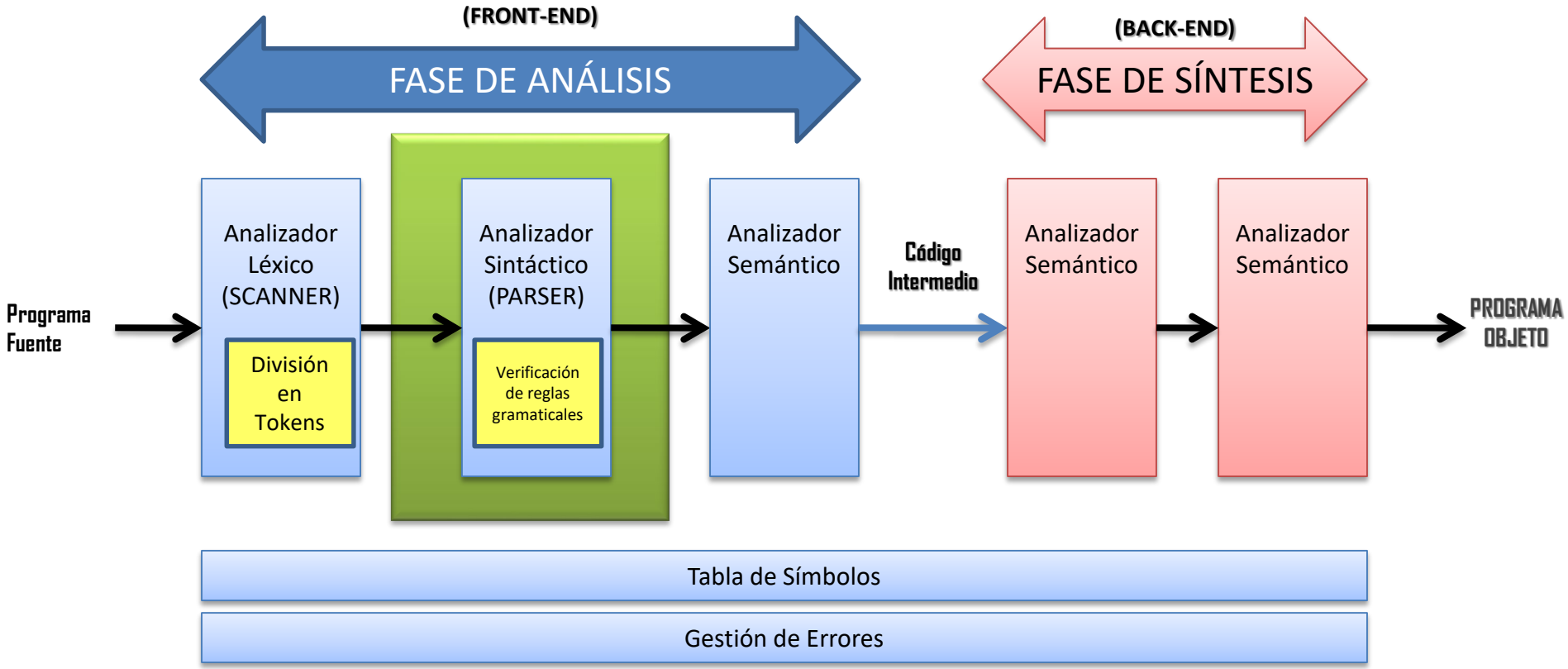
Sesión 6
Analizador Sintáctico
Análisis Sintáctico Ascendente
LR, LR(0), SLR(1), LALR(1)

Antonio Moratilla Ocaña

Resumen del tema

- Objetivo:
  - Estudiar cómo se produce el análisis ascendente de una expresión para una gramática, tanto con backtracking como predictivo.

Posición en el diagrama



Retomando el hilo...

- Ya sabemos
  - Que el análisis descendente es un método intuitivo para la construcción de un AST
- Lo que queremos saber
  - ¿Existen métodos ascendentes que en vez de partir del axioma partan de las reglas a aplicar por los tokens?
  - ¿Cómo funcionan?

Resumen del tema

Introducción

- Análisis sintáctico ascendente
- Analizadores ascendentes tipo LR
- Analizador LR(0)
- Analizador SLR(1)
- Analizador LALR(1)

Tipos de analizador sintáctico

- **Analizadores sintácticos descendentes** (Top-down)
  - Construyen el árbol sintáctico de la raíz (arriba) a las hojas (abajo).
  - Parten del símbolo inicial de la gramática (*axioma*) y van expandiendo producciones hasta llegar a la cadena de entrada.
- **Analizadores sintácticos ascendentes** (Bottom-up)
  - Construyen el árbol sintáctico comenzando por las hojas.
  - Parten de los terminales de la entrada y mediante reducciones llegan hasta el símbolo inicial.

En ambos casos se examina la entrada de izquierda a derecha, analizando los testigos o tokens de entrada de uno en uno.

Tipos de analizador sintáctico

- El **Análisis Sintáctico Ascendente** es una técnica de análisis sintáctico que intenta comprobar si una cadena **x** pertenece al lenguaje definido por una gramática L(G) aplicando los siguientes criterios
  - Partir de los **elementos terminales** de la frase **x** .
  - Escoger reglas gramaticales estratégicamente.
  - Aplicar a la inversa derivaciones por la derecha. (Right Most Derivation)
  - Procesar la cadena de izquierda a derecha.
  - Intentar alcanzar el axioma para obtener el árbol de análisis sintáctico o error.

Analizadores Sintácticos Ascendentes

- Comprueban si una cadena de símbolos pertenece a un lenguaje aplicando reducciones sobre la entrada hasta alcanzar el axioma de la gramática.
- En general son más potentes que los métodos descendentes.
  - Pueden manejar gramáticas recursivas a izquierdas. (Siempre comienzan por un terminal, no se puede entrar en bucle en el análisis)
- Los algoritmos de implementación son más complejos que los de los métodos ascendentes.
  - La implementación “a mano” de estos algoritmos es demasiado compleja por lo que generalmente se utilizan generadores automáticos.

Analizadores Sintácticos Ascendentes

- Ejemplo

Reglas de producción

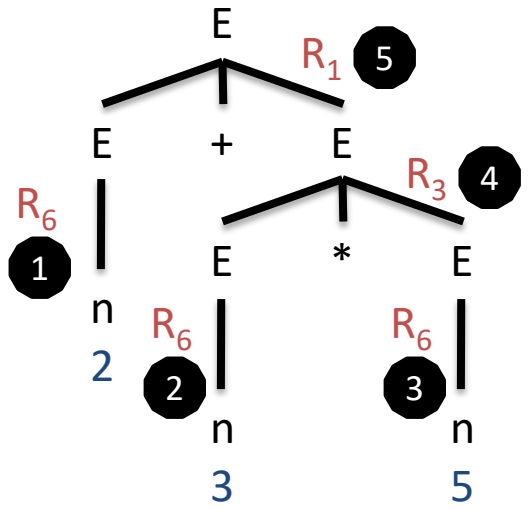
R_1 E → E + E
R_2 E → E -E
R_3 E → E * E
R_4 E → E / E
R_5 E → (E)
R_6 E → n

Cadena de derivación

n + n * n
E + n * n ← R_6
E + E * n ← R_6
E + E * E ← R_6
E + E ← R_3
E ← R_1

El análisis ascendente genera una cadena de derivación por la derecha leída en sentido inverso.

Árbol de análisis sintáctico



Se parte la cadena de entrada leída de izquierda a derecha y se aplican reglas a la inversa para intentar alcanzar el axioma

CONFLICTOS

Reducción -Desplazamiento
El analizador puede tanto aplicar un paso de reducción como uno de desplazamiento

Reducción –Reducción
El analizador puede aplicar dos reglas distintas para realizar diferentes pasos de reducción

Analizador Sintáctico - Implementación

Dos formas de implementar los Analizadores Sintácticos Ascendentes:

- **Analizadores con retroceso**:
  - Prueban todas las posibilidades con un algoritmo de *backtracking.*
  - Fáciles de implementar.
  - Los mismos defectos que el analizador descendente equivalente.
- **Analizadores predictivos**:
  - Utilizan información de la cadena de entrada para predecir qué subcadena lleva al éxito.

Analizador Sintáctico Ascendentes

IMPLEMENTACIÓN

Los Analizadores Sintácticos Ascendentes son capaces de decidir qué regla de producción aplicar a cada paso en función de los elementos terminales que encuentran en la lectura de la cadena de entrada.

Estos analizadores son llamados genéricamente analizadores **LR(K)** o analizadores por Desplazamiento-Reducción

Arquitectura general de análisis sintáctico ascendente

Todos los tipos de analizadores sintácticos ascendentes el modelo de arquitectura consta de tres elementos: una pila de estados, una tabla de acciones y una tabla Ir-A.

Analizador Sintáctico Ascendentes

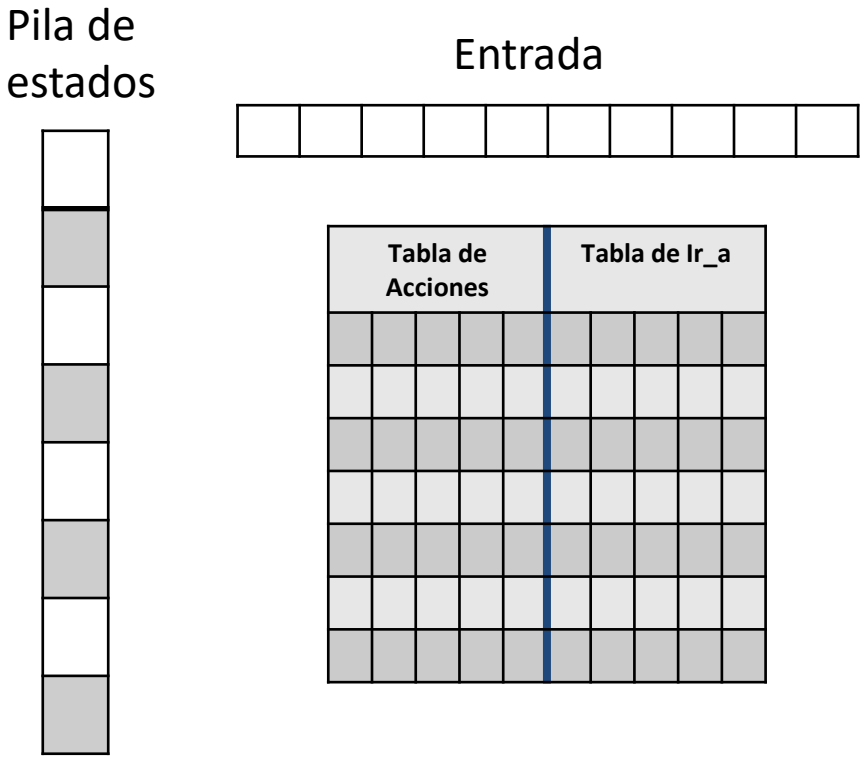
Arquitectura General

- **Pila de estados** El analizador sintáctico puede atravesar, a lo largo del proceso de análisis, distintos estados. En cada momento el estado en curso aparece en la cima de la pila.

