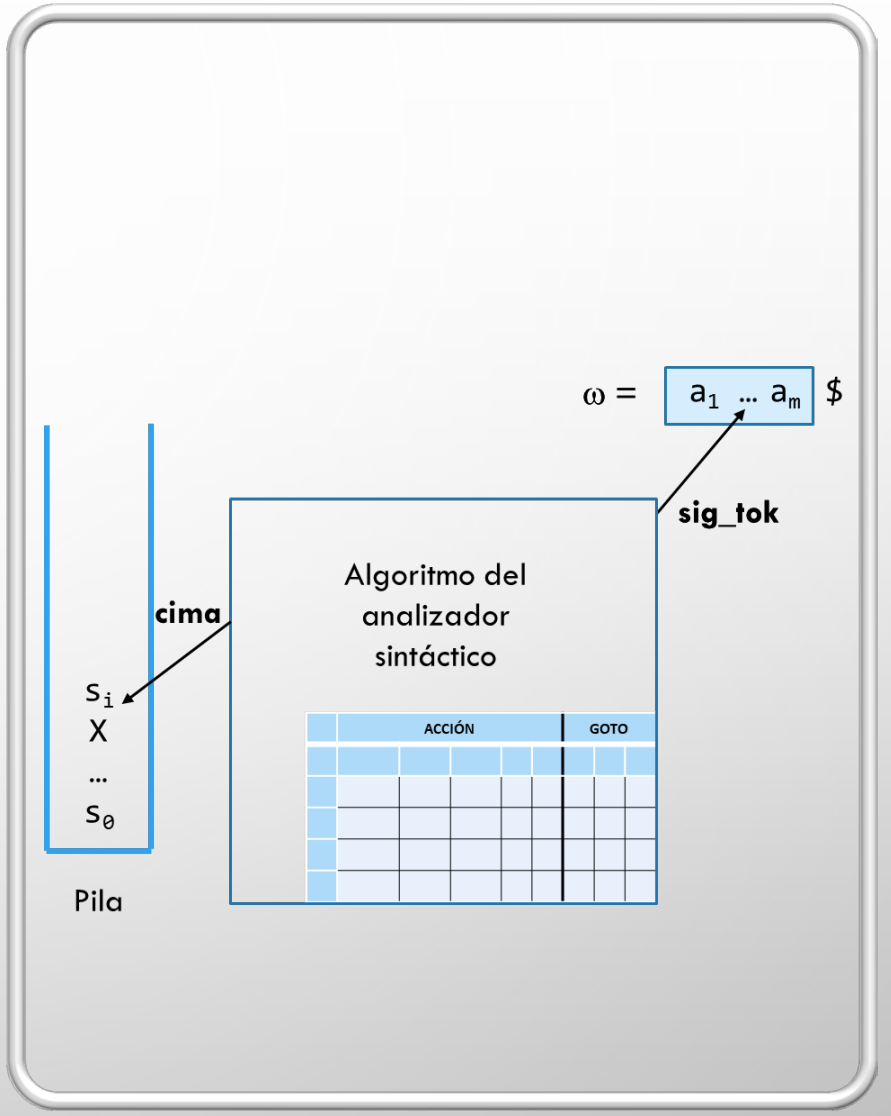
ANÁLISIS SINTÁCTICO ASCENDENTE CON GRAMÁTICAS LR(1)

ANALIZADOR SINTÁCTICO SIMPLE LR(1)

