Sesión 22

Parseo Top-down

Proceso de parseo

- Del Latin: Partes del habla o categorías gramáticales
- Si G en una CFG sober Σ & $x \in \Sigma^*$, parsear x consiste en encontrar una derivación para x en G o determinar que ésta no existe (i.e. por lo que x no está en el lenguaje de G).

Proceso de parseo

- Antecedentes: simulación de derivaciones en G por un a PDA:
  - Top-down (derivaciones más izquierdas)
  - Bottom-up (derivaciones más derechas)
- Pero, estos procesos son no determinísticos y no proveén de un algoritmo directamente!

Algoritmos de parseo

- Algoritmo trivial: explorar todas las trayectorias (en el espacion no determinístico) en cierto orden (i.e. depth first o breath first) & ver si alguna trayectoria termina exitosamente
  - Muy costoso: búsqueda exponencial!

Algoritmos de parseo

- Enfrentar el no-determinismo directamente:
  - Utilizar toda la información disponible en cada paso de la computación (local) y seleccionar la mejor alternativa (en algunos casos determinísticamente)
  - Capitalizar la forma de la gramática: Para seleccionar la siguiente movida considerar no sólo el símbolo hasta arriba del stack, sino también uno o más de los símbolos que siguen en la cinta!

Top-down *Lookahead* Parser

- Movida 1:
  - Si el símbolo hasta arriba del stack es una variable en el lado izquierdo de una producción, reemplazar dicha variable por el lado derecho de la producción
  - Inspeccionar el símbolo actual (i.e. consumir) y escoger sólo producciones que tengan a dicho símbolo como el más izquierdo del lado derecho!
  - Usar no-determinismo sólo si no hay opción!

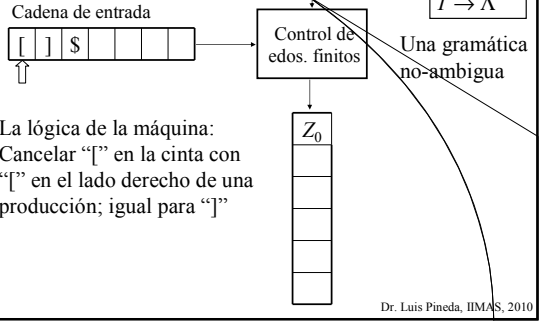
Top-down *Lookahead* Parser

- Movida 2:
  - Si el símbolo de entrada corresponde con el símbolo hasta arriba del stack, consumir el símbolo y pop.

Simulación top-down

L = El lenguaje de los paréntesis balanceados

$S \rightarrow T\$$
$T \rightarrow [T]T$
$T \rightarrow \Lambda$

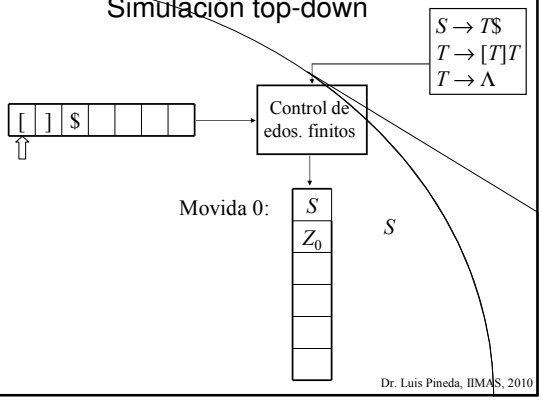


AP no determinístico

Id	estado	entrada	top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	(q_1, TZ_0)
3	q_1	Λ	T	$(q_1, [T]T)$, (q_1, Λ)
4	q_1	[	[	(q_1, Λ)
5	q_1	]	]	(q_1, Λ)
6	q_1	$	$	(q_1, Λ)
7	q_1	Λ	Z_0	(q_2, Z_0)
Otras combinaciones				ninguna