- **Tabla de acciones** Para cada estado y cada terminal (ó $) a la entrada el analizador tiene información de que acciones debe realizar. Las operaciones posibles son:
  - Desplazar y apilar el estado,
  - Reducir por la regla r,
  - Emitir un error y
  - Aceptar la cadena de entrada.

- **Tabla Ir-A** En el caso de aplicar Reducción, y del estado en curso, esta tabla indica el estado que hay que apilar en función del antecedente de la regla de producción aplicada.

La diferencia entre los tipos de analizadores sintácticos estriba en cómo se construyen estas tablas y en el tipo de gramáticas (lenguajes) que admiten.



LA PILA EN LOS ANALIZADORES PREDICTIVOS

- Un analizador ascendente utiliza una pila similar a la utilizada por el analizador LL(1), que incluye:
  - Símbolos terminales y no terminales.
  - Información de estado.
- Estado de la pila:
  - Al comenzar el análisis la pila está vacía.
  - Al final de un análisis con éxito contiene el símbolo inicial.
- Dos posibles acciones además de *"éxito"*:
  - **Desplazar**: pasa un terminal de la entrada a la cima de la pila.
  - **Reducir**: sustituye una cadena en la pila por un no-terminal.

Analizador Sintáctico Ascendentes Predictivos

1.- LR(*k*)

- Leen la entrada de izquierda a derecha (Left to rigth)
- Aplican derivaciones por la derecha para cada entrada (Right)
- Utilizan k componentes léxicos de búsqueda hacia adelante.

2.- SLR(1)

- LR(1) Sencillo.
- Mejora el LR(1) pero hace uso de la búsqueda hacia delante.

3.- LALR(1)

- Look Ahead LR(1): Análisis sintáctico de búsqueda hacia adelante
- Más potente que SLR(1) y más sencillo que LR(1).
  - Es un compromiso entre SLR(1) y LR(1)

Se puede generar un analizador sintáctico ascendente predictivo para prácticamente todas las gramáticas de libre contexto.

Inconveniente: requieren mucho más trabajo de implementación que los analizadores descendentes.

Analizador Sintáctico - Ascendentes Predictivos

EJEMPLO

Gramática

S → A

A → A a b | b

Entrada

bab

Pila	Entrada	Acción
$	**bab**$	desplazar
$b	ab$	reducir A → b
$A	ab$	desplazar
$Aa	b$	desplazar
$Aab	$	reducir A → A a b
$A	$	reducir S → A
$S	$	exito

Analizador Sintáctico - Ascendentes Predictivos

LA PILA EN LOS ANALIZADORES PREDICTIVOS

- Análisis del ejemplo anterior:
  - Con un mismo valor en la pila "$A" se han realizado acciones diferentes:
    - "desplazar"
    - "reducir S → A"
  - El analizador desplaza terminales hasta que es posible aplicar una reducción, pero esto no implica retrocesos.
  - Algunas reducciones obligan a mirar no sólo la cima de la pila, sino más elementos: "reducir A → A a b".

Conclusión

Es necesaria una tabla de análisis que permita hacer predicciones.

Gramática
S → A
A → A a b | b

Entrada
bab

Pila	Entrada	Acción
$	bab$	desplazar
$b	ab$	reducir A → b
$A	ab$	**desplazar**
$Aa	b$	desplazar
$Aab	$	**reducir** A → A a b
$A	$	reducir S → A
$S	$	exito

Recordatorio

Reducción – Desplazamiento : El analizador puede tanto aplicar un paso de reducción como uno de desplazamiento.

Reducción – Reducción : El analizador puede aplicar dos reglas distintas para realizar diferentes pasos de reducción.

Análisis Sintáctico LR(0)

Conceptos

Elemento LR(0):

- Regla de producción con una **posición distinguida** (por un punto en su lado derecho) S → A.ab
- Un elemento registra un paso intermedio en el reconocimiento del lado derecho de una regla.

A → xy (regla)
- A →.xy (elemento LR(0) inicial)
- A → x.y (elemento)
- A → xy. (elemento LR(0) final)

Elementos iniciales:

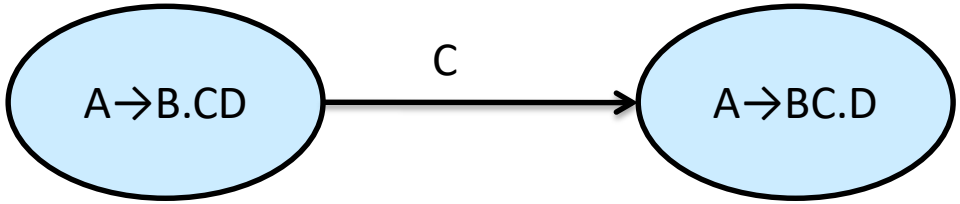
- Los que tienen la forma A → .XY

Elementos completos o finales:

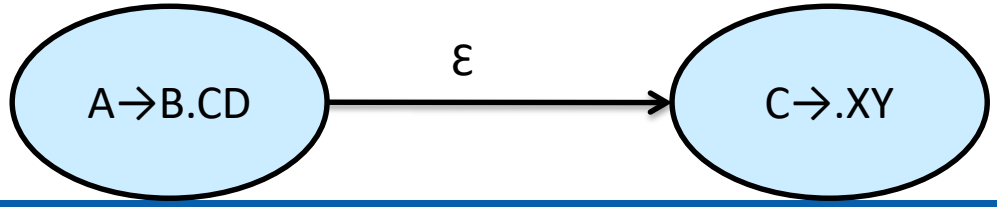
- Los que tienen la forma A → XY.

Análisis Sintáctico LR(0)

- Los elementos LR(0) se pueden emplear como estados de un autómata finito que mantienen información sobre:
  - La pila de análisis sintáctico.
  - El progreso del análisis.
- Construcción del **autómata finito no determinista (AFND)**:
  - Para todo elemento A→B.CD con BCD terminales o no-terminales, existe una transición para el símbolo C :



  - Para todo C (no-terminal) en una producción A → B.CD y para toda regla con C en la parte izquierda (C →.XY) habrá una transición:



Análisis Sintáctico LR(0)

- **Estado inicial**:
  - En este estado la pila está vacía y falta por reconocer un estado S.
  - En principio cualquier elemento inicial S→.X podría servir.
  - Si hay más de uno ¿cuál elegir?

    SOLUCION: Ampliar la gramática con una producción S' → S
- **Estado final**:
  - El autómata no tiene estados específicos "de aceptación".
  - El propósito del autómata es el seguimiento del análisis, no el reconocimiento directo de las cadenas de entrada.
  - El autómata, por tanto, tiene información acerca de la aceptación pero no estados de aceptación.

- El analizador LR(0) trabaja con una **Gramática Aumentada**.

Análisis Sintáctico LR(0)

EJEMPLO

Para una gramática

$S \rightarrow (S)\ S$

$S \rightarrow \varepsilon$

Tenemos los siguientes elementos

$S' \rightarrow .S$

$S' \rightarrow S.$

$S \rightarrow .(S)S$

$S \rightarrow (.S)S$

$S \rightarrow (S.)S$

$S \rightarrow (S).S$

$S \rightarrow (S)S.$

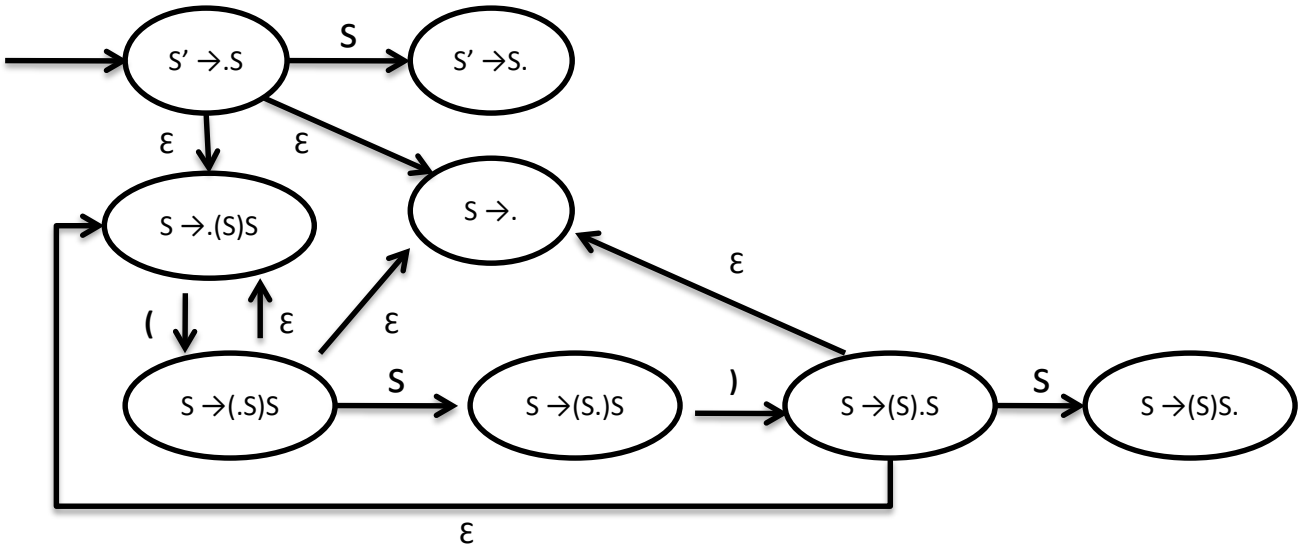
$S \rightarrow .$

Análisis Sintáctico LR(0)

EJEMPLO

El autómata finito no determinista es:

S → (S) S | ε



S' →.S

S' → S.

S →.(S)S

S →(.S)S

S →(S.)S

S →(S).S

S →(S)S.

S → .

Análisis Sintáctico LR(0)

Es posible construir un autómata finito determinista a partir del autómata no determinista mediante la cerradura de cada estado nuclear.