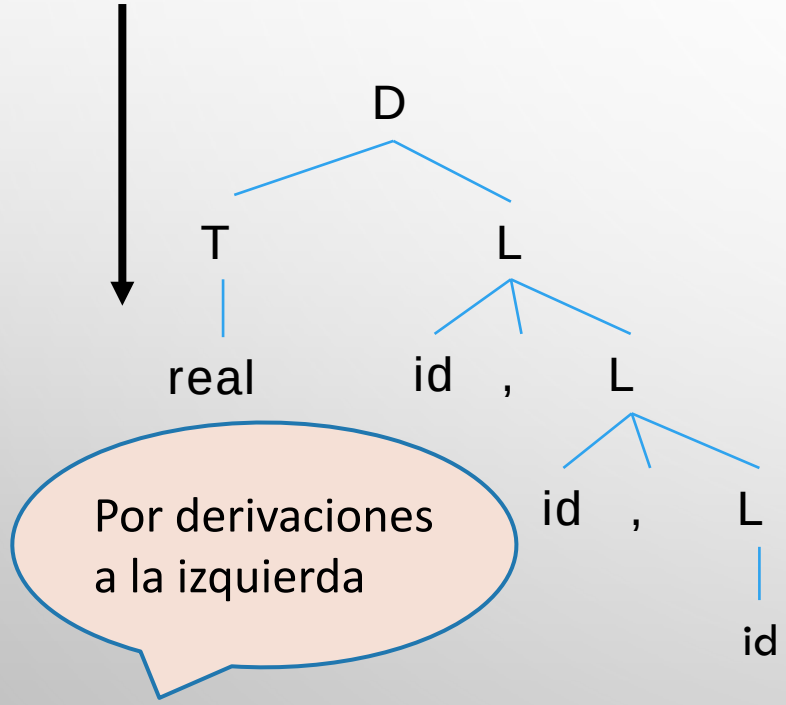


Ejemplo de construcción del árbol

1. D → T L
2. T → integer
3. T → real
4. L → id , L
5. L → id

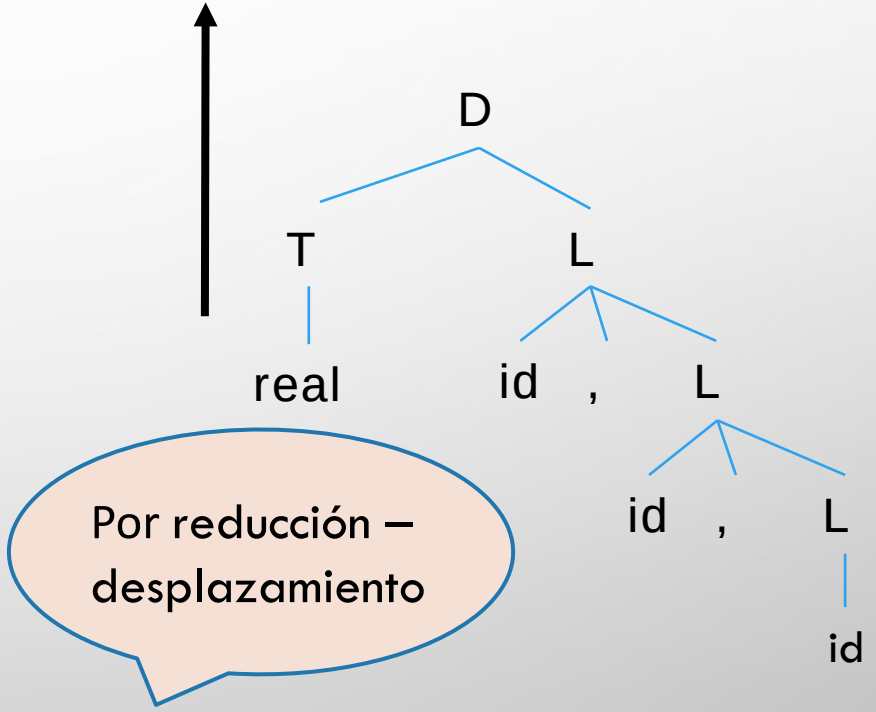
ω = `real id , id , id`

A. St. Descendente



Parse: 1 3 4 4 5

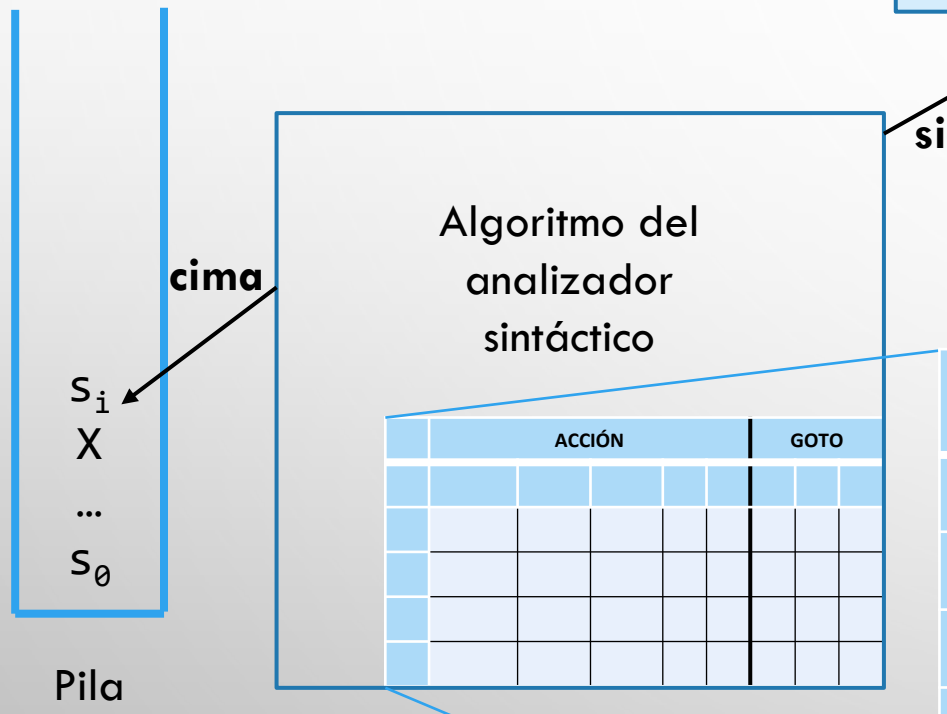
A. St. Ascendente



Parse: 3 5 4 4 1

Analizador Sintáctico Ascendente LR

$\omega =$ $a_1\ a_2\ a_3\ a_4\ a_5\ \dots\ a_m$ \$



Las tablas tienen por filas los estados, y por columnas **símbolos gramaticales** y \$

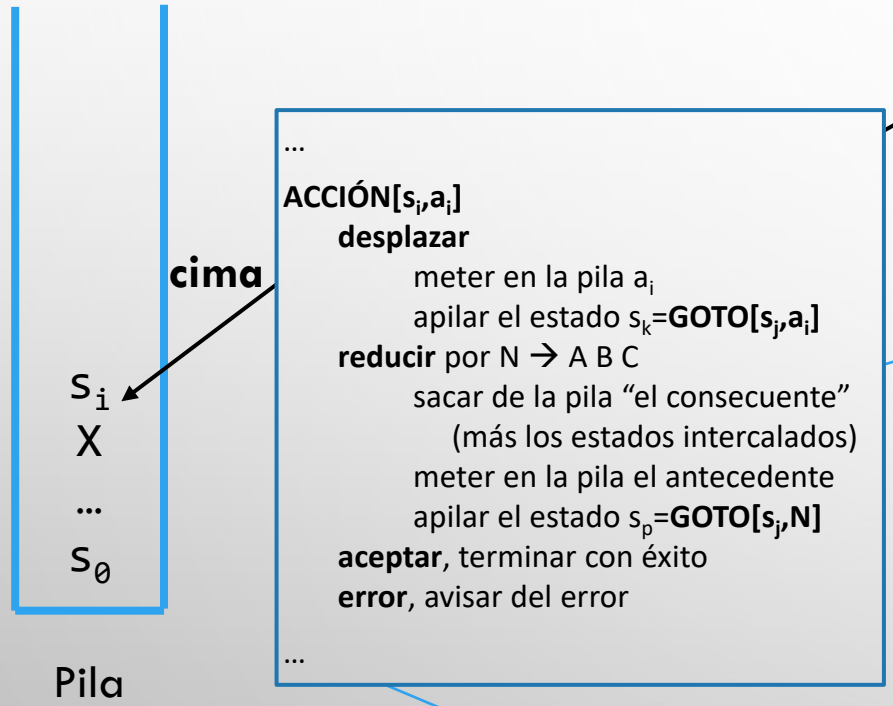
	ACCIÓN					**GOTO**		
	t_1	t_2	t_3	...	\$	X_1	...	X_j
s_0	desp			...			...	
s_1				...	acept		...	
s_2		red $N_3 \rightarrow t_2$	red $N_2 \rightarrow \lambda$	...	...	**s_4**	...	**s_q**
...	...	...	...	...	...		...	...
s_q	red $N_p \rightarrow N_3\ t_1$	desp			...	**s_3**		...

Analizador Sintáctico Ascendente LR

- Pila → en la que construye implícitamente el árbol
  - En la pila almacena símbolos gramaticales y estados (siempre un estado en la cima)
- Tabla **ACCIÓN** → contiene la acción a realizar para cada par "estado, siguiente_token"
  - Las acciones posibles son desplazar, reducir, aceptar y error
- Tabla **GOTO** → contiene el estado a apilar tras desplazar o reducir
- Funcionamiento: en cada instante, mira el estado de la cima de la pila y el sig_token y accede a la tabla ACCIÓN:
  - Desplazar → mete el sig_token en la pila
  - Reducir → saca de la pila el consecuente y mete el antecedente
  - Aceptar → termina con éxito (cadena correcta)
  - Error → detecta un error sintáctico en la cadena
- Configuración inicial: la pila contiene el estado inicial s_0
- Configuración final: la pila contiene el estado inicial s_0, el axioma de G, y el estado final s_1

Analizador Sintáctico Ascendente LR

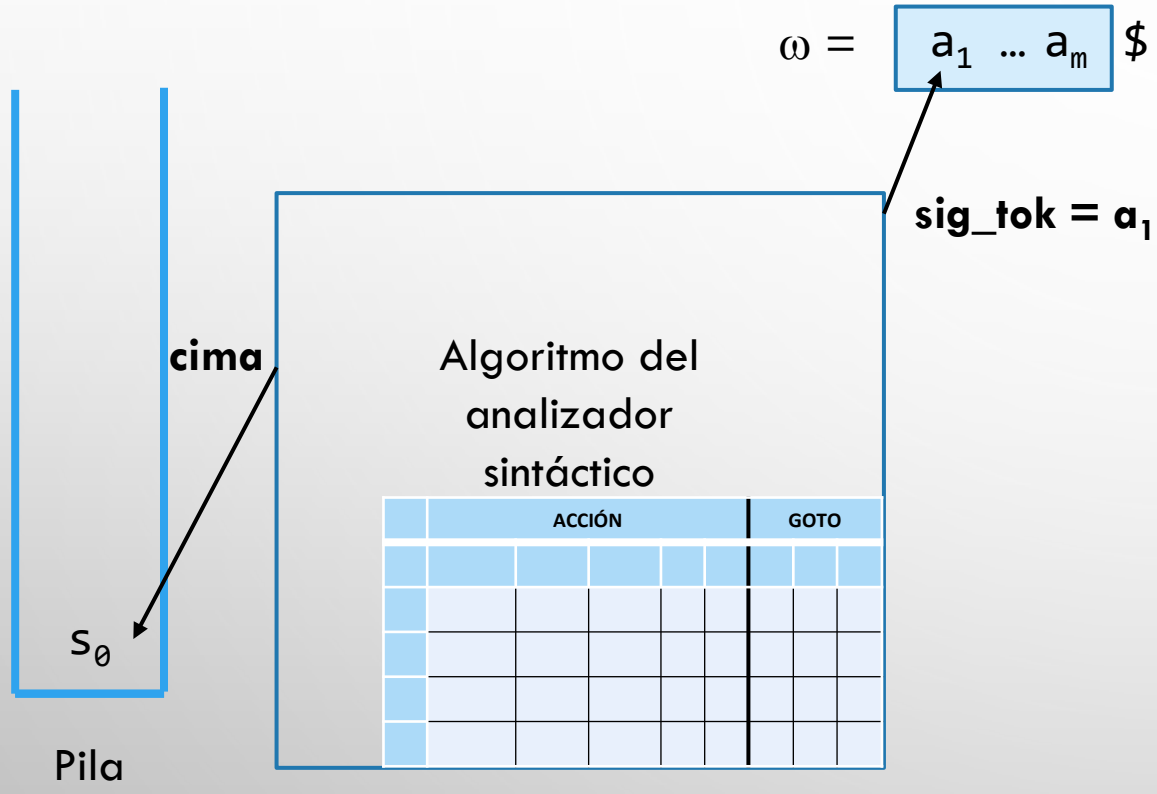
$\omega =$ $a_1\ a_2\ a_3\ a_4\ a_5\ \dots\ a_m$ \$