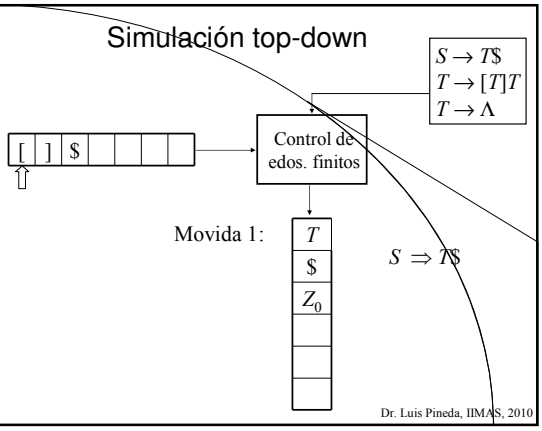
Simulación top-down

$S \rightarrow T\$$
$T \rightarrow [T]T$
$T \rightarrow \Lambda$



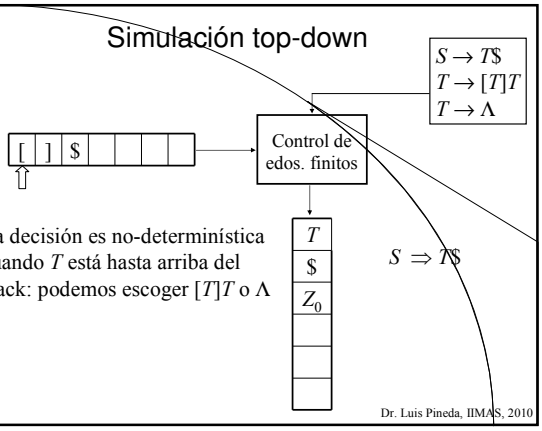
Simulación top-down

$S \rightarrow T\$$
$T \rightarrow [T]T$
$T \rightarrow \Lambda$



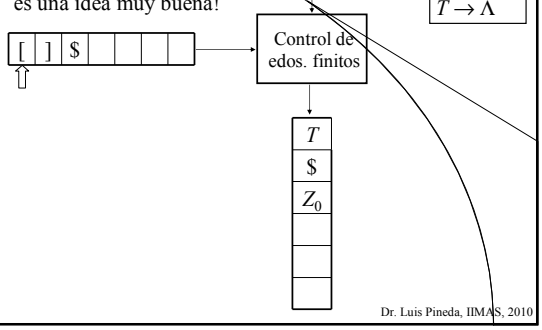
Simulación top-down

$S \rightarrow T\$$
$T \rightarrow [T]T$
$T \rightarrow \Lambda$



Simulación top-down

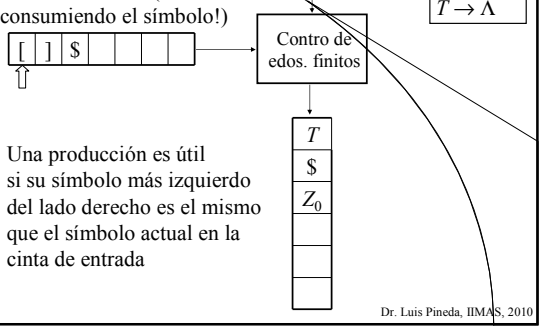
Claramente, escoger Λ no es una idea muy buena!



$S \to T\$$
$T \to [T]T$
$T \to \Lambda$

Ver hacia adelande*: Lookahead*

Podemos ver el símbolo actual en la cinta (i.e. consumiendo el símbolo!)



$S \to T\$$
$T \to [T]T$
$T \to \Lambda$

Una producción es útil si su símbolo más izquierdo del lado derecho es el mismo que el símbolo actual en la cinta de entrada

¡Ver hacia delante!

- Remover el no-determinismo inspeccionando un símbolo hacia adelante
- Sólo considerar producciones que tengan a dicho símbolo como el más izquierdo del lado derecho de la producción
- Diseñar una AP determinístico que corresponda con el AP no-determinístico original

AP lookahead determinístico

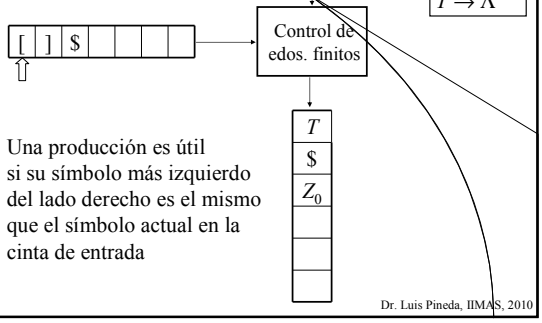
Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	$(q_\$, \Lambda)$
8	$q_\$$	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

