Procesadores de lenguaje

Tema 3 – Análisis sintáctico (Parte II)

Salvador Sánchez, Daniel Rodríguez
Departamento de Ciencias de la Computación
Universidad de Alcalá

Resumen

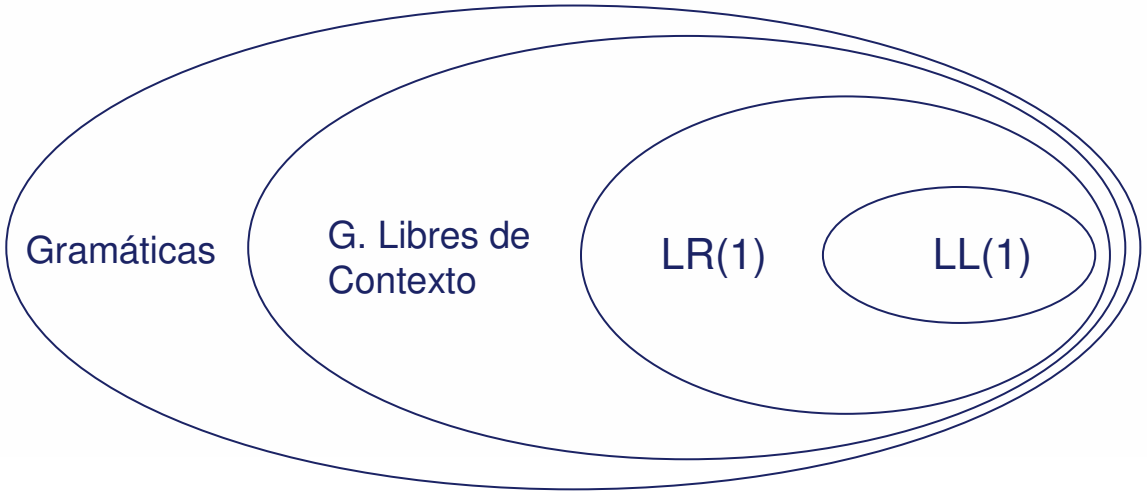
- Análisis sintáctico ascendente
- Analizadores ascendentes LR: LR(0)
- Analizador SLR(1)
- Analizador LALR(1)
- Recuperación de errores
- Generadores automáticos de analizadores sintácticos

Recordatorio: tipos de analizador

- 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, tomando los símbolos de uno en uno.

Analizadores sintácticos ascendentes

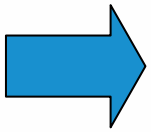
- Comprueban si una cadena de símbolos pertenece a un lenguaje aplicando reducciones sobre la entrada hasta alcanzar el axioma de la gramática.
- En general son más potentes que los métodos descendentes.
  - Pueden manejar gramáticas recursivas a izquierdas.
- Los algoritmos de implementación son más 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.



Analizador ascendente: funcionamiento

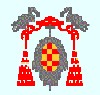
- Busca aquellas subcadenas de la entrada que sean la parte derecha de una producción y las sustituye por la parte izquierda correspondiente.

S → aAcBe
A→ Ab|b
B→ d



abbcde
aAbcde
aAcde
aAcBe
S
(ÉXITO)

- Problema: varias subcadenas que coinciden en una parte derecha.
  - No es lo mismo sustituir una u otra: una podría llevar al éxito y la otra no



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.

Analizadores ascendentes predictivos

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

Analizadores ascendentes predictivos

- 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.

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

Ejemplo de uso de la pila

S → A

A → Aab | b

Pila	Entrada	Acción
$	bab$	desplazar
$b	ab$	reducir A→b
$A	ab$	desplazar
$Aa	b$	desplazar
$Aab	$	reducir A→Aab
$A	$	reducir S→A
$S	$	ÉXITO

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→Aab"
- Conclusiones:
  - Es necesaria una tabla de análisis que permita hacer predicciones

Análisis sintáctico LR(0)

- 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.