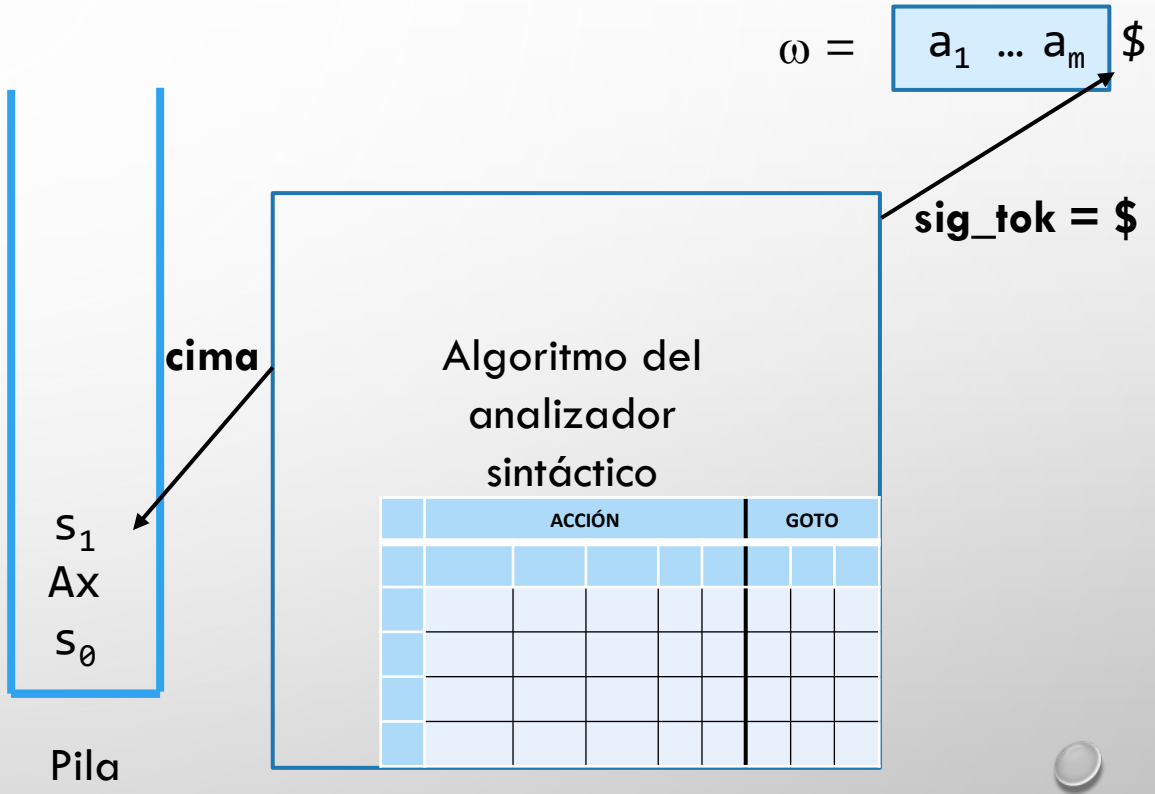
Las tablas tienen por filas los estados, y por columnas **símbolos gramaticales** y \$

	ACCIÓN					GOTO		
	t_1	t_2	t_3	...	\$	X_1	...	X_j
s_0	desplazar			...			...	
s_1				...	acept		...	
s_2		red $N_3 \rightarrow t_2$	red $N_2 \rightarrow \lambda$	...	...	**s_4**	...	**s_q**
...	...	...	...	...	...		...	...
s_q	red $N_p \rightarrow N_3\ t_1$	desplazar			...	**s_3**		...

Analizador Sintáctico Ascendente LR



Configuración inicial



Configuración final

Ejemplo de tabla de análisis de un A. St. Ascendente LR(1)

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

	ACCIÓN						GOTO								
	id	+	*	(	)	$	id	+	*	(	)	$	E	T	F
0	d			d			5			4			1	2	3
1		d				Acep		6							
2		r2	d		r2	r2			7						
3		r4	r4		r4	r4									
4	d			d			5			4			8	2	3
5		r6	r6		r6	r6									
6	d			d			5			4				9	3
7	d			d			5			4					10
8		d			d			6			11				
9		r1	d		r1	r1			7						
10		r3	r3		r3	r3									
11		r5	r5		r5	r5									

Tabla ACCIÓN

d: desplazar
r #: reducir por la regla #
Acep: aceptar
celda en blanco: error

Tabla GOTO

#: estado a apilar
(se necesita después de desplazar o de reducir)

Ejemplo de tabla de análisis de un A. St. Ascendente LR(1)

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

	ACCIÓN						GOTO		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Acep			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Tabla ACCIÓN

d #: desplazar **y apilar el estado #**
r #: reducir por la regla #
Acep: aceptar
celda en blanco: error

Tabla más compacta, con la misma información

Tabla GOTO

#: estado a apilar
(se necesita después de reducir)

Ejemplo de tabla de análisis de un A. St. Ascendente LR(1)

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

Desplazar el token "id" de la entrada a la pila y apilar el estado 5

	ACCIÓN						GOTO		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Acep			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Reducir por la regla 5

Tabla ACCIÓN

d #: desplazar **y apilar el estado #**
r #: reducir por la regla #
Acep: aceptar
celda en blanco: error

Tabla GOTO

#: estado a apilar
(se necesita después de reducir)

Tabla más compacta, con la misma información

La gramática es LR(1) puesto que **no hay más de una acción en cada celda**

Algoritmo del Analizador Sintáctico Ascendente LR

```
sig_tok := Alex()      /* sig_tok contiene el primer token de la cadena de entrada w$
repeat
{  /* Siendo s el estado de la cima de la pila y a el terminal almacenado en sig_tok, consultar ACCION[s,a]
   if ACCION[s,a]=desp s_i then
   {  push a              /* meter a en la pila
      push s_i            /* meter s_i en la pila
      sig_tok := Alex()      /*pedir el siguiente token al A. Léxico
   }
   else  if ACCION[s,a]=reducir por X → Y_1 Y_2 … Y_k
         {  for i=1 to 2*k do pop cima      /* sacar 2*k elementos de la pila (el
                                            /* consecuente y los estados intercalados)
            /* sea s_j el estado que está ahora en la cima de la pila
            push X     /* meter el antecedente en la pila
            s_k := GOTO[s_j,X]
            push s_k
         }
         else  if ACCION[s,a]=aceptar then return
               else error ()
}
```