Consume lookahhead symbol

$T \to [T]T$

Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	$(q_\$, \Lambda)$
8	$q_\$$	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

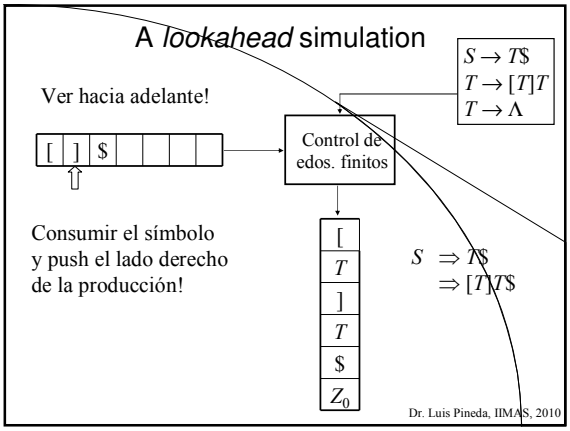
Ver hacia adelande*: Lookahead*



$S \to T\$$
$T \to [T]T$
$T \to \Lambda$

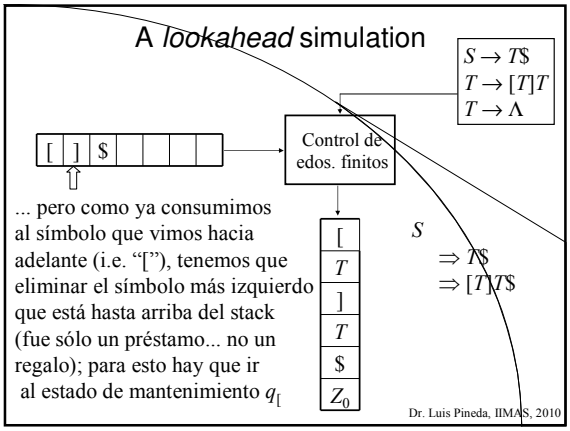
Una producción es útil si su símbolo más izquierdo del lado derecho es el mismo que el símbolo actual en la cinta de entrada

A *lookahead* simulation



Dr. Luis Pineda, IIMAS, 2010

A *lookahead* simulation



Dr. Luis Pineda, IIMAS, 2010

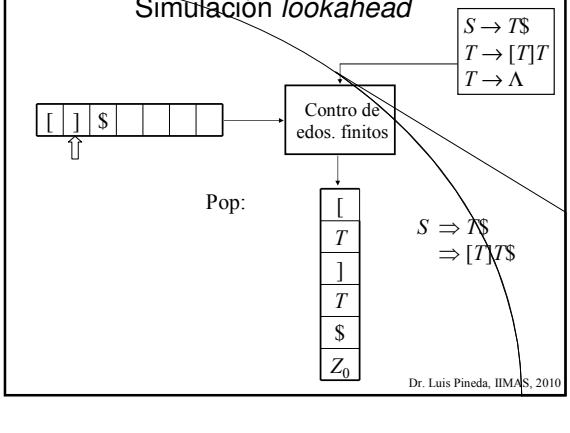
Consume lookahhead symbol

$T \to [T]T$

Id	State	Input	Stack symbol	Move(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	(q_s, Λ)
8	q_s	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
Other combinations				non

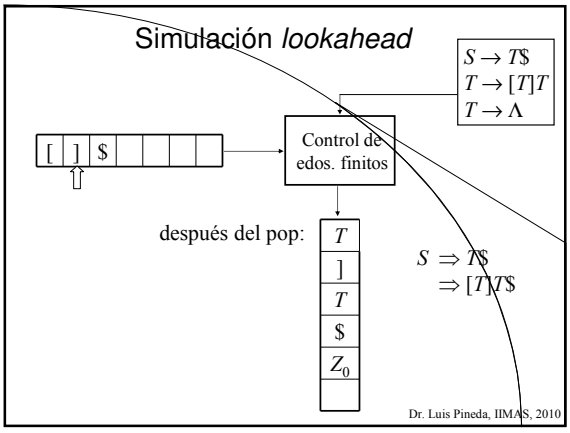
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