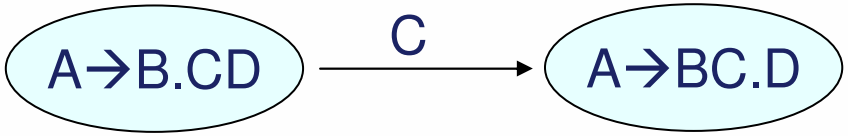
A → xy (regla) ➔
- A→.xy (elemento LR(0) inicial)
- A→x.y (elemento)
- A→xy. (elemento LR(0) final)

- 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:
  - 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, 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)

- Para una gramática como:

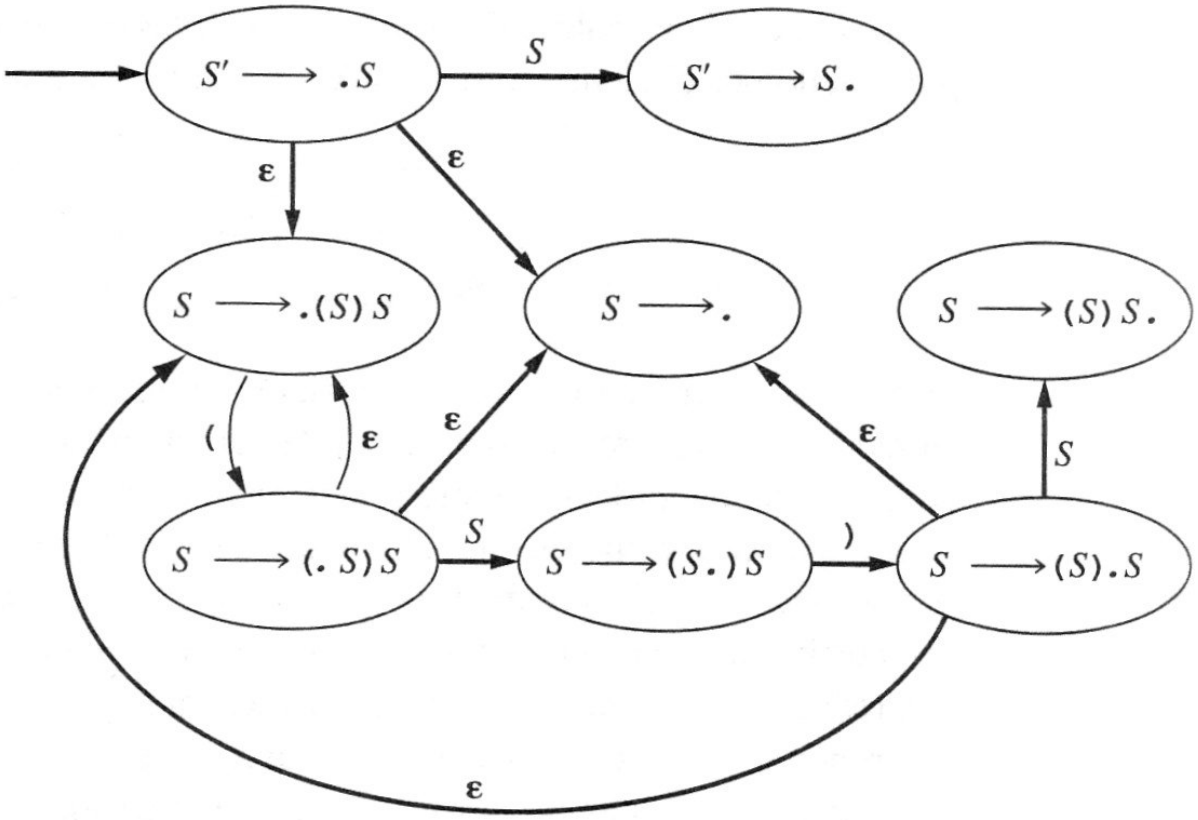
$$S \rightarrow (S)S \mid \varepsilon$$

- 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)

- El autómata finito no determinista es:



Análisis sintáctico LR(0)

- Para una gramática como:

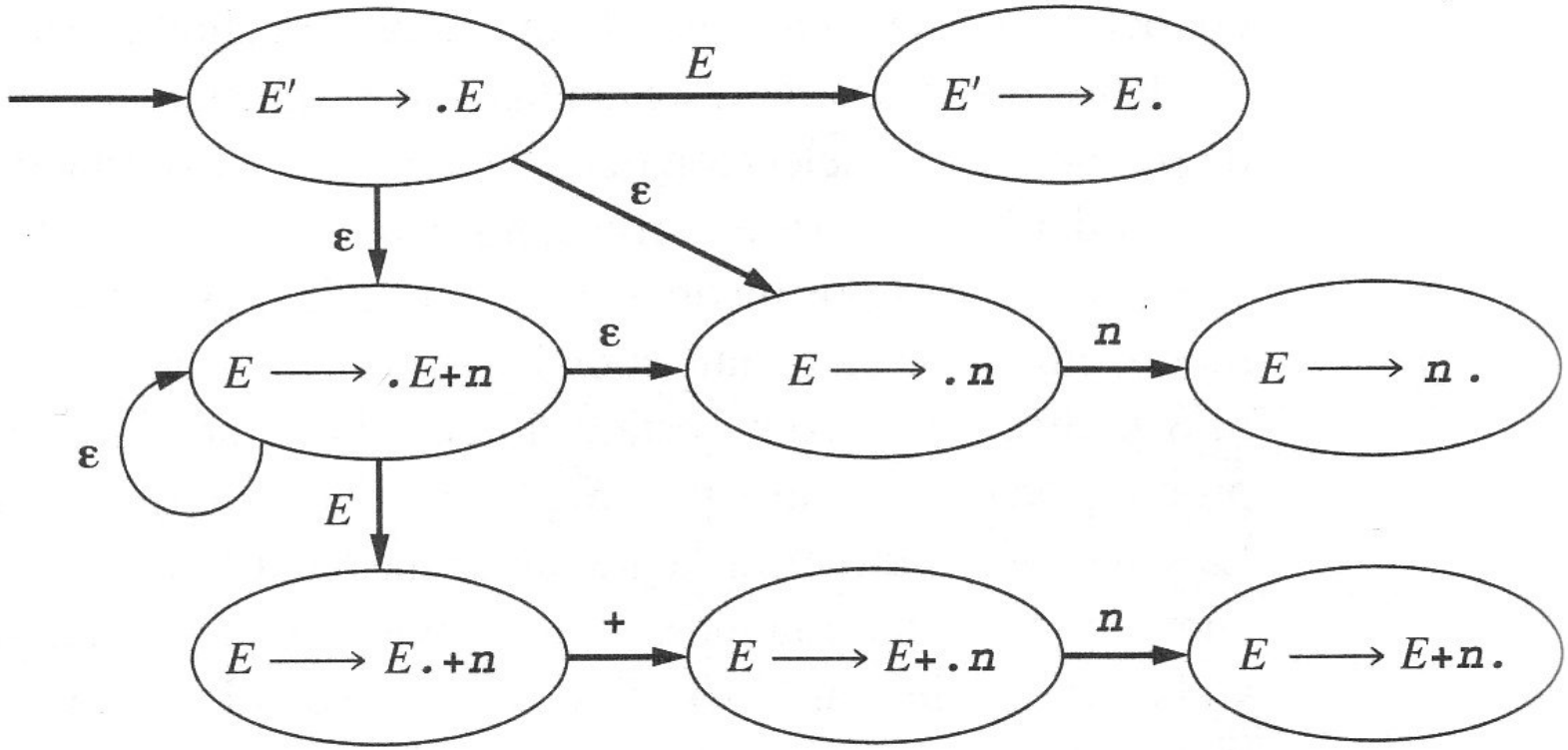
$$E \rightarrow E+n \mid n$$

- Tenemos los siguientes elementos:

$$E' \rightarrow .E$$
$$E' \rightarrow E.$$
$$E \rightarrow .E+n$$
$$E \rightarrow E.+n$$
$$E \rightarrow E+.n$$
$$E \rightarrow E+n.$$
$$E \rightarrow .n$$
$$E \rightarrow n.$$

Análisis sintáctico LR(0)

- El autómata finito no determinista es:



Análisis sintáctico LR(0)