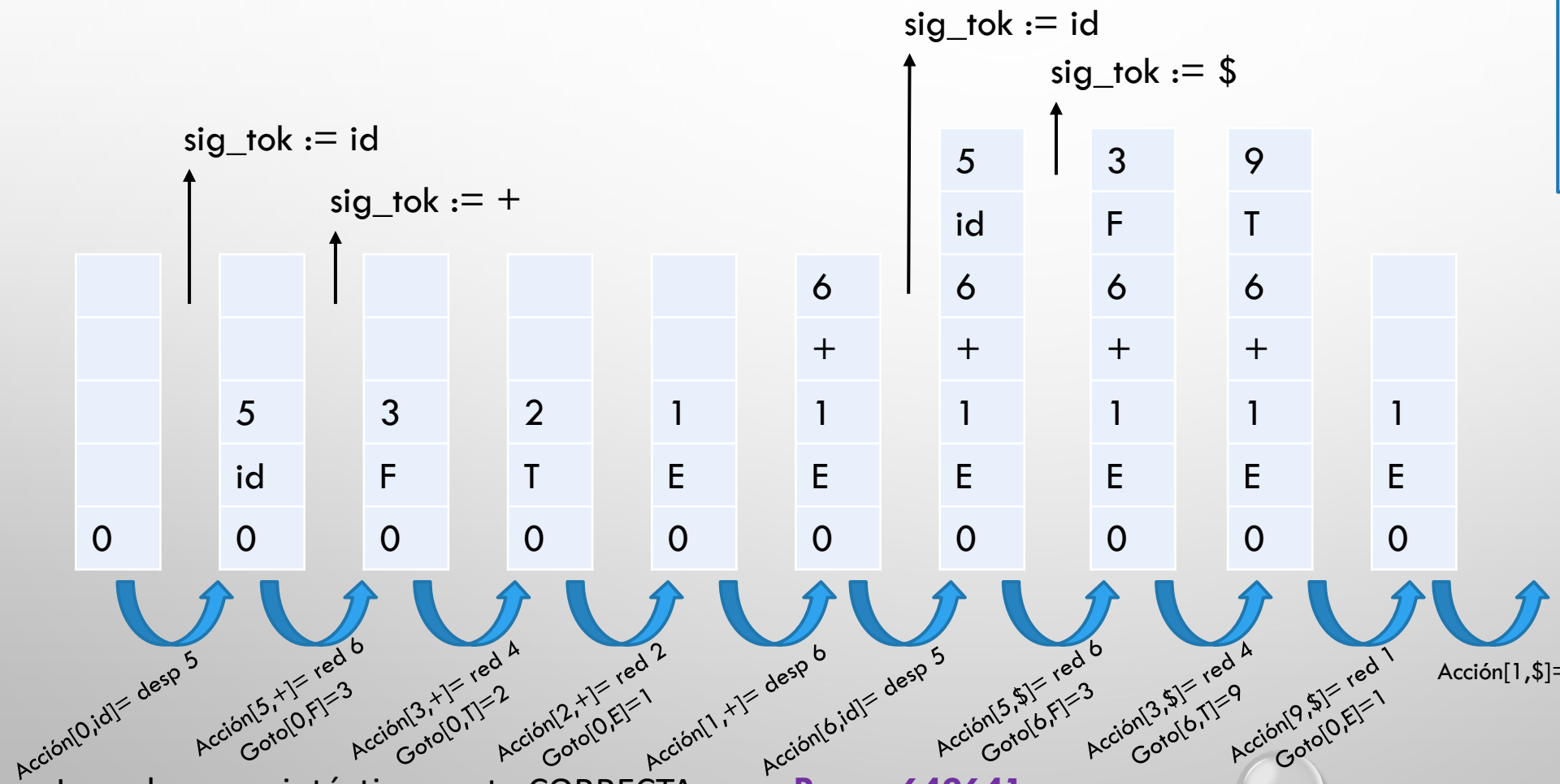
ANALIZADOR SINTÁCTICO **ASCENDENTE LR**

- EJEMPLOS DE ANÁLISIS DE CADENAS CORRECTAS E INCORRECTAS

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

1. Ejemplo de análisis de una cadena correcta

ω = `id + id` $



La cadena es sintácticamente CORRECTA

Parse: 642641

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

1. Ejemplo de análisis de una cadena correcta

ω = `id + id` $

Desplazamiento: apilar el token, apilar el estado y llamar al ALex()

sig_tok := id

sig_tok := +

sig_tok := id

sig_tok := $

					6	6	6	6	
					+	+	+	+	
	5	3	2	1	1	1	1	1	1
	id	F	T	E	E	E	E	E	E
0	0	0	0	0	0	0	0	0	0