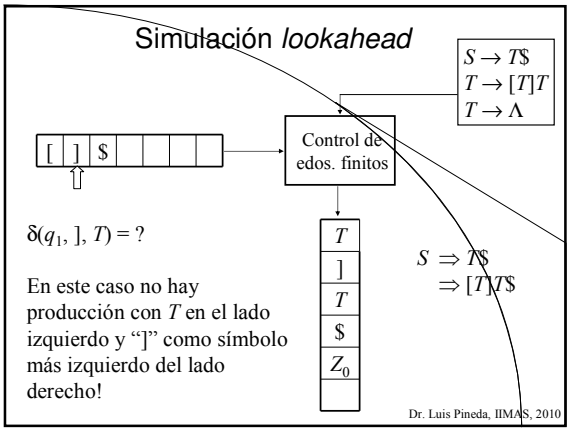
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



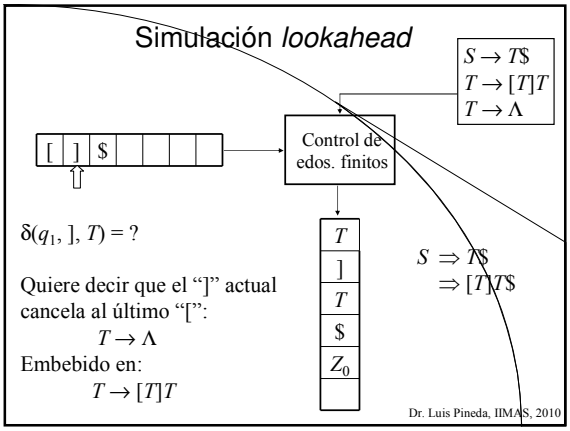
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



$\delta(q_1,], T) = ?$

Quiere decir que el "]" actual cancela al último "[":

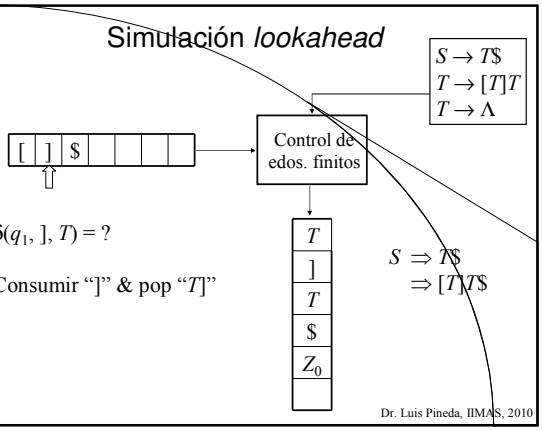
$T \to \Lambda$

Embebido en:

$T \to [T]T$

$S \Rightarrow T\$$
$\Rightarrow [T]T\$$

Simulación *lookahead*



$\delta(q_1,], T) = ?$

Consumir "]" & pop "T]"

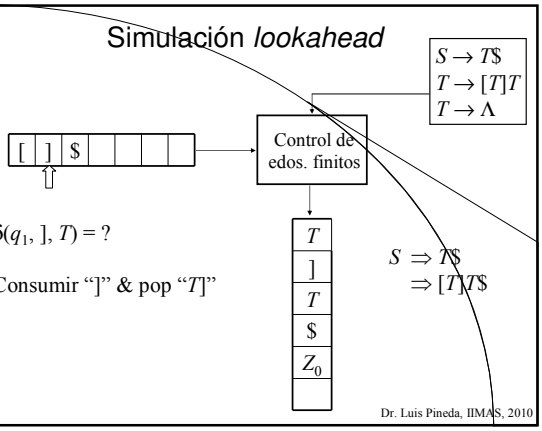
$S \Rightarrow T\$$
$\Rightarrow [T]T\$$

Pop "T]"

$T \to \Lambda$

Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	(q_s, Λ)
8	q_s	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

Simulación *lookahead*



$\delta(q_1,], T) = ?$

Consumir "]" & pop "T]"