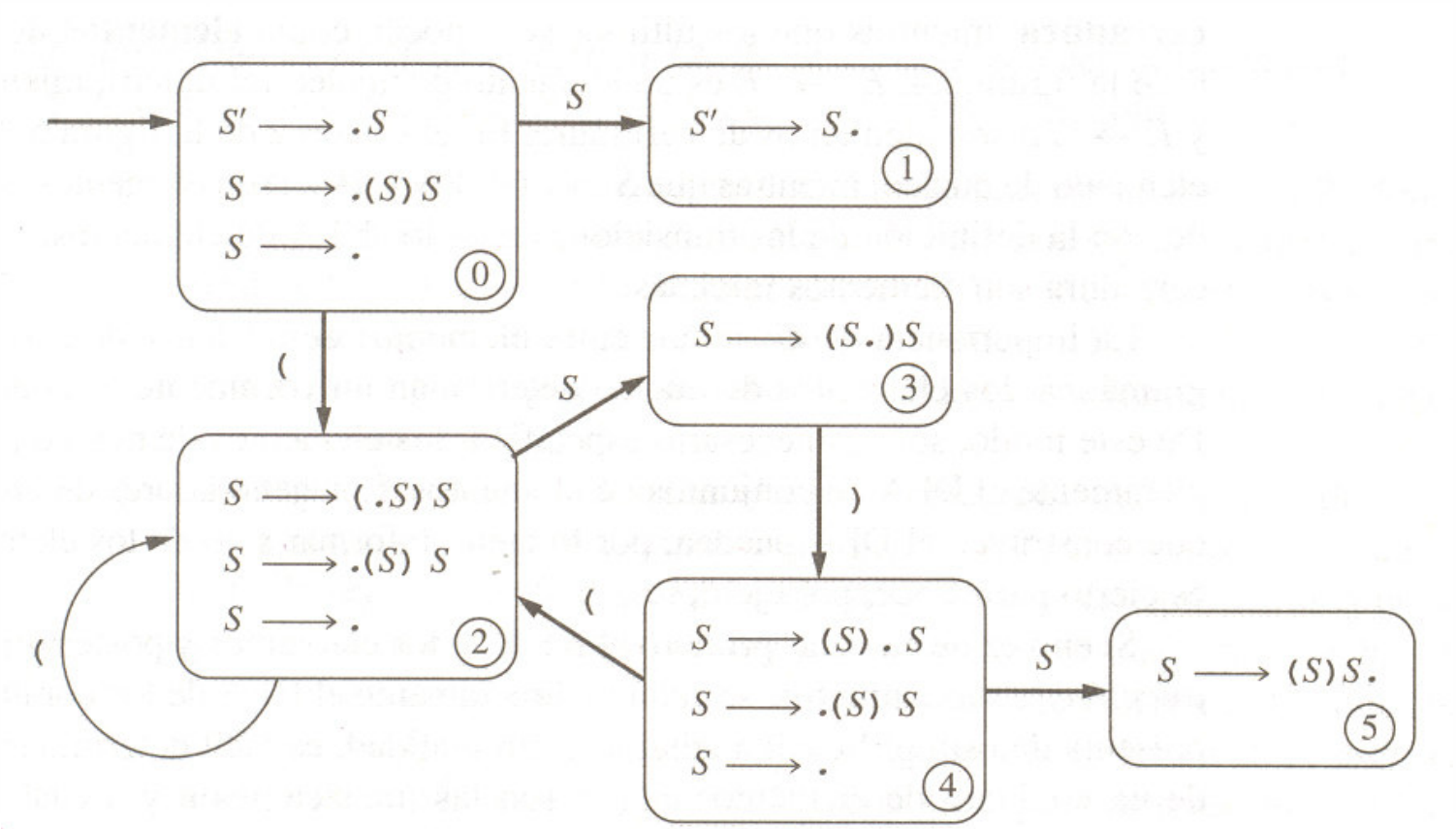
- Es posible construir un autómata finito determinista a partir del autómata no determinista mediante la cerradura de cada estado nuclear.
- Elementos de núcleo: los que se agregan a un estado tras una transición que no es vacía (transición no ε).
- Elementos de cerradura: los que se agregan a un estado tras una transición vacía (transición ε).
- 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).
        a. 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 realiza 3 hasta que no se puedan añadir nuevos estados al autómata.