Acción[0,id]= desp 5

Acción[5,+]= red 6
Goto[0,F]=3

Acción[3,+]= red 4
Goto[0,T]=2

Acción[2,+]= red 2
Goto[0,E]=1

Acción[1,+]= desp 6

Acción[6,id]= desp 5

Acción[5,$]= red 6
Goto[6,F]=3

Acción[3,$]= red 4
Goto[6,T]=9

Acción[9,$]= red 1
Goto[0,E]=1

Acción[1,$]= **ACEPTAR**

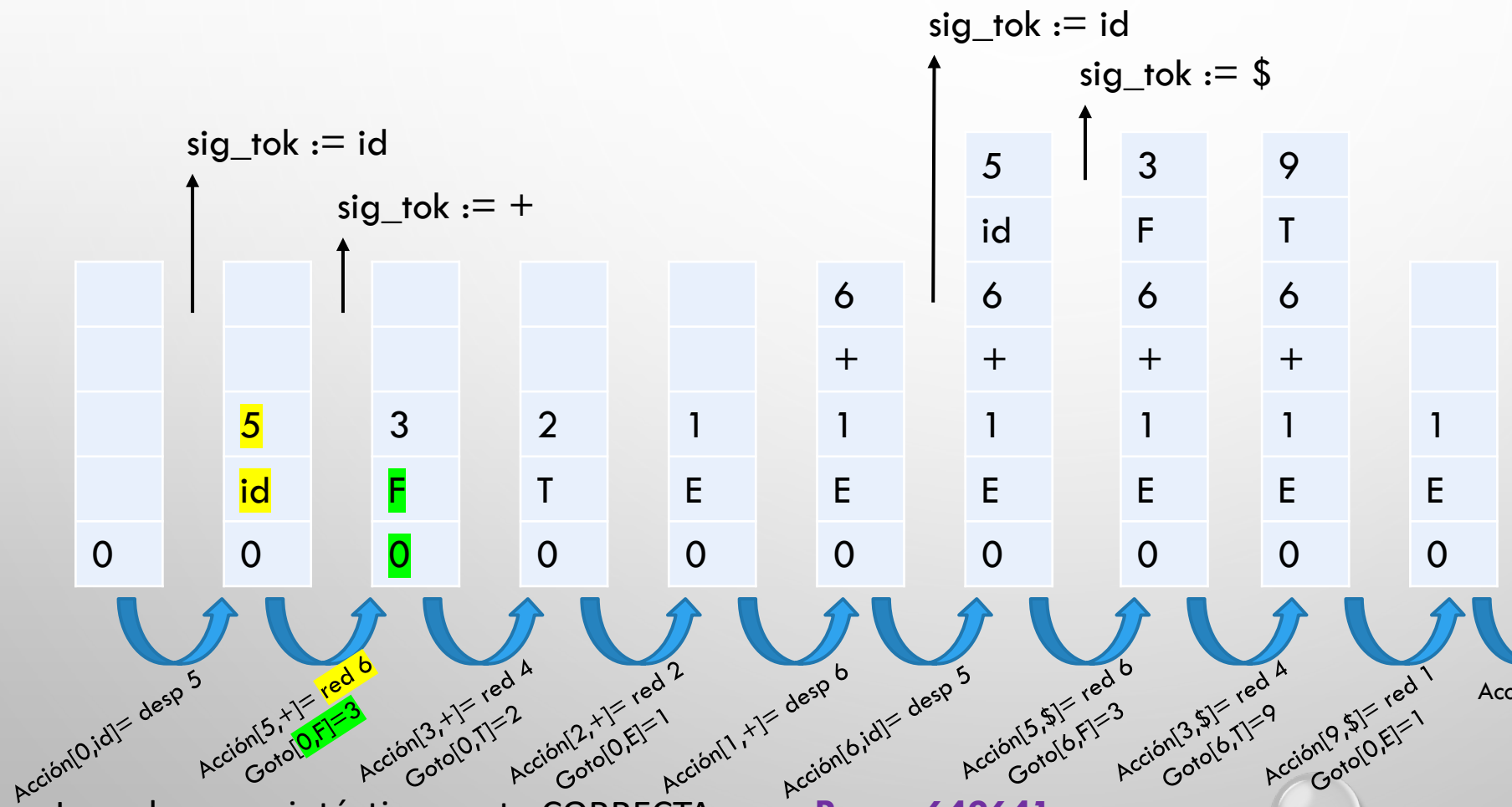
La cadena es sintácticamente CORRECTA

Parse: 642641

1. Ejemplo de análisis de una cadena correcta

$\omega =$ `id + id` $

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id



Reducción: sacar el consecuente (y estados intercalados). Queda en la cima un estado. El goto de ese estado y el antecedente que se apila, te da el nuevo estado a apilar

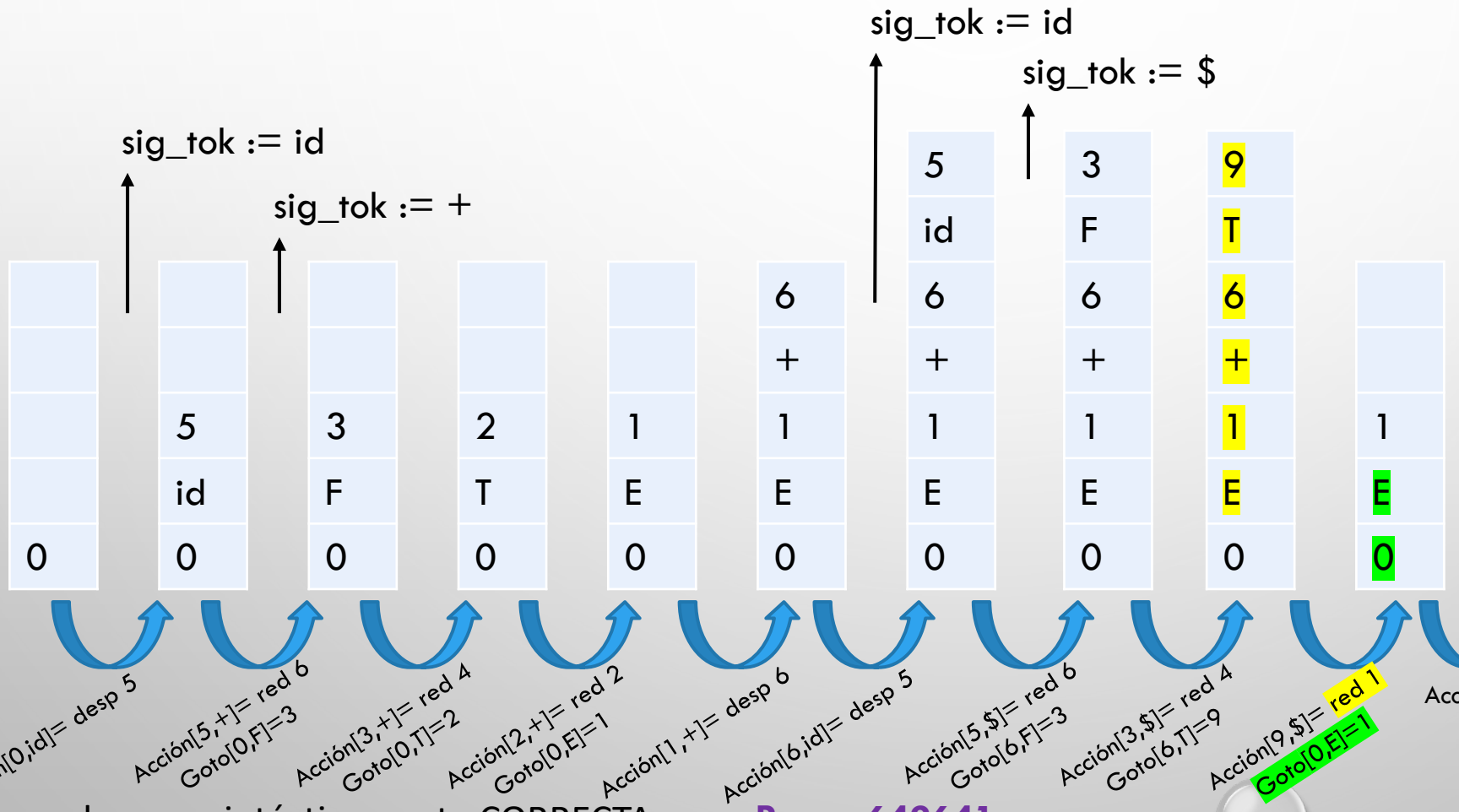
La cadena es sintácticamente CORRECTA

Parse: 642641

1. Ejemplo de análisis de una cadena correcta

$\omega =$ `id + id` $

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id



Reducción: sacar el consecuente (y estados intercalados). Queda en la cima un estado. El goto de ese estado y el antecedente que se apila, te da el nuevo estado a apilar