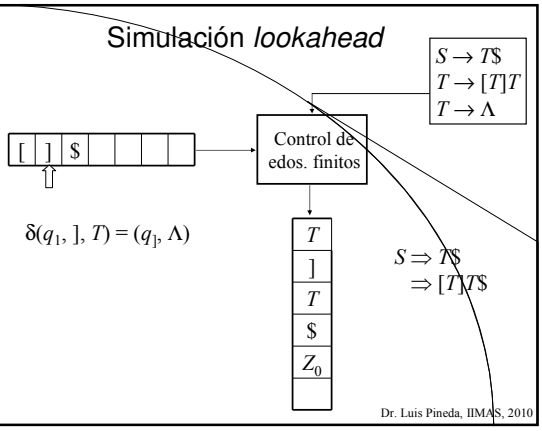
$S \Rightarrow T\$$
$\Rightarrow [T]T\$$

No hay entrada para $T \to \Lambda$

Id	estado	entrada	Top del stack	Movida(s)
3	q_1	[	T	$(q_1, [T]T)$
X	q_1	Λ	T	(q_1, Λ)

No hay entrada en la tabla para la regla: $T \to \Lambda$

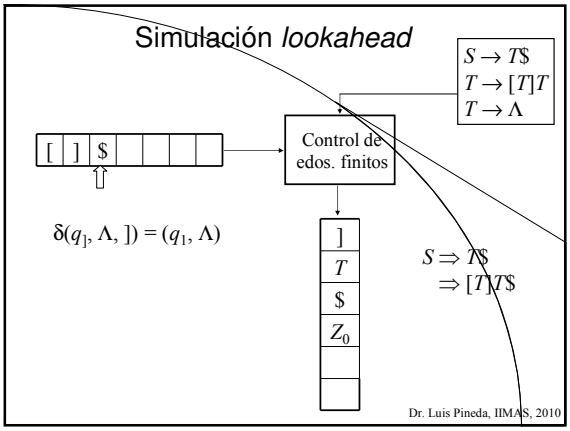
Simulación *lookahead*



$\delta(q_1,], T) = (q_1, \Lambda)$

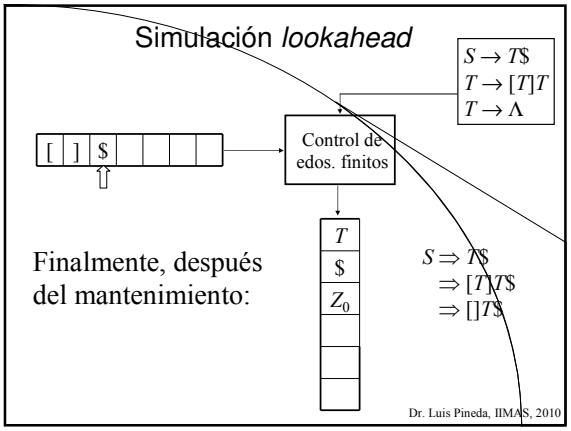
$S \Rightarrow T\$$
$\Rightarrow [T]T\$$

Simulación *lookahead*



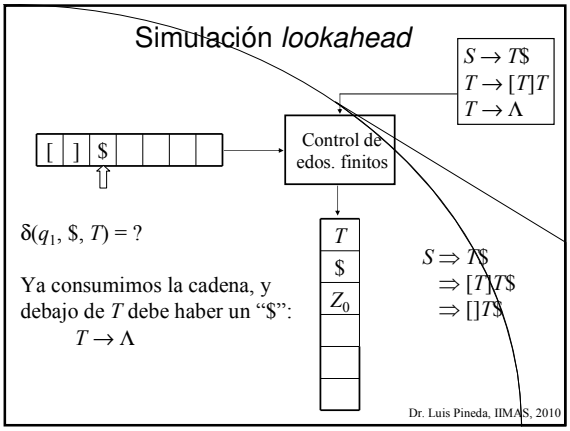
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