- **Elementos de núcleo**: los que se agregan a un estado tras una transición que no es vacía (transición no Ɛ).
- **Elementos de cerradura**: los que se agregan a un estado tras una transición vacía (transición Ɛ).
- Para cada estado nuclear (sólo hay núcleo) se calcula la cerradura:
  - Para todos aquellos elementos que tengan un no terminal a la derecha del punto, a ese no terminal le llamaremos **símbolo de cerradura**.
  - Se añaden al conjunto cerradura, y por tanto al estado, todas los elementos LR(0) iniciales correspondientes al símbolo de cerradura. Por tanto, si añadimos A→XY lo que se añade es el elemento: A→.XY

Análisis Sintáctico LR(0)

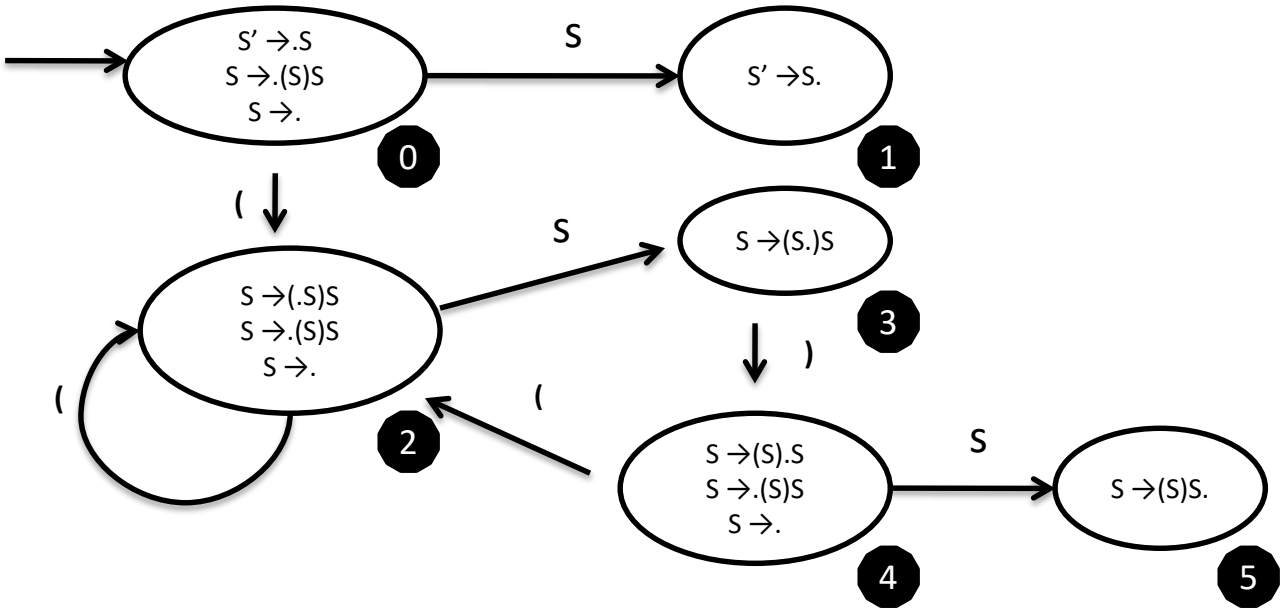
ALGORITMO DE CONSTRUCCIÓN DEL AUTÓMATA

1. Se construye el núcleo del estado inicial con **S'→.S**
2. Para cada **núcleo** nuevo:
   a) Se hace la cerradura del núcleo
   b) Se repite la cerradura para los nuevos elementos añadidos hasta que no se añadan nuevos elementos al estado y se considera que el estado está completo.
3. Para cada estado completo **n** :
   a) Se crean nuevos núcleos de estados a partir de **n** mediante las transiciones no **ε**. (Las transiciones ε ya no se realizan porque están representadas por la cerradura).
      - Si el núcleo creado ya existe en otro estado no se añade y la transición se dirige al estado existente.
   b) Se realizan las acciones descritas en el punto 2
   c) Se repite el punto 3 hasta que no se puedan añadir nuevos estados al autómata.

Análisis Sintáctico LR(0)

EJEMPLO

El autómata finito determinista del ejemplo anterior es:



$S \rightarrow (S)\ S \mid \varepsilon$

S' →.S

S' → S.

S →.(S)S

S →(.S)S

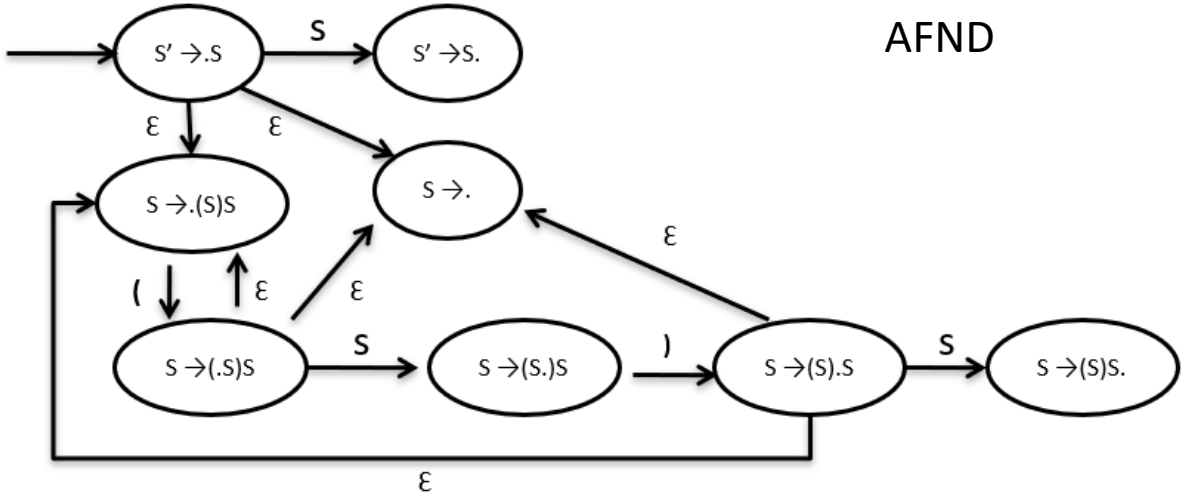
S →(S.)S

S →(S).S

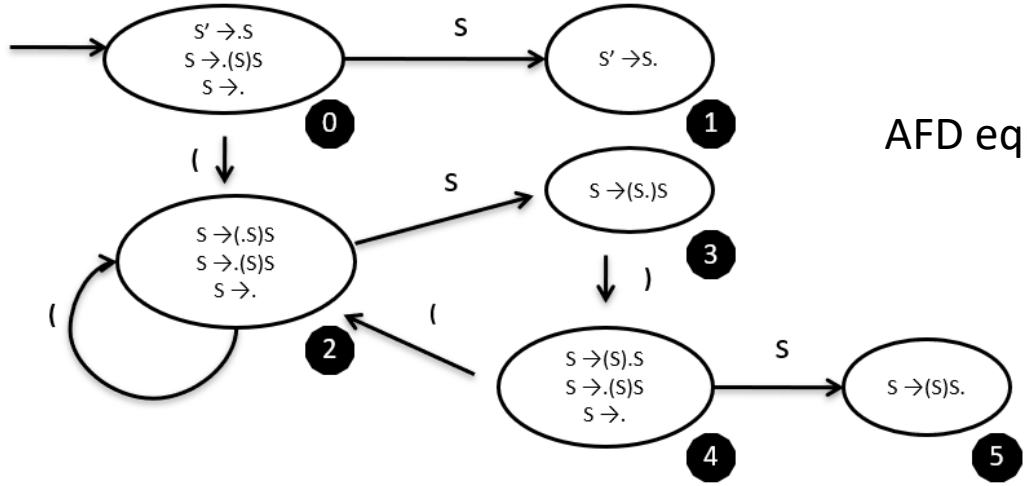
S →(S)S.

S → .

Análisis Sintáctico LR(0)



AFND



AFD equivalente

S → (S) S | ε

S' →.S
S' → S.
S →.(S)S
S →(.S)S
S →(S.)S
S →(S).S
S →(S)S.
S → .

Análisis Sintáctico LR(0)

TABLA DE ANÁLISIS SINTÁCTICO

- Se puede generar la tabla de análisis sintáctico a partir del autómata finito determinista (AFD).
- **Estados del autómata** LR(0):
  - De "desplazamiento" (sin elementos completos)
  - De "reducción" (un elemento completo)
- Las filas de la tabla se etiquetan con los estados.
- En cuanto a las columnas, 2 bloques:
  - Parte de "**Entrada**" (V_t) se indican acciones de desplazamiento, reducción o aceptación
  - Parte "**Ir_a**" con transiciones de no-terminales indican transiciones entre estados.
- Las entradas vacías de la tabla representan errores.

Análisis Sintáctico LR(0)

EJEMPLO

Gramática

A' → A

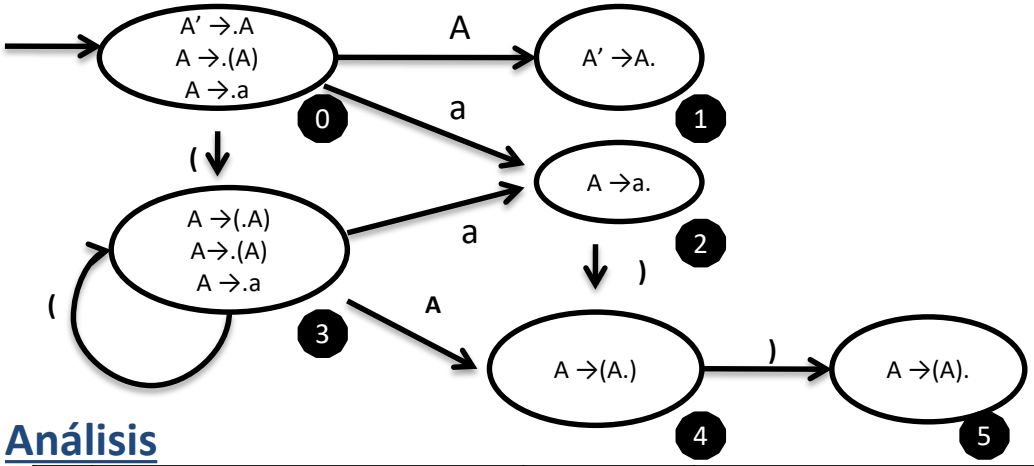
A → (A) | a

Entrada

((a))

Tabla

Estado	Acción	Regla	Entrada			Ir_a
			(	a	)	A
0	D		3	2		1
1	R	A' →A				
2	R	A → a				
3	D		3	2		4
4	D				5	
5	R	A →(A)				



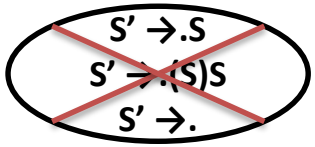
Análisis

#	Pila de Análisis Sintáctico	Entrada	Acción
1	$0	**((a))**$	desplazar
2	$0(3	(a))$	desplazar
3	$0(3(3	a))$	desplazar
4	$0(3(3a2	))$	reducir A → a
5	$0(3(3A4	))$	desplazar
6	$0(3(3A4)5	)$	reducir A → (A)
7	$0(3A4	)$	desplazar
8	$0(3A4)5		reducir A → (A)
9	$0A1		aceptar

Análisis Sintáctico LR(0)

Conflictos

- Si un estado contiene un elemento completo (**A→α.**) :
  - Si además tiene un elemento "no completo" se produce un conflicto de **reducción - desplazamiento**.
  - Si además contiene otros elementos completos (**B →β.**) hay un conflicto de **reducción - reducción**.
- Una gramática es LR(0) si y sólo si cada estado es un estado de desplazamiento (contiene únicamente elementos de desplazamiento) o un estado de reducción (contiene únicamente un elemento completo).
  - Esto es, en LR(0) no puede haber estados mixtos.