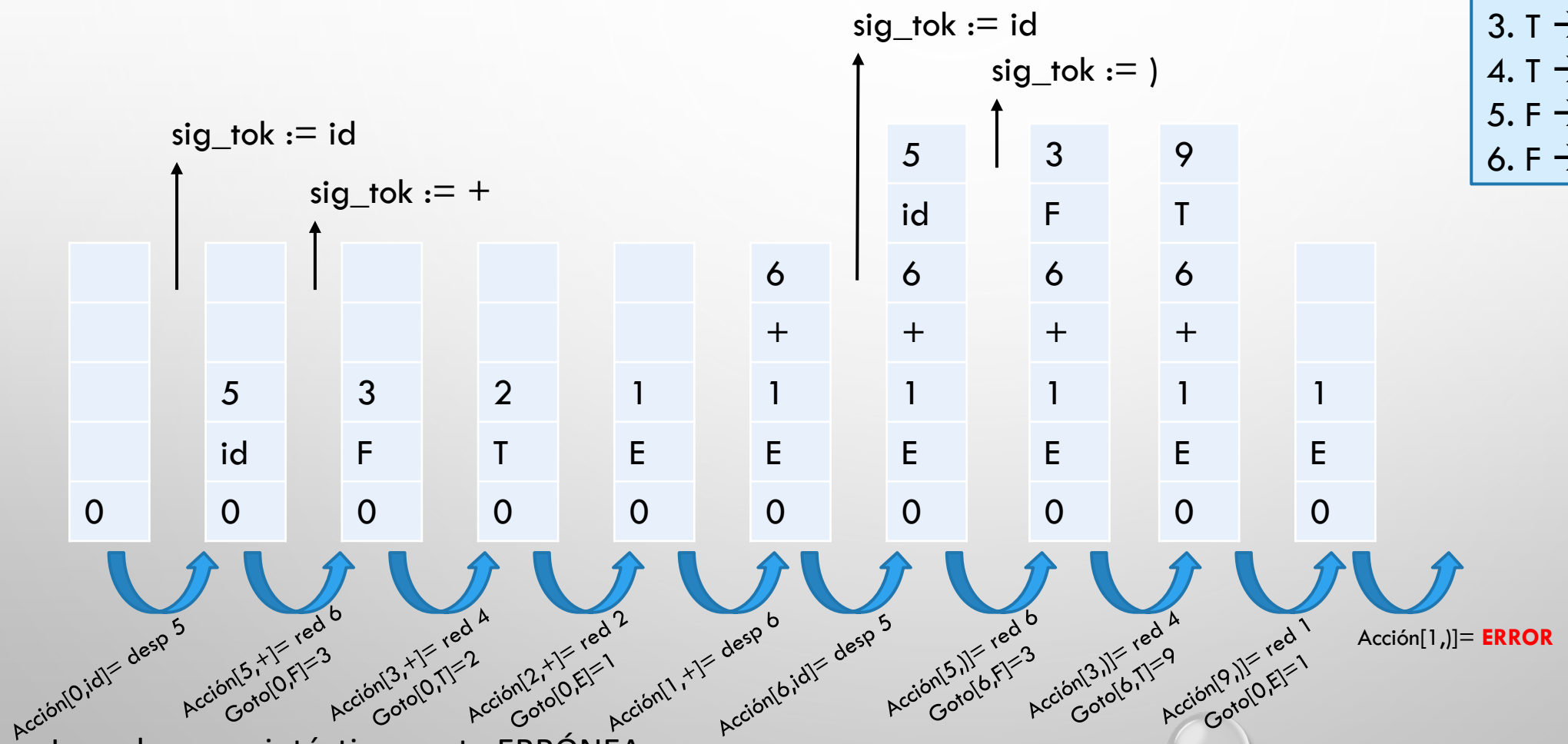
La cadena es sintácticamente CORRECTA

Parse: 642641

2. Ejemplo de análisis de una cadena errónea

$\omega =$ `id + id )` $

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

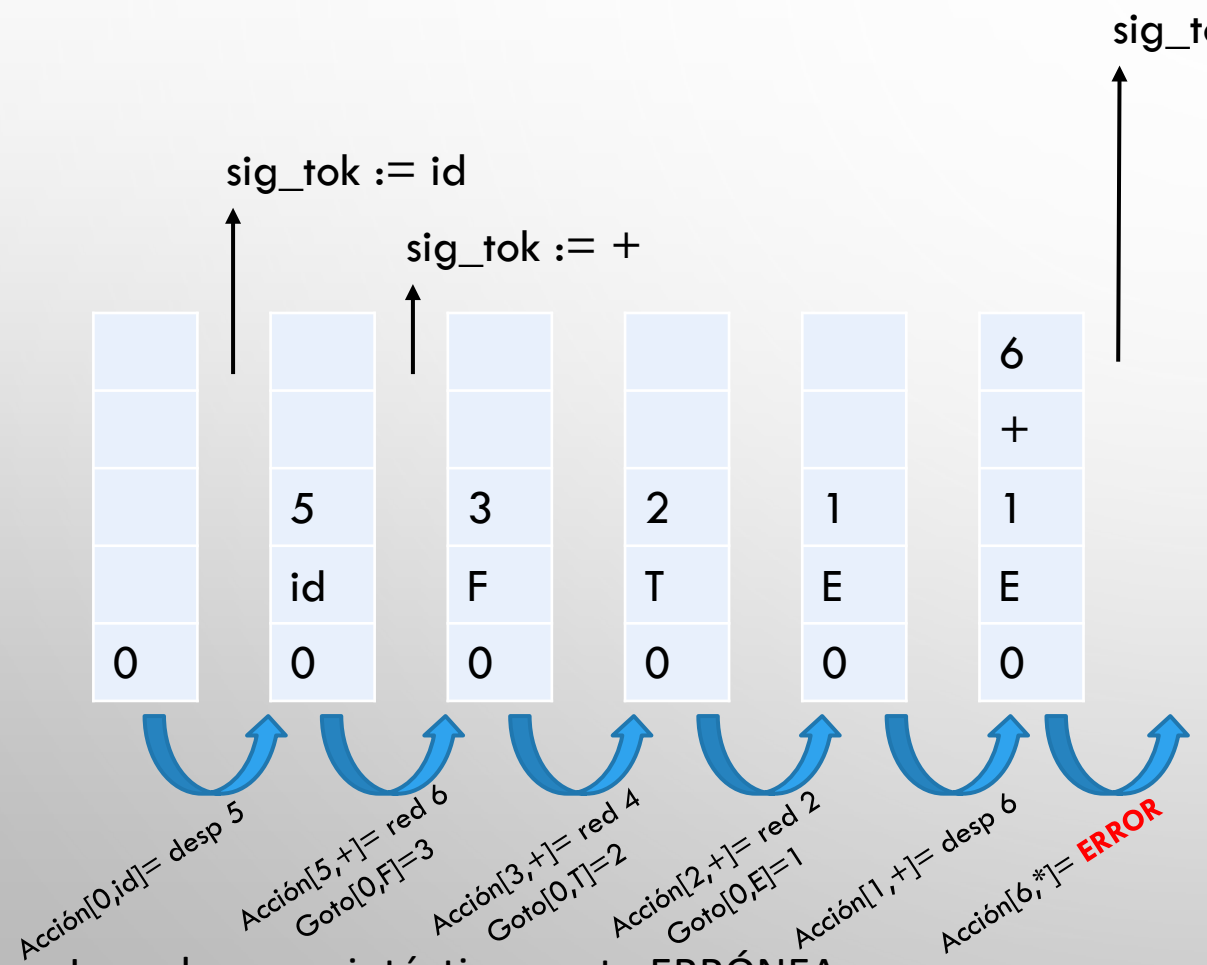


La cadena es sintácticamente ERRÓNEA

3. Ejemplo de análisis de una cadena errónea

$\omega =$ `id + * id` $

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id



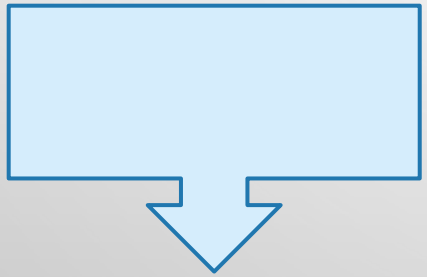
La cadena es sintácticamente ERRÓNEA

ANALIZADOR SINTÁCTICO **ASCENDENTE LR**

- MÉTODO DE CONSTRUCCIÓN DE LAS TABLAS ACCIÓN Y GOTO DE UN SIMPLE LR(1)

¿Qué se necesita para construir la tabla del Analizador?

1. Gramática Aumentada
2. Ítem LR(0)
3. Cierre de un conjunto de ítems LR(0)
4. Goto de un conjunto de ítems y un símbolo gramatical
5. Construcción de la colección canónica de ítems LR(0) para una gramática aumentada
6. Autómata Finito Determinista reconocedor de los prefijos viables de la gramática



7. Construcción de la tabla de análisis de un SLR(1)

¿Qué se necesita para construir la tabla del Analizador?

1. Gramática Aumentada

- Se añade un nuevo axioma (S') y la regla S' → S

2. Ítem LR(0)

- Como una regla gramatical pero con un punto (".") en alguna posición del consecuente. El punto separa, la parte del consecuente que ya se ha recibido, de la que se espera recibir para poder aplicar esa reducción.