Dr. Luis Pineda, IIMAS, 2010

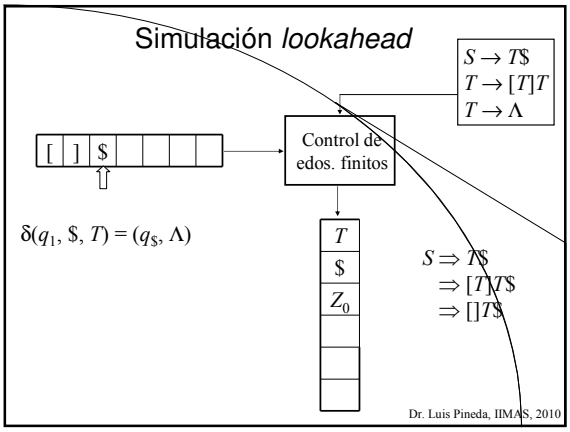
lookahhead el symbol $

$T \to \Lambda$

Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	$(q_\$, \Lambda)$
8	$q_\$$	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

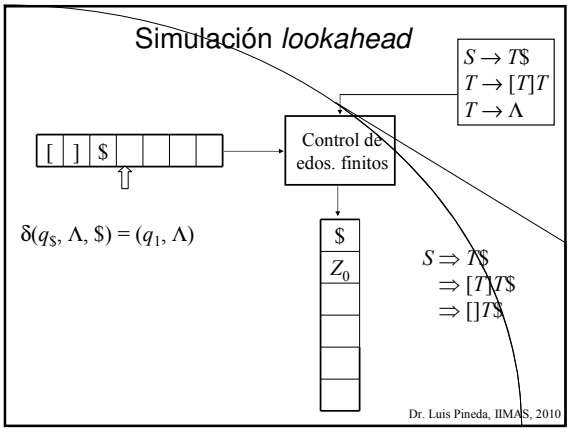
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



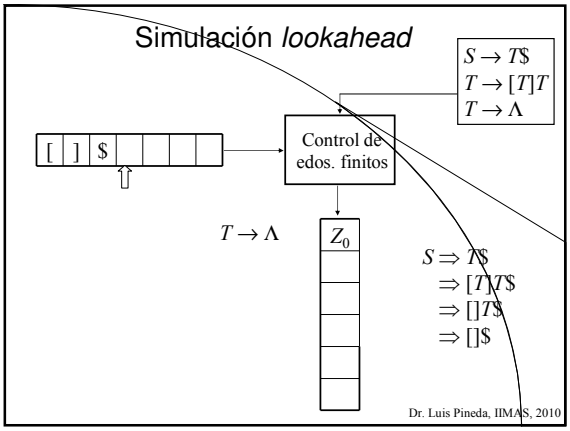
Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



Dr. Luis Pineda, IIMAS, 2010

Simulación *lookahead*



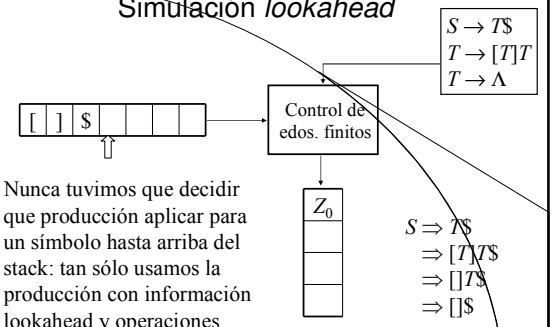
Operaciones de pop

Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	$(q_\$, \Lambda)$
8	$q_\$$	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

Estado final

Id	estado	entrada	Top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	$(q_1, T\$Z_0)$
3	q_1	[	T	$(q_1, [T]T)$
4	q_1	Λ	[	(q_1, Λ)
5	q_1	]	T	(q_1, Λ)
6	q_1	Λ	]	(q_1, Λ)
7	q_1	$	T	$(q_\$, \Lambda)$
8	$q_\$$	Λ	$	(q_1, Λ)
9	q_1	[	[	(q_1, Λ)
10	q_1	]	]	(q_1, Λ)
11	q_1	$	$	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
otras combinaciones				ninguna

Simulación *lookahead*



Nunca tuvimos que decidir que producción aplicar para un símbolo hasta arriba del stack: tan sólo usamos la producción con información lookahead y operaciones de mantenimiento al stack!

Derivación más izquierda!