Análisis Sintáctico SLR(1)

Análisis sintáctico LR(1) Simple o **SLR(1)**

- Capaz de analizar prácticamente todas las construcciones que se presentan en un lenguaje
  - No obstante, hay situaciones en las que no sirve y por tanto no es analizable.
- Un estado puede tener tanto reducciones como desplazamientos.
- Utiliza el autómata finito determinista del LR(0) con mayor potencia, empleando el *token siguiente en la entrada para guiar sus acciones:*
  - Consulta el *token de entrada antes de cada desplazamiento para asegurar* que existe una transición adecuada
  - Utiliza el conjunto *siguiente* de un *no terminal* para decidir si se debería realizar una reducción o no.

Análisis Sintáctico SLR(1)

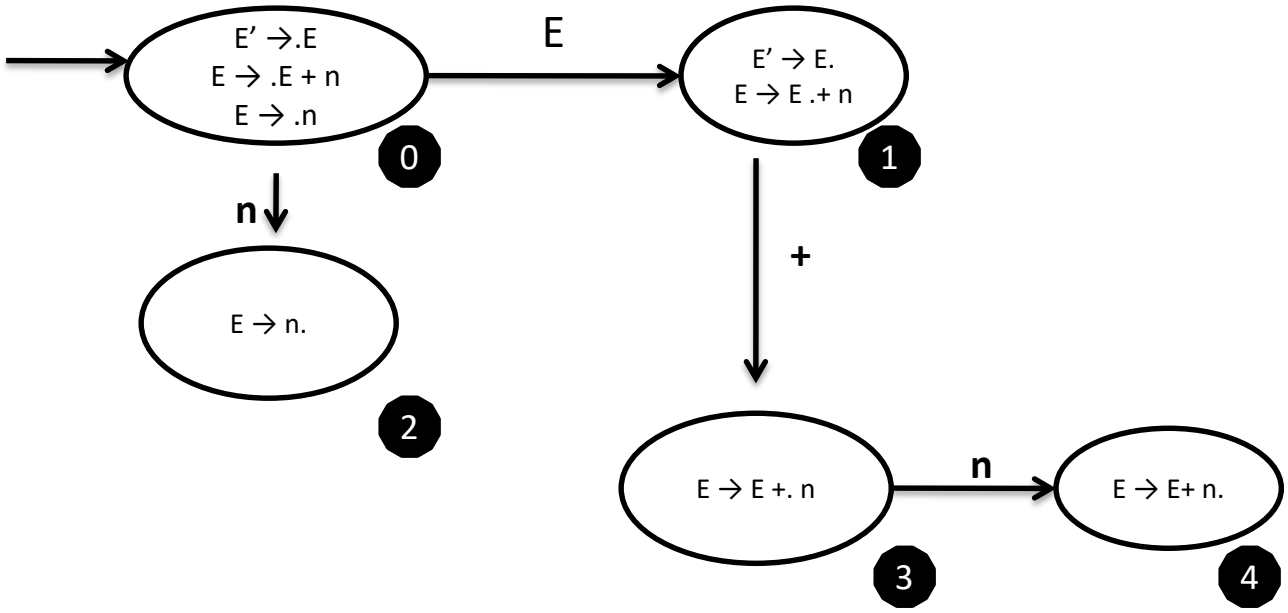
Construcción de la Tabla de Análisis

- Construir la tabla LR(0) teniendo en cuenta que un estado puede ahora tener desplazamientos y reducciones.
- No son necesarias las columnas "**acción**" y "**regla**".
- El símbolo **$** es un símbolo válido de lectura adelantada, luego debe formar parte de las entradas.
- Poner en cada entrada de la tabla una etiqueta "**R**" o "**D**"
  1. Si es una reducción con la forma A→XYZ. , incluir la regla gramatical A→XYZ para todas las entradas en el **Siguiente (A).**
  2. En un estado que contiene el elemento S'→S. hay que poner **aceptación** cuando la entrada sea '**$**'.
  3. Si es un desplazamiento, poner el estado al que transita.

Análisis Sintáctico SLR(1)

EJEMPLO

Construcción de la tabla de análisis para el siguiente autómata :



E' → E

E → E + n

E → n

E' →.E

E' → E.

E → .E + n

E → E. + n

E → E +.n

E → E + n.

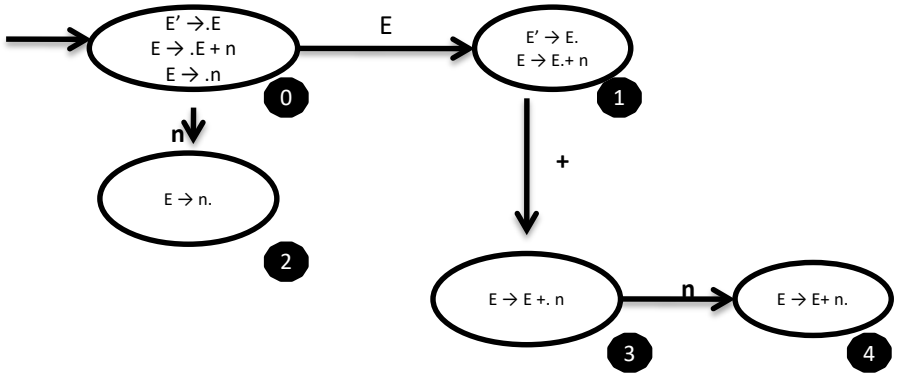
E → .n

E → n.

Análisis Sintáctico SLR(1)

EJEMPLO

Construcción de la tabla de análisis para el siguiente autómata :



Sig(E') = {$}

Sig(E) = {+ , $}

E' → E

E → E + n

E → n

E' →.E
E' → E.
E → .E + n
E → E. + n
E → E +.n
E → E + n.
E → .n
E → n.

Estado	Entrada			Ir_a
	+	n	$	E
0		D : 2		1
1	D : 3		Aceptar	
2	R : E → n		R : E → n	
3		D : 2		
4	R : E → E + n		R : E → E + n	

REGLAS

1. El símbolo **$** es un símbolo válido de lectura adelantada, luego debe formar parte de las entradas.
2. Poner en cada entrada de la tabla una etiqueta "**R**" o "**D**"
   - Si es una reducción con la forma AXYZ. , incluir la regla gramatical AXYZ para todas las entradas en el **Siguiente (A).**
   - En un estado que contiene el elemento S'→S. hay que poner **aceptación** cuando la entrada sea '**$**'.
   - Si es un desplazamiento, poner el estado al que transita.

Análisis Sintáctico SLR(1)

EJEMPLO

Gramática

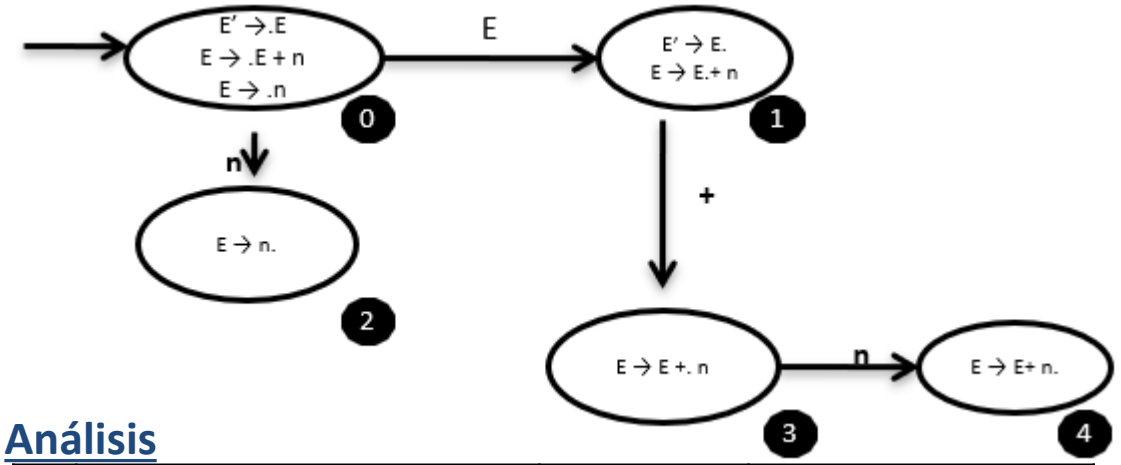
E’ → E

E → E + n

E → n

Entrada

n + n + n



Tabla

Estado	Entrada			Ir_a
	+	n	$	E
0		D : 2		1
1	D : 3		Aceptar	
2	R : E → n		R : E → n	
3		D : 4		
4	R : E → E + n		R : E → E + n	

Análisis

#	Pila de Análisis sintáctico	Entrada	Acción
1	$0	**n + n + n$**	Desplazar 2
2	$0n2	**+ n + n$**	Reducir E → n
3	$0E1	**+ n + n$**	Desplazar 3
4	$0E1+3	**n + n$**	Desplazar 4
5	$0E1+3n4	**+ n$**	Reducir E → E + n
6	$0E1	**+ n$**	Desplazar 3
7	$0E1+3	**n$**	Desplazar 4
8	$0E1+3n4	$	Reducir E → E + n
9	$0E1	$	**Aceptar**

Análisis Sintáctico SLR(1)

Algoritmo SLR(1) para el estado actual *E*

- Si *E* contiene un elemento de la forma **A→α.Xβ** (X es un terminal) y X es el siguiente token de la entrada:
  - Desplazar X a la pila.
  - El siguiente estado es A→αX.β .
- Si *E* contiene un elemento completo (A→α.) y el token siguiente de la entrada está en Siguiente(A):
  - Reducir por la regla A→α .
  - El nuevo estado se calcula eliminando α de la pila y todos sus estados
  - Retroceder al estado donde comenzó la construcción de α .
- Una reducción por la regla S'→S con entrada=$ implica aceptación.
- Si el siguiente token de la entrada no permite aplicar ninguna de las reglas anteriores, error sintáctico.

Análisis Sintáctico SLR(1)

Una gramática es SLR(1) si y sólo si:

1. Para cualquier elemento **A→α.xβ,** en un estado *E* con un terminal *x y* no hay elemento completo B → μ. en *E* con *x* en Siguiente(B).
2. Para cualesquiera dos elementos completos A y B en un estado *E*,

$$\text{Siguiente}(A) \cap \text{Siguiente}(B) = \emptyset$$

Conflictos

- Reducción - Desplazamiento: si se viola la regla 1.
- Reducción - Reducción: si se viola la regla 2.

- No obstante, SLR(1) permite postponer la decisión hasta el último momento esto implica mayor potencia en el análisis.