Regla Gramatical	ítems posibles
A → X Y Z	A→ . X Y Z A → X . Y Z A → X Y . Z A → X Y Z .
A → λ	A → .

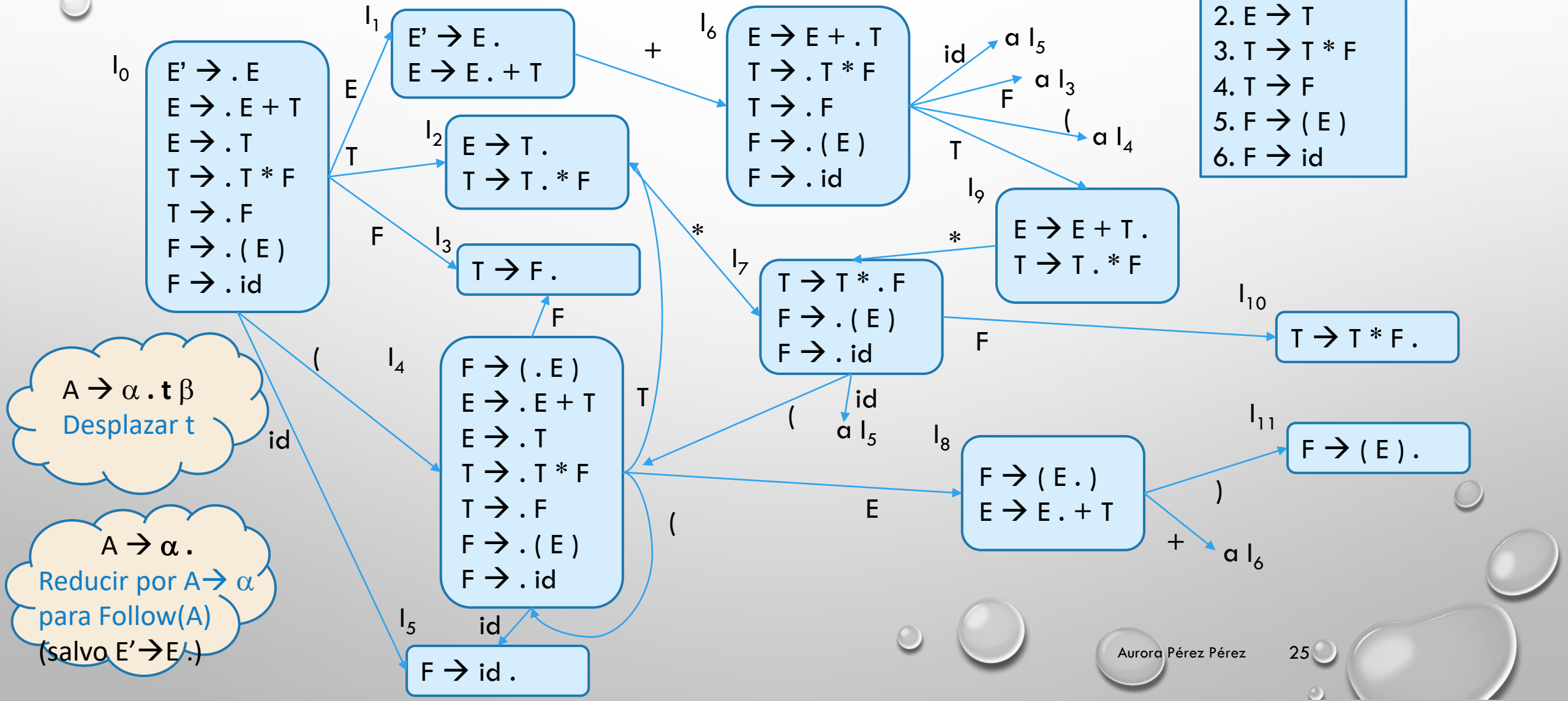
Indica que se puede aplicar la regla A → X Y Z

Indica que se puede aplicar la regla A → λ

¿Qué se necesita para construir la tabla del Analizador Ascendente LR?

E' → E .
Aceptar

6. AFD reconocedor de los prefijos viables de la gramática



A → α . t β
Desplazar t

A → α .
Reducir por A→ α para Follow(A)
(salvo E'→E .)

7. Tabla de análisis obtenida del AFD de los prefijos viables

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

	ACCIÓN						GOTO		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				Acep			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Tabla ACCIÓN

d #: desplazar y apilar el estado #
r #: reducir por la regla #
Acep: aceptar
celda en blanco: error

Tabla GOTO

#: estado a apilar
(se necesita después de reducir)

Algoritmo para la construcción de la tabla de análisis de un SLR(1)

Construir la colección canónica de ítems LR(0) para *G'*: $C = \{I_0, I_1, ..., I_n\}$

1. El estado *i* es el que se obtiene a partir de I_i. Las acciones para el estado *i* se determinan de la siguiente manera:

   a) Si $[A \rightarrow \alpha \,.\, a\, \beta]$ está en I_i y $goto(Ii,a)=Ij$ entonces $ACCION[i,a]$ = "***desplazar*** *j*". En este caso, *a* debe ser un terminal, y *j* es el estado que se apila tras el símbolo desplazado.

   b) Si $[A \rightarrow \alpha \,.]$ está en I_i entonces $ACCION[i,a]$ = "***reducir*** *por* $A \rightarrow \alpha$", y esto para todo terminal *a* perteneciente a *FOLLOW(A)*. En este caso, *A* es cualquier no terminal excepto S'.

   c) Si $[S' \rightarrow S \,.]$ está en I_i entonces $ACCION[i,\$]$ = "***aceptar***".

2. Si estas reglas generan conflicto en alguna casilla, se dice que la gramática no es SLR(1). En tal caso, no es posible construir un analizador sintáctico mediante este algoritmo.

3. Las transiciones *GOTO* del estado *i* se construyen para todos los no terminales *A* mediante la regla: si $goto(I_i,A)=I_j$ entonces $GOTO[i,A] = j$.

4. Las casillas en blanco y corresponden a los casos de "***error***".

5. El estado inicial del analizador es el construido a partir del conjunto de ítems que contiene a $[S' \rightarrow .\ S]$.

¿Cómo comprobar si una gramática es o no LR(1)?

- **OPCIÓN 1.** Construir la tabla ACCIÓN del analizador y ver que no hay más de una acción en cada celda ← se necesita mucho tiempo
- **OPCIÓN 2.** Realizar el estudio de los posibles conflictos sobre el AFD reconocedor de los prefijos viables ← más rápido que la opción 1
  - Conflicto: cuando en un mismo estado, diferentes ítems válidos me indican diferentes acciones. Conflictos posibles:
    - Reducción/reducción

      $A \rightarrow \alpha .$
      $B \rightarrow \beta .$
      ...