El problema de la recursión izquierda

- Recursión izquierda:
  - Un problema con la simulación top-down lookahead:
- Otra gramática no ambigua para L

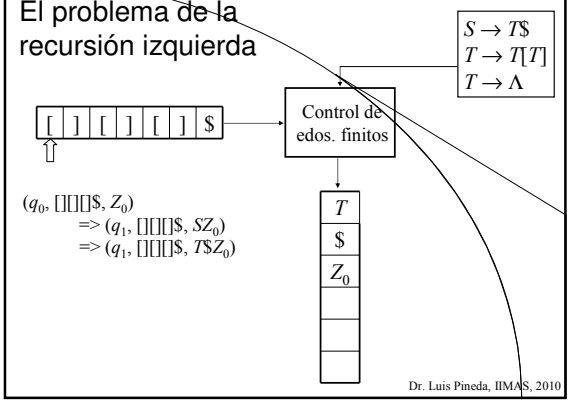
  $S \to T\$$

  $T \to T[T] \mid \Lambda$
- Una derivación más izquierda:

  $S \Rightarrow T\$ \Rightarrow T[T]\$ \Rightarrow T[T][T]\$ \Rightarrow T[T][T][T]\$\ldots$

  $\ldots\Rightarrow []\,[]\,[]\$$
- Pero no hay una operación que implemente la producción $T \to \Lambda$ en el AP determinístico!

El problema de la recursión izquierda

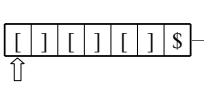


AP no determinístico

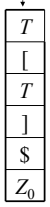
Id	estado	entrad	top del stack	Movida(s)
1	q_0	Λ	Z_0	(q_1, SZ_0)
2	q_1	Λ	S	(q_1, TZ_0)
3	q_1	Λ	T	$(q_1, T[T])$, (q_1, Λ)
4	q_1	[	[	(q_1, Λ)
5	q_1	]	]	(q_1, Λ)
6	q_1	$	$	(q_1, Λ)
7	q_1	Λ	Z_0	(q_2, Z_0)
Otras combinaciones				ninguna

El problema de la recursión izquierda

$S \rightarrow T\$$
$T \rightarrow T[T]$
$T \rightarrow \Lambda$



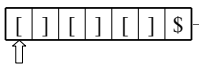
Control de edos. finitos



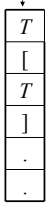
$(q_0, [][][]\$, Z_0)$
$\Rightarrow (q_1, [][][]\$, SZ_0)$
$\Rightarrow (q_1, [][][]\$, T\$Z_0)$
$\Rightarrow (q_1, [][][]\$, T[T]\$Z_0)$

El problema de la recursión izquierda

$S \rightarrow T\$$
$T \rightarrow T[T]$
$T \rightarrow \Lambda$



Control de edos. finitos



$(q_0, [][][]\$, Z_0)$
$\Rightarrow (q_1, [][][]\$, SZ_0)$
$\Rightarrow (q_1, [][][]\$, T\$Z_0)$
$\Rightarrow (q_1, [][][]\$, T[T]\$Z_0)$
$\Rightarrow (q_1, [][][]\$, T[T][T]\$Z_0)$
$\Rightarrow (q_1, [][][]\$, T[T][T][T]\$Z_0)$
…
per secula seculorum!

Eleminando la recursión izquierda

- La solución:
  - Encontra una gramática equivalente sin recursión izq.
- Considerar las producciones-T (β no empieza con *T)*

$$T \rightarrow T\alpha \mid \beta$$

- Es posible generar $\beta\alpha^n$:

$$T \rightarrow T\alpha,\ T \rightarrow T\alpha\alpha,\ ...,\ T \rightarrow T\alpha^n$$

pero $T \rightarrow \beta$ & $T \rightarrow \beta\alpha^n$

- *T* genera la concatenación de β con la cerradura de α:

$$T \rightarrow \beta U \ \ \& \ U \rightarrow \alpha^n$$

$$T \rightarrow \beta U \ \ \& \ U \rightarrow \alpha U \mid \Lambda$$

Eleminando la recursión izquierda

- La solución:
  - Encontra una gramática equivalente sin recursión izq.
- Considerar las producciones-T (β no empieza con *T)*