Análisis Sintáctico SLR(1)

- Los conflictos en SLR(1) pueden eliminarse mediante reglas de eliminación de la ambigüedad:
  - Reducción - Desplazamiento: Preferir el desplazamiento.
  - Reducción - Reducción: Generalmente indican un error en el diseño de la gramática, aunque no siempre.

- Preferir el desplazamiento en un conflicto D-R selecciona automáticamente la regla de anidamiento más cercana en sentencias “if..then..else”.

```
sentencia       → sentencia_if | otro
sentencia_if    → if (expresion) sentencia |
                  if (expresion) sentencia else sentencia
expresión       → true | false
```

Análisis Sintáctico SLR(1)

- Simplificada y aumentada:

```
S'→ S
S → I | otro
I → if (E) S | if (E) S else S
E → true | false
```

- Eliminando la condición obtenemos una gramática igualmente ambigua:

```
S'→ S
S → I | otro
I → if S | if S else S
```

Análisis Sintáctico LALR(1)

"Look Ahead" LR Método más utilizado, especialmente por los generadores automáticos como CUP o Yacc.

- SLR(1): aplica las búsquedas hacia delante después de construir el autómata de elementos LR(0)
  - Utiliza el mismo autómata que LR(0), sólo cambia la construcción de la tabla.
  - El autómata ignora la búsqueda hacia delante.
- La capacidad de los métodos LR(*k) radica en la construcción de un* autómata que integra las búsquedas hacia delante.
- Se utilizan nuevos elementos, denominados LR(1).

Análisis Sintáctico LALR(1)

Cada elemento **LR(1)** incluye un *token de búsqueda* hacia delante: **[A→α.β, a]**

En las transiciones entre estados también se tiene en cuenta la búsqueda hacia delante.

1. Transiciones **no Ɛ** (entre estados) :
   - Idénticas a las del autómata LR(0) .
   - Dado un elemento [A →α.Xμ, a], donde X es cualquier símbolo, existe una transición con X al elemento [A →αX.μ ,a] .
   - Estas transiciones no provocan la aparición de nuevas búsquedas hacia delante, pues 'a' aparece en ambos elementos.

2. Transiciones **Ɛ** (se hace la cerradura dentro del estado) según:
   - Dado un elemento [A →α.Bμ], donde B es un no terminal, se añade al estado elementos [B →.β,b] para cada producción B→β y para cada token en ***Primero(μa)***. Nota: μa ≡ 'μ' concatenado con 'a' .
   - El elemento [A →α.Bμ, a] indica que podríamos querer reconocer una B pero sólo si va seguida de una cadena derivable de "μa".

El símbolo inicial es [S' →.S,$] .

Análisis Sintáctico LALR(1)

Algoritmo de Construcción del Autómata

1. Se construye el núcleo del estado inicial con **S'→.S,$**
2. Para cada núcleo nuevo:
   a) Se hace la cerradura del núcleo según lo visto para elementos LR(1).
   b) Se repite la cerradura para los nuevos elementos añadidos hasta que no se añadan nuevos elementos al estado y se considera que el estado está completo.
3. Para cada estado completo n:
   a) Se crean nuevos núcleos de estados a partir de n mediante las transiciones no Ɛ . (Las transiciones Ɛ ya no se realizan porque son representadas por la cerradura).

      Si el núcleo creado ya existe en otro estado no se añade y la transición se dirige al estado existente.

   b) Se vuelve a 2
   c) Se realiza 3 hasta que no se puedan añadir nuevos estados al autómata.

Análisis Sintáctico LALR(1)

EJEMPLO

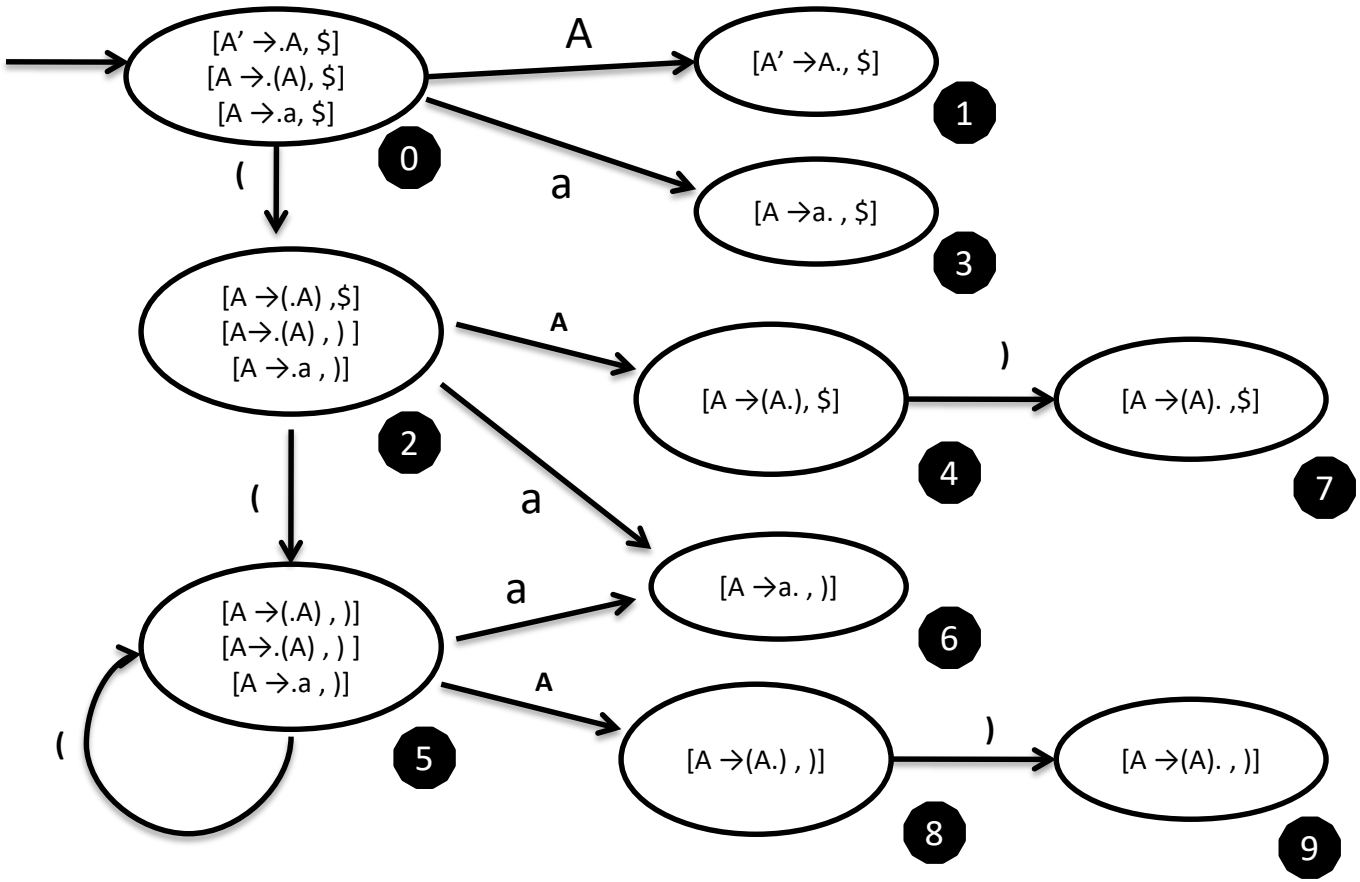
Gramática: A →(A)|a

- Estado 0:
  - [A'→ .A, $]
  - [A → .(A), $]
  - [A → .a, $]
- Estado 1: (Estado 0 con transición "A")
  - [A' → A., $]
- Estado 2: (Estado 0 con transición "(")
  - [A → (.A), $]
  - [A → .(A),)]
  - [A → .a,)]
- Estado 3: (Estado 0 con transición "a")
  - [A → a., $]
- Estado 4: (Estado 2 con transición "A")
  - [A (A.), $]
- Estado 5: (Estado 2 con transición "(")
  - [A → (.A),)]
  - [A → .(A),)]
  - [A → .a,)]
- Estado 6: (Estado 2 con transición "a")
  - [A → a.,)]
- Estado 7: (Estado 4 con transición ")")
  - [A → (A)., $]
- Estado 8: (Estado 5 con transición "A")
  - [A → (A.),)]
- Estado 9: (Estado 8 con transición ")")
  - [A → (A).,)]

Análisis Sintáctico LALR(1)

EJEMPLO

El autómata finito determinista es:



Análisis Sintáctico LALR(1)

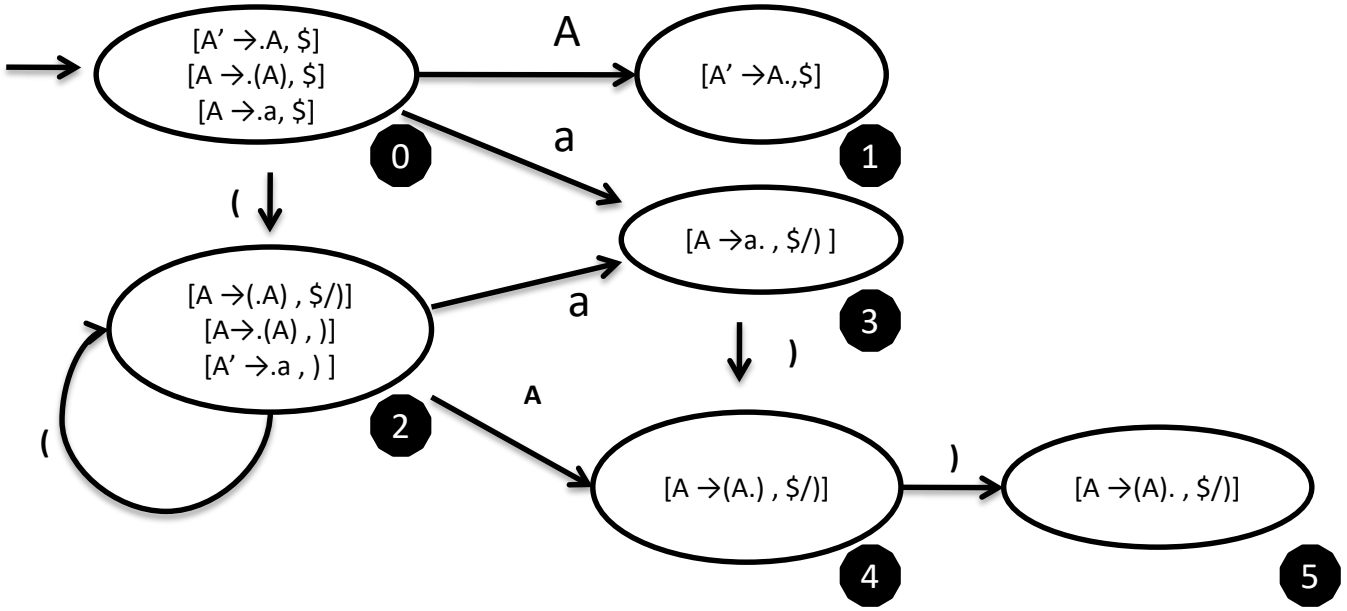
El análisis LALR(1) se basa en que el tamaño del autómata LR(1) puede **simplificarse** unificando todos los estados con igual núcleo para los que existen varios elementos de búsqueda hacia delante.