      Red por $A \rightarrow \alpha$ para Follow(A)
      Red por $B \rightarrow \beta$ para Follow(B)

      Si Follow(A) $\cap$ Follow(B) $\neq \varnothing$ tendremos un **conflicto red/red**

      (por supuesto lo hay si tenemos dos ítems $A \rightarrow \alpha .$ y $A \rightarrow \beta .$)
    - Reducción/desplazamiento

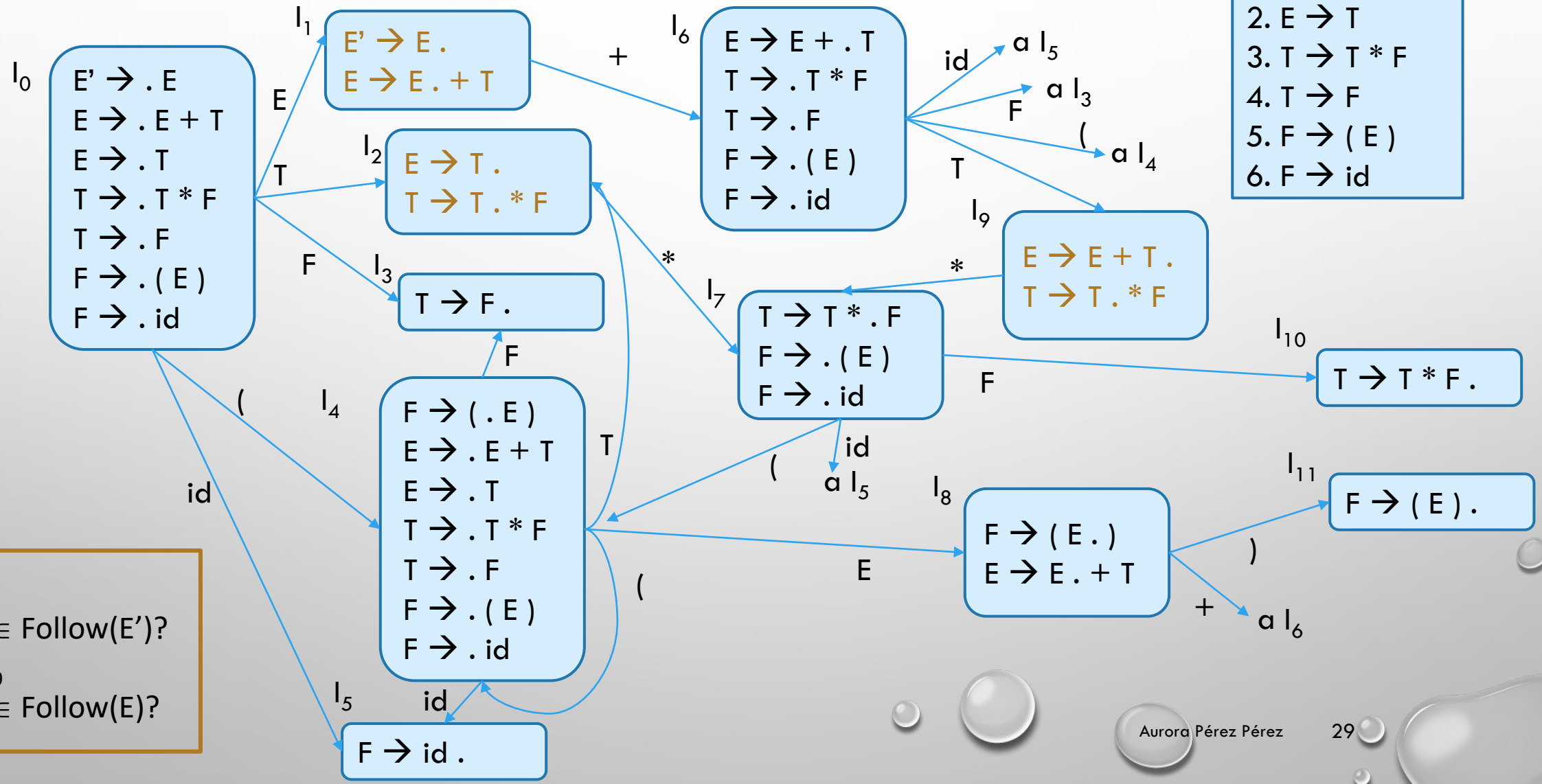
      $A \rightarrow \alpha .$
      $B \rightarrow \beta . t \gamma$
      ...

      Red por $A \rightarrow \alpha$ para Follow(A)
      Desplazar t

      Si $t \in$ Follow(A) tendremos un **conflicto red/despl**

¿Es LR(1) la gramática? Estudio de los posibles conflictos

0. E' → E
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id



I_1
¿+ ∈ Follow(E')?
$I_2 = I_9$
¿* ∈ Follow(E)?

¿Qué se necesita para construir la tabla del Analizador? (Continuamos con la teoría)

3. Cierre de un conjunto de ítems LR(0)

Si *I* es un conjunto de ítems LR(0) de una gramática *G*, entonces *cierre(I)* es el conjunto de ítems construido a partir de *I* mediante las dos siguientes reglas:

1. Inicialmente, todos los elementos de *I* se añaden a *cierre(I)*
2. Si $A \rightarrow \alpha \,.\, B\beta$ está en *cierre(I)* y $B \rightarrow \gamma$ es una producción de *G*, entonces el ítem $B \rightarrow .\, \gamma$ se añade a *cierre(I)* si todavía no está. Se sigue aplicando esta regla 2 hasta que no se puedan añadir más ítems nuevos a *cierre(I)*.

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

I = {F → (. E)}

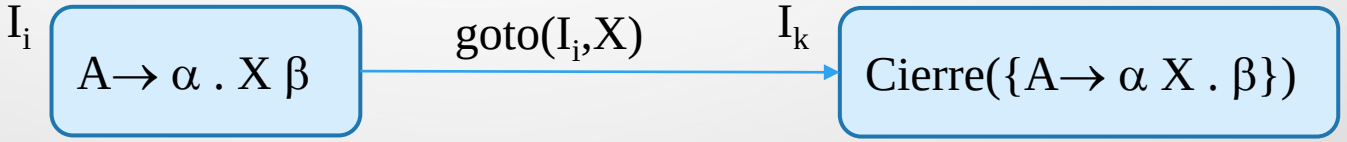
cierre (I) = {
F → (. E) — Punto 1
E → . E + T — Punto 2
E → . T
T → . T * F — Punto 2
T → . F
F → . (E) — Punto 2
F → . id
}

¿Qué se necesita para construir la tabla del Analizador? (Continuamos con la teoría)

4. Goto de un conjunto de ítems y un símbolo gramatical

Se define *goto(I,X)* como el cierre del conjunto de todos los ítems $[A \rightarrow \alpha X . \beta]$ tales que $[A \rightarrow \alpha . X \beta]$ esté en *I*.

Intuitivamente, si *I* es el conjunto de ítems válidos para algún prefijo viable γ, *goto(I,X)* es el conjunto de ítems válidos para el prefijo viable γX.



1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

I_4 = { F → (. E)
E → . E + T
E → . T
T → . T * F
T → . F
F → . (E)
F → . id
}

goto (I_4, E) = cierre({F → (E .), E → E . + T})
= { F → (E .)
E → E . + T
}
= I_8

¿Qué se necesita para construir la tabla del Analizador? (Continuamos con la teoría)

5. Construcción de la colección canónica de ítems LR(0) para una gramática aumentada

function coleccion (*G'*);
{
C:= {cierre({[S'→ . S]})}
repeat
for cada conjunto de ítems *I* en *C* y cada símbolo gramatical *X* tal que *goto(I,X)* no esté vacío y no esté ya en *C*
do añadir *goto(I,X)* a *C*
until no se puedan añadir más conjuntos de elementos a *C*
}

- Cada elemento de la colección es un conjunto de ítems
- El primer elemento (I_0) se obtiene como el cierre del conjunto que tiene solo el ítem S' → . S Los restantes elementos se obtienen como el goto de un conjunto de ítems previo y un símbolo gramatical.
- La colección canónica de ítems LR(0) se corresponde con los estados de un Autómata Finito Determinista capaz de reconocer todos los prefijos viables de la gramática.
- De este autómata (o de la colección canónica) se obtienen las tablas ACCION y GOTO del analizador sintáctico.

Colección canónica representada como AFD de los prefijos viables