$$T \rightarrow T\alpha \mid \beta$$

- Reemplazar por:

$$T \rightarrow \beta U \ \ \& \ U \rightarrow \alpha U \mid \Lambda$$

- Ejemplo:

$$T \rightarrow T[T] \mid \Lambda \qquad (\alpha = [T],\ \beta = \Lambda)$$

- Reemplazar por:

$$T \rightarrow U \ \ \& \ U \rightarrow [T]U \mid \Lambda$$

Factorización

- Otra vez la gramática original:

$$S \rightarrow T\$,\ \ T \rightarrow [T]T \mid \Lambda$$

- Eleminar las producciones-Λ

$$S \rightarrow T\$,\ S \rightarrow \$,\ T \rightarrow [T]T \mid []T \mid [T] \mid []$$

- Considerar el parseo por un lookahead AP:

$$S \rightarrow T\$,\ S \rightarrow \$,\ T \rightarrow [T]T \mid []T \mid [T] \mid []$$

  - Cuando *T* está hasta arriba del stack hay 4 opciones
- Factorizar el primer símbolo del lado derecho:

$$S \rightarrow T\$,\ S \rightarrow \$,\ T \rightarrow [\mathrm{U},\ \mathrm{U} \rightarrow T]T \mid]T \mid T] \mid]$$

Factorización

- Los lados derechos que empiezan con $T]$ se pueden factorizar:
  $S \to T\$, S \to \$, T \to [U, \quad U \to T]T \mid]T \mid T] \mid]$
- Factorizar otra vez los lados derechos que empiezan igual:
  Sea $W \to T \mid \Lambda$
  Entonces en $U \to T]W \mid]W$ $\quad T \;\; \Lambda \;\; T \;\; \Lambda$

  Consecuentemente:
  $S \to T\$, S \to \$, T \to [U, \quad U \to T]W \mid]W \;\&\; W \to T \mid \Lambda$
- Eliminando a T $\quad (T \to [U)$:
  $S \to [U\$, S \to \$, U \to [U]W \mid]W \;\&\; W \to [U \mid \Lambda$
- El primer símbolo del lado derecho de todas las producciones con el mismo lado izquierdo es siempre diferente

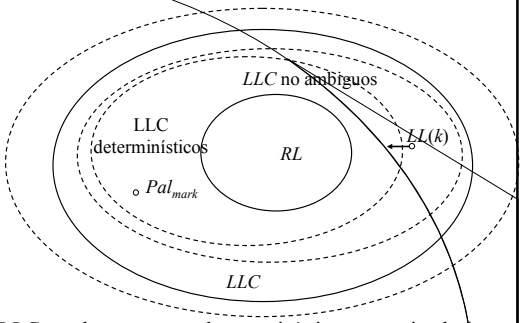
Gramáticas *LL*(1)

- Gramáticas *LL*(1):
  - Sin recursión izquierda
  - Factorizadas: Para todas las producciones $\alpha \to a_i\beta$, donde α_i es diferentes para todas las producciones con el mismo lado izquierdo α

Gramáticas *LL*(1)

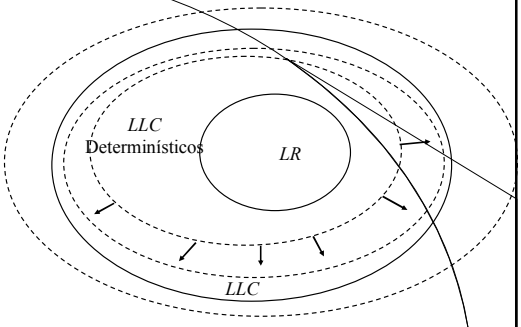
- Se pueden parsear determinísticamente con un AP top-down viendo un símbolo hacia delante en la cinta de entrada
- Gramáticas *LL*(*k*):
  - Parseo top-down determinístico
  - Viendo *k* símbolos hacia delante en la cinta de entrada

LLC no ambiguos



- LLC pueden parsearse deteterminísticamente viendo *k* hacia delante en la cinta de entrada

Aumentando el espacio de las *LLC-D*



El parseo determinístico extiende la clase de los *LLC-D*