- Primer principio del análisis LALR(1) :
  - El núcleo de un estado del autómata de elementos LR(1) es un estado del autómata de elementos LR(0)
- Segundo principio del análisis LALR(1) :
  - Si hay dos estados S_1 y S_2 del autómata LR(1) con el mismo núcleo y,
  - Si hay una transición con un símbolo X desde S_1 hasta T_1,
  - Entonces también hay una transición con un símbolo X desde S_2 hasta T_2 y T_1 y T_2 tienen el mismo núcleo.
  - Por tanto se unifican S_1 y S_2 y T_1 y T_2.

Análisis Sintáctico LALR(1)

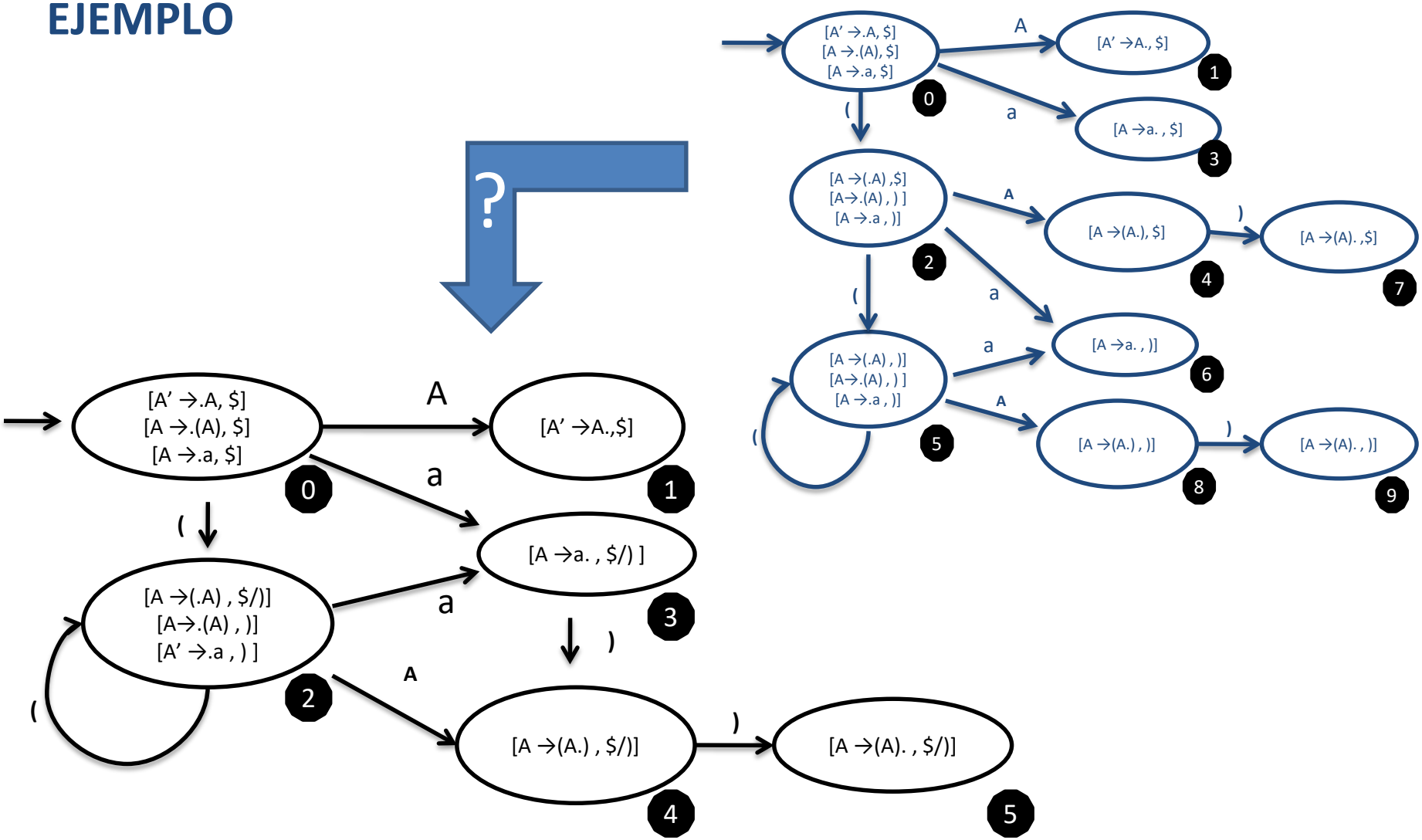
EJEMPLO

Con la aplicación del mecanismo de simplificación obtenemos :



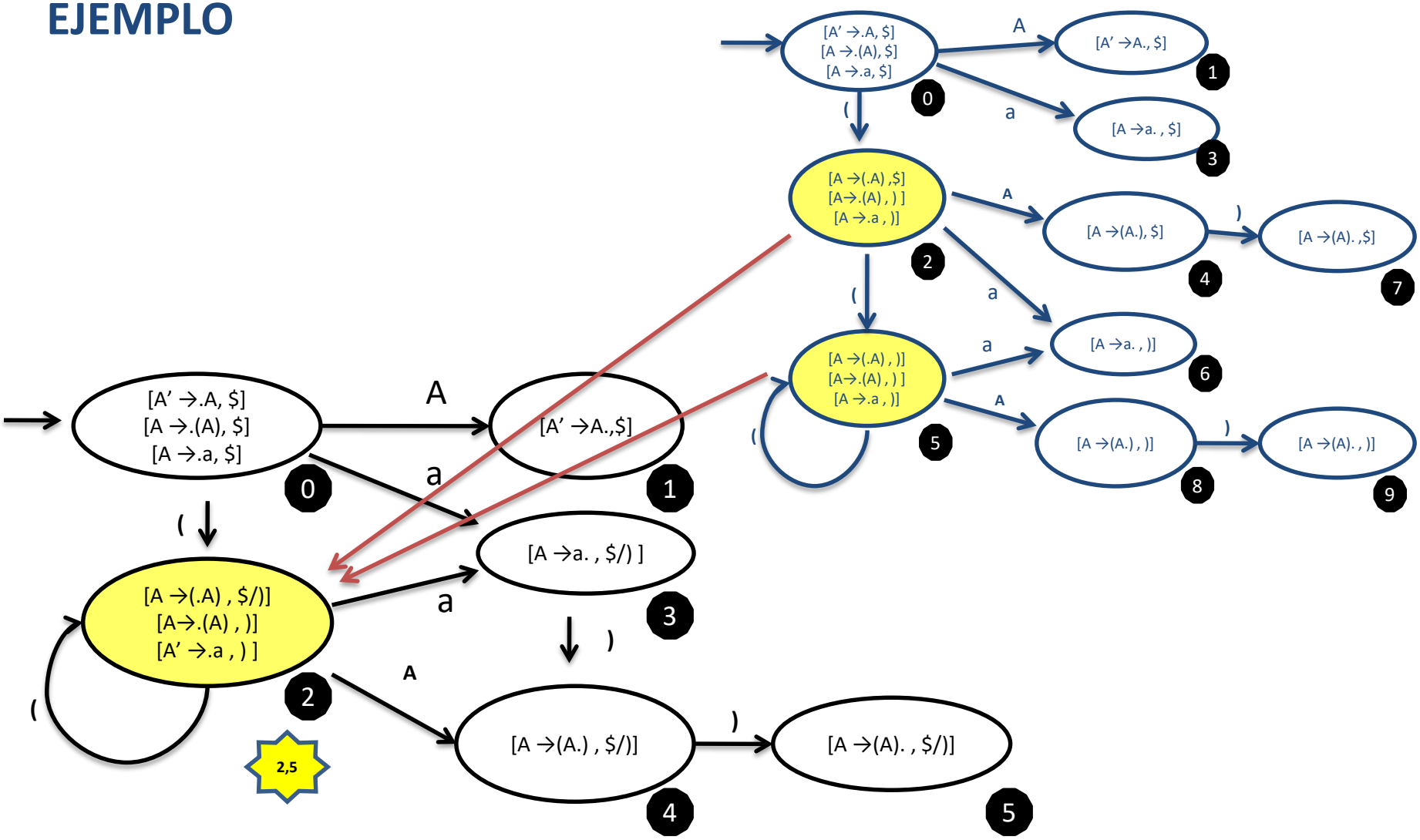
Análisis Sintáctico LALR(1)

EJEMPLO



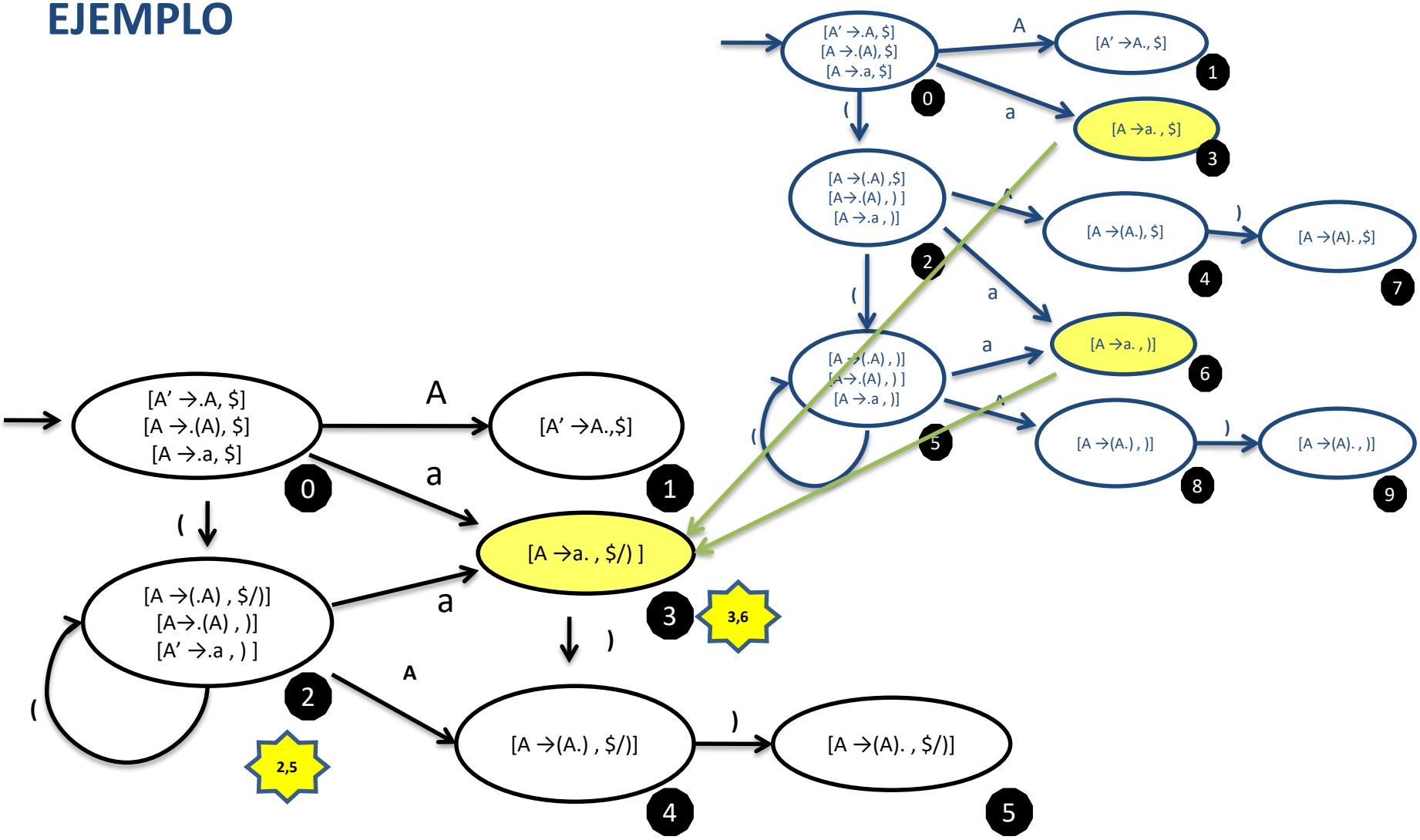
Análisis Sintáctico LALR(1)

