cómo convertimos de una GLC a un AP y viceversa?

# Múltiples maneras en la literatura

## Compiladores LL y LR

Como en el caso pasado, distintas personas lo manejan de distinta manera:

- 1 Brena (2003) diferencia entre el tipo de compilador, y convierte una gramática de cierto tipo a un AP utilizando un algoritmo que depende del tipo de compilador.
- 2 Ullman (1979), uno de los autores del libro más usado para este curso a nivel mundial, utiliza otra versión, sólo para compiladores LL y es la que utilizaremos.

# Múltiples maneras en la literatura

## Compiladores LL y LR

Como en el caso pasado, distintas personas lo manejan de distinta manera:

- 1 Brena (2003) diferencia entre el tipo de compilador, y convierte una gramática de cierto tipo a un AP utilizando un algoritmo que depende del tipo de compilador.
- 2 Ullman (1979), uno de los autores del libro más usado para este curso a nivel mundial, utiliza otra versión, sólo para compiladores LL y es la que utilizaremos.

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene un estado  $q$
- El alfabeto de la cinta de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El alfabeto de la pila de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El símbolo inicial (de la pila) de  $M$  sigue siendo  $\$$  (como antes)

La función de transición puede calcularse con respecto a eso (como antes)

Las condiciones de aceptación son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)



# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Conversión GLC a AP

Sea  $L$  un lenguaje de una GLC  $G$ .

Para construir un AP  $M$  que acepte  $L$ , hay que considerar lo siguiente:

- $M$  tiene **un estado**  $q$
- El **alfabeto de la cinta** de  $M$ ,  $\Sigma$  es igual a las terminales de  $G$
- El **alfabeto de la pila** de  $M$ ,  $\Gamma \subseteq V \cup \Sigma$
- El **símbolo inicial (de la pila)** de  $M$  sigue siendo  $\$$  (como antes)

La **función de transición** puede calcularse con respecto a eso (como antes)

Las **condiciones de aceptación** son:

- La pila está vacía (como antes)
- La cinta llegó al final (como antes)

# Ejemplo: $\{a^n b^n\}$

## Formalización y diseño de APs

❶  $S \rightarrow aSb$

❷  $S \rightarrow ab$

•  $Q = \{q\}$

•  $\Sigma = \{a, b\}$

•  $\Gamma = \{\$, S\}$

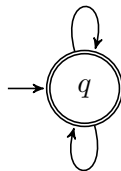
•  $\delta =$

- ▶  $((q, a, \$)(q, \$S))$
- ▶  $((q, a, S)(q, SS))$
- ▶  $((q, b, S)(q, \varepsilon))$
- ▶  $((q, \square, \$)(q, \varepsilon))$

•  $q = q$

•  $F = \{q\}$

$(a, \$, \$S), (a, S, SS)$



$(b, S, \varepsilon), (\square, \$, \varepsilon)$

¿Es este enfoque correcto? ¿Qué pasa con  $aabb$ ? ¿Y con  $abab$ ?

# Ejemplo: $\{a^n b^n\}$

## Formalización y diseño de APs

❶  $S \rightarrow aSb$

❷  $S \rightarrow ab$

•  $Q = \{q\}$

•  $\Sigma = \{a, b\}$

•  $\Gamma = \{\$, S\}$

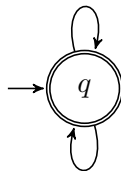
•  $\delta =$

- ▶  $((q, a, \$)(q, \$S))$
- ▶  $((q, a, S)(q, SS))$
- ▶  $((q, b, S)(q, \varepsilon))$
- ▶  $((q, \square, \$)(q, \varepsilon))$

•  $q = q$

•  $F = \{q\}$

$(a, \$, \$S), (a, S, SS)$



$(b, S, \varepsilon), (\square, \$, \varepsilon)$

¿Es este enfoque correcto? ¿Qué pasa con  $aabb$ ? ¿Y con  $abab$ ?

# Ejemplo: $\{a^n b^n\}$

## Formalización y diseño de APs

❶  $S \rightarrow aSb$

❷  $S \rightarrow ab$

•  $Q = \{q\}$

•  $\Sigma = \{a, b\}$

•  $\Gamma = \{\$, S\}$

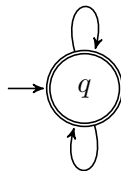
•  $\delta =$

- ▶  $((q, a, \$)(q, \$S))$
- ▶  $((q, a, S)(q, SS))$
- ▶  $((q, b, S)(q, \varepsilon))$
- ▶  $((q, \square, \$)(q, \varepsilon))$

•  $q = q$

•  $F = \{q\}$

$(a, \$, \$S), (a, S, SS)$



$(b, S, \varepsilon), (\square, \$, \varepsilon)$

¿Es este enfoque correcto? ¿Qué pasa con  $aabb$ ? ¿Y con  $abab$ ?



## Ejemplo: $\{a^n b^n\}$

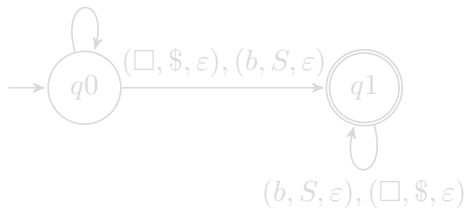
### Formalización y diseño de APs

Claramente hemos agotado las combinaciones posibles entre  $\Sigma \times \Gamma \dots$

¿Con cuál otro conjunto que tenga el AP podemos hacer producto cartesiano para obtener más posibilidades de cómputo?

Con el **conjunto de estados**.

$(a, \$, \$S), (a, S, SS)$



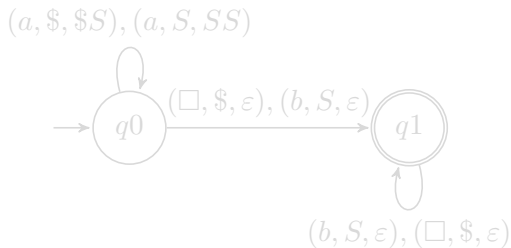
## Ejemplo: $\{a^n b^n\}$

### Formalización y diseño de APs

Claramente hemos agotado las combinaciones posibles entre  $\Sigma \times \Gamma \dots$

¿Con cuál otro conjunto que tenga el AP podemos hacer producto cartesiano para obtener más posibilidades de cómputo?

Con el **conjunto de estados**.



## Ejemplo: $\{a^n b^n\}$

### Formalización y diseño de APs

Claramente hemos agotado las combinaciones posibles entre  $\Sigma \times \Gamma \dots$

¿Con cuál otro conjunto que tenga el AP podemos hacer producto cartesiano para obtener más posibilidades de cómputo?

Con el **conjunto de estados**.

$(a, \$, \$S), (a, S, SS)$