EJEMPLO



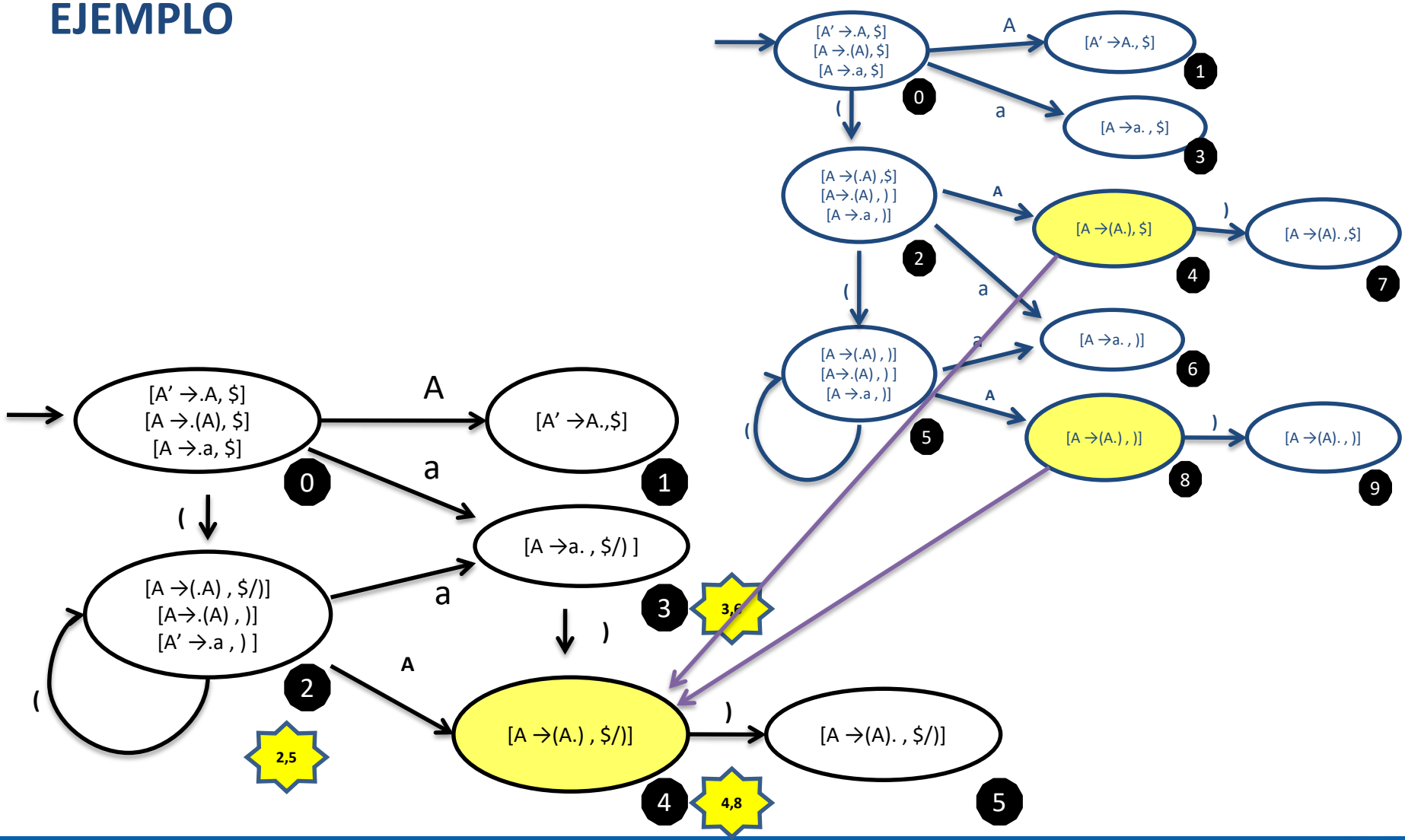
Análisis Sintáctico LALR(1)

EJEMPLO



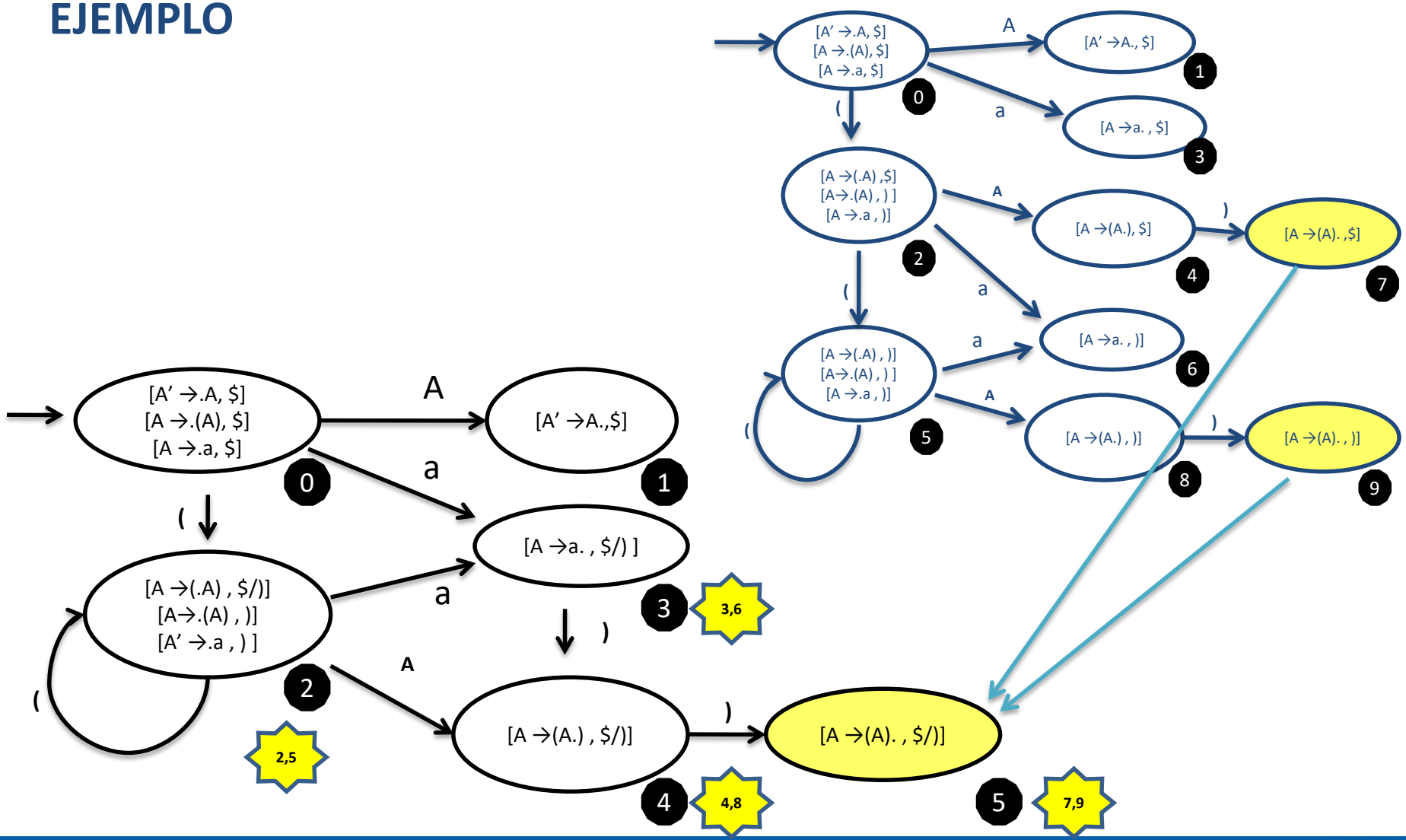
Análisis Sintáctico LALR(1)

EJEMPLO



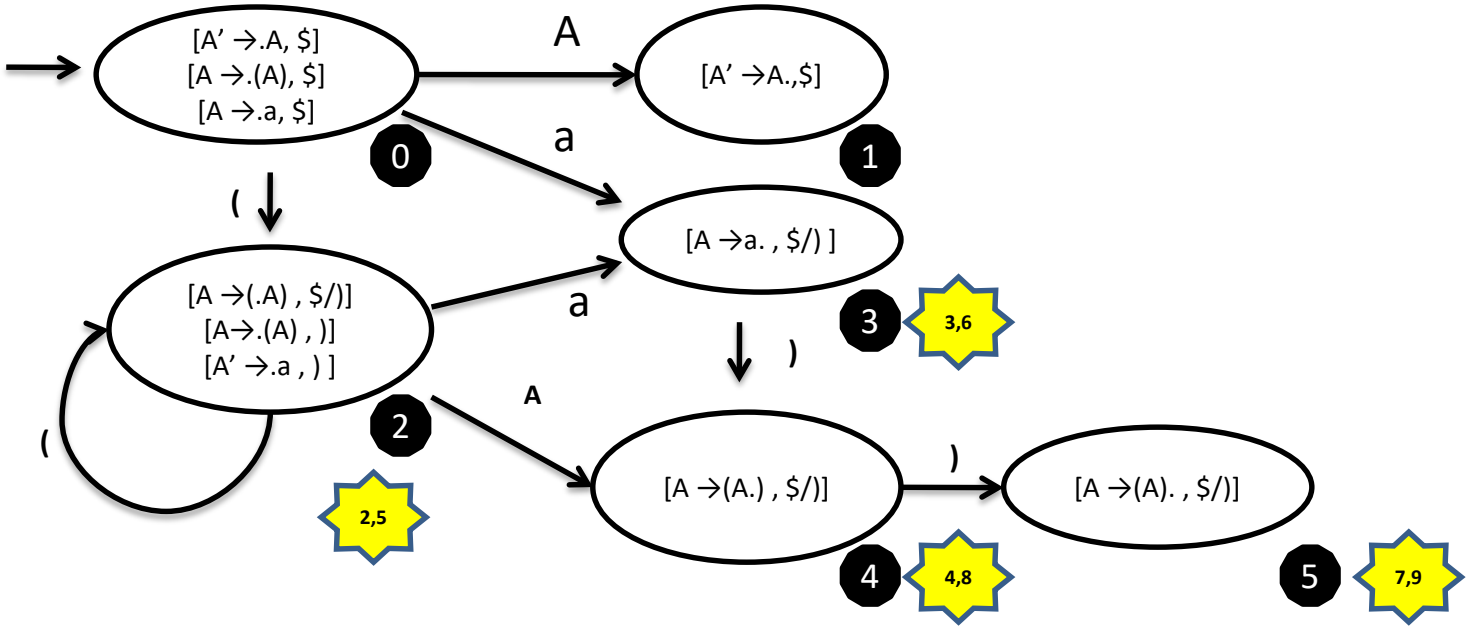
Análisis Sintáctico LALR(1)

EJEMPLO



Análisis Sintáctico LALR(1)

EJEMPLO



Análisis Sintáctico LALR(1)

El algoritmo de análisis, es similar al SLR(1), utiliza los tokens de búsqueda hacia delante, en lugar de los conjuntos *Siguiente.*

Para el estado actual *E*:

- Si *E* contiene un elemento de la forma [A→α.Xβ,a] y X es un terminal y X es el siguiente token de la entrada:
  - Desplazar X a la pila.
  - El siguiente estado es [A→αX.β,a].
- Si E contiene un elemento completo [A→α.,a] y el token siguiente de la entrada es "a":
  - Reducir por la regla A→α .
  - El nuevo estado se calcula eliminando de la pila y todos sus estados.
  - Retroceder al estado donde comenzó la construcción de α .
- Una reducción por la regla S'→E con entrada $ implica aceptación.
- Si el siguiente token de la entrada no permite aplicar ninguna de las reglas anteriores, error sintáctico.

Análisis Sintáctico LALR(1)

Conflictos

- **Desplazamiento - Reducción** (en un estado): Cuando el símbolo de *look-ahead* del elemento completo es también símbolo de transición.

- **Reducción - Reducción** (en un estado): Cuando dos elementos completos tienen el mismo símbolo de *look-ahead*.

Resumen de Analizadores Sintáctico

- Una gramática es **LL(1)** si y sólo si los conjuntos de predicción de las alternativas de producción sobre el mismo símbolo son disjuntos.
  - Si tiene recursividad por la izquierda no es LL(1). No obstante, que no sea LL(1) no significa necesariamente que sea recursiva por la izquierda.
- Gramática **LR(0)** - Todos los estados del autómata son bien de reducción o bien de desplazamiento.
- Una gramática es **SLR(1)** si es posible construir una tabla de análisis SLR sin entradas múltiples.
- Gramática **LR(1)** – No es usada en la práctica debido al gran número de estados que produce, sino que se utiliza una gramática simplificada, LALR(1).
- Una gramática es **LALR(1)** si no incluye conflictos de análisis. Es la más importante y la que implementan la mayoría de los generadores.
- Una gramática ambigua no es **LL(1)** ni **SLR** ni **LR(1)**.

Resumen de Analizadores Sintáctico

Ventajas e inconvenientes de los analizadores LR (0)

- Los analizadores sintácticos ascendentes del tipo LR(0) son los más sencillos de construir.
- El conjunto de gramáticas que reconocen es relativamente pequeño.
- Estos autómatas no utilizan ninguna estrategia predictiva en función de los terminales a la cabeza para dirigir los procesos de reducción.

Ventajas	Inconvenientes
Fácil de entender y construir.	Conjunto reducido de gramáticas.
La estrategia de reducción es sencilla.	No aplica estrategias predictivas.
Cada estado de reducción es por 1 regla.	No pueden existen 2 reducciones por estado.
Los errores se delegan a desplazamiento.	Mensajes de error menos localizados.

Resumen de Analizadores Sintáctico

Ventajas e inconvenientes de los analizadores SLR

- Los analizadores sintácticos ascendentes del tipo SLR suponen una mejora con respecto a los del tipo LR(0) que refinan las condiciones de entrada que deben darse para poder hacer una reducción por una determinada regla.

Ventajas	Inconvenientes
Fácil de entender y construir.	Conjunto mayor pero reducido de gramáticas.
Predice reducción mediante Siguientes.	Estrategia de reducción más compleja.
Varias reducciones por el mismo estado.	No existen estados de (única) reducción.
Utiliza el mismo autómata LR (0).	El autómata LR (0) no contempla predicción.

Resumen de Analizadores Sintáctico

Ventajas e inconvenientes de los analizadores LALR(1)

- Los analizadores sintácticos ascendentes del tipo LALR(1) son los más generales que hemos presentado y admiten un gran número de gramáticas.
- Proceden de la simplificación del analizador LR(1).

Ventajas	Inconvenientes
Gran conjunto de gramáticas posibles.	Mas difícil de entender y construir.
Predicción con terminales de búsquedas.	Estrategia de reducción más compleja.
Varias reducciones por el mismo estado.	No existen estados de (única) reducción.
La predicción está muy ajustada.	Utiliza un autómata más complejo.

Análisis Sintáctico - Recuperación de errores

Errores en el Análisis Sintáctico

- El analizador sintáctico detecta un error de sintaxis cuando el analizador léxico proporciona el siguiente símbolo y éste es incompatible con el estado actual del analizador sintáctico. Los errores sintácticos típicos son:
  - *Paréntesis o corchetes omitidos*, por ejemplo, x : = y * (1 + z;
  - *Operadores u operando omitidos,* por ejemplo, x : = y (1 + z);
  - *Delimitadores omitidos,* por ejemplo, x : = y + 1 IF a THEN y : = z.
- No hay estrategias de recuperación de errores cuya validez sea general, y la mayoría de las estrategias conocidas son heurísticas, ya que se basan en suposiciones acerca de cómo pueden ocurrir los errores y lo que probablemente quiso decir el programador con una determinada construcción.

Análisis Sintáctico - Recuperación de errores

- El Análisis Sintáctico es la fase de compilación que contempla un mayor número de errores
  - Los programadores suelen cometer un mayor número de errores sintácticos que de otro tipo.
- Objetivos principales de todo manejador de errores:
  - Detectar los errores lo antes posible (mensajes de error más significativos)
  - Informar de los errores con claridad y exactitud.
  - Recuperarse del error sintáctico.
  - No retrasar el procesamiento de programas correctos.

- Sin embargo, hay algunas estrategias que gozan de amplia aceptación y las veremos seguidamente.

Análisis Sintáctico - Recuperación de errores

Recuperación en MODO PÁNICO

- Método más sencillo de recuperación.
- Consiste en **desechar componentes léxicos de la entrada** hasta encontrar uno que permita continuar con el proceso de compilación.
  - Generalmente estructuras de cierre: *punto y coma, end o delimitadores.*
- Intenta aislar la frase que contiene el error, determinando qué cadena derivable de A tiene el error y asumiendo que se ha reconocido un A completo. Se descartan símbolos de la entrada, hasta encontrar un símbolo a que pertenece a SIG(A) y el parser realiza Ir_a[s,A]
- Es un método muy sencillo, y no puede producir bucles infinitos de recuperación, pero desecha muchos símbolos de la entrada.

Análisis Sintáctico - Recuperación de errores

Recuperación en MODO PÁNICO

- Al descubrir un error
  1. POP de la pila hasta que lleguemos a un estado X de donde se pueda hacer un **goto** para alguno de los no-terminales de pánico.
     - Se termina el análisis si no se encuentra ninguno.
  2. Descartamos tokens del buffer de entrada hasta que encontremos un token sincronizador.
     - Un token que pertenece a **Sig.(A)** del no-terminal de pánico A que encontramos en el paso anterior.
  3. PUSH de A y del estado **goto(S,A**) en la pila y seguimos analizando.

Análisis Sintáctico - Recuperación de errores

Recuperación A NIVEL DE FRASE

- Este método realiza una corrección local de la entrada restante.
- Sustituye parte de la entrada inválida por otra que permita continuar:

$$= \rightarrow :=$$

- Este nivel de recuperación depende de las sustituciones implementadas por el diseñador.
- Puede producir ciclos infinitos de recuperación

Análisis Sintáctico - Recuperación de errores

Recuperación por PRODUCCIONES DE ERROR

- El método añade a la gramática reglas o producciones erróneas, permitiendo trabajar en situaciones no locales.
- Producción de error:

A → error

- Marca un contexto en el que los tokens de error pueden eliminarse hasta que se encuentre uno correcto.
- Se amplia la gramática con reglas de error que aparezcan con frecuencia y en cada regla se aplica una recuperación especifica.
- Ver documento anexo "Analizadores Sintacticos (Recuperación de error) ".

Análisis Sintáctico - Recuperación de errores

Recuperación por CORRECCIÓN GLOBAL

- El método que eligen una secuencia mínima de cambios para obtener una corrección global de menor coste.
- Ejemplo:

x=a(p+q(-b(r-s); → a(p+q)-b(r-s);

ifa=b then sum=0; → if a =b then sum=0;

- Esta técnicas es costosas en tiempo y espacio: métricas de distancias, búsqueda, optimización.

Análisis Sintáctico - Recuperación de errores

Conclusiones de Manejo de Errores

- Aspecto complejo del diseño del compilador .
  - Análisis cuidadoso del lenguaje y posibles errores.
  - No hay una estrategia de aceptación universal.
  - Abundan técnicas heurísticas y ad hoc.
- Principio general de recuperación
  - Minimizar tokens eliminados/modificados.
  - Dejar el analizador listo para continuar procesando.
- Objetivos
  - Informar con claridad, exactitud y extensión.
  - Evitar errores en cascada y procesos infinitos.
- Analizadores LL y LR poseen gran capacidad para estrategias en modo Pánico y Frase en función del contexto

Fuentes

- Para la elaboración de estas transparencias se han utilizado:
  - Transparencias de cursos previos (elaboradas por los profesores Dr. D. Salvador Sánchez, Dr. D. José Luis Cuadrado).
  - Libros de referencia.