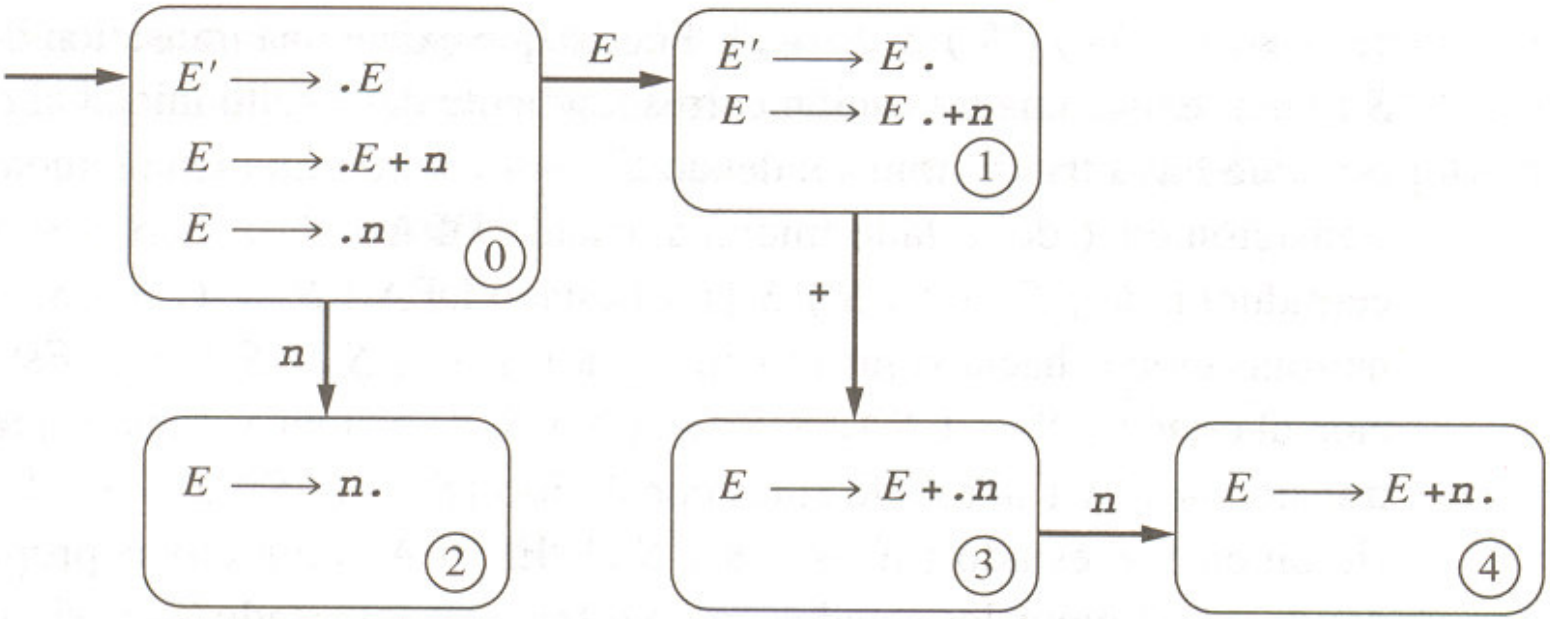
Análisis sintáctico LR(0)

- En el primer ejemplo anterior:



Análisis sintáctico LR(0)

- En el segundo ejemplo anterior:

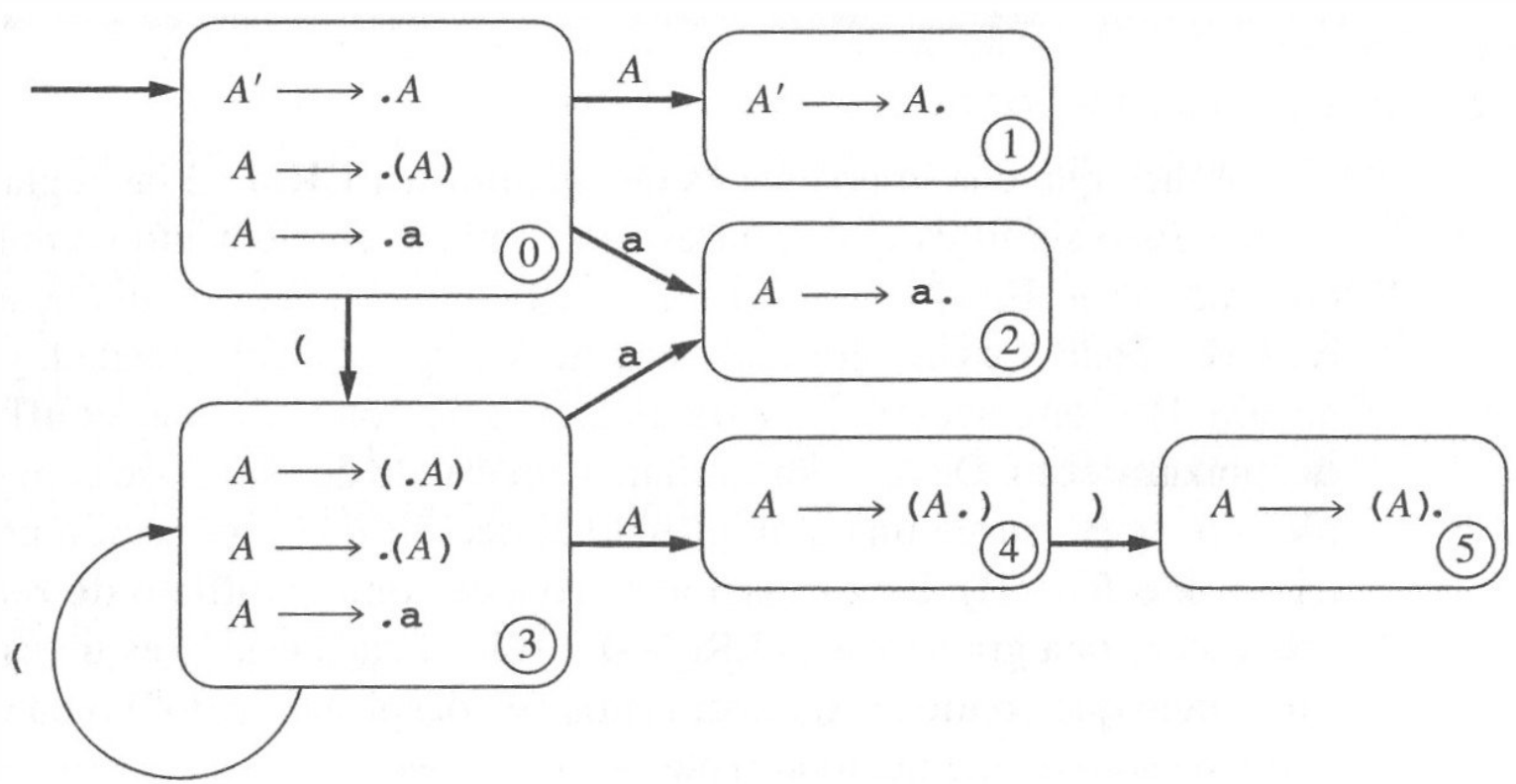


Análisis sintáctico LR(0)

- Se puede generar la tabla de análisis sintáctico a partir del autómata finito determinista.
- 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” (una por terminal) → 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.

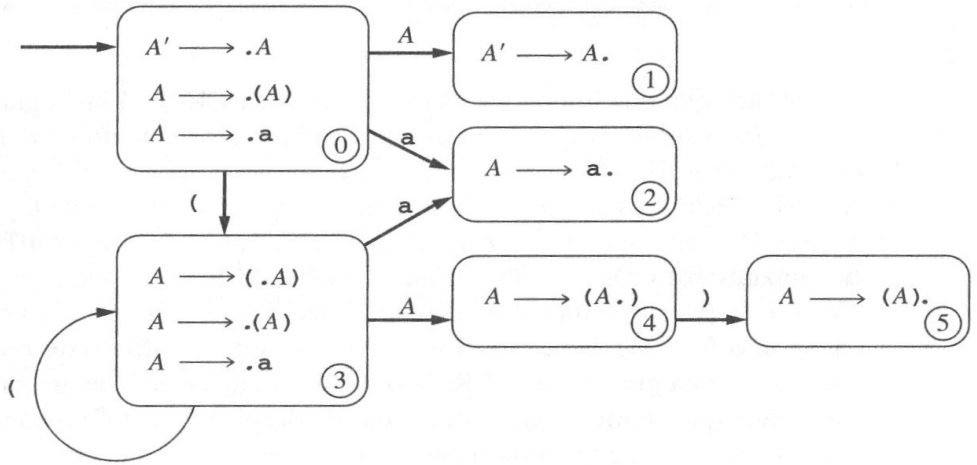
Análisis sintáctico LR(0)

A'→ A
A→ (A) | a



Análisis sintáctico LR(0)

A'→ A
A→ (A) | a



Estado	Acción	Regla	Entrada			Goto
			(	a	)	A
0	desplazamiento		3	2		1
1	reducción	$A' \rightarrow A$				
2	reducción	$A \rightarrow$ a				
3	desplazamiento		3	2		4
4	desplazamiento				5	
5	reducción	$A \rightarrow (A)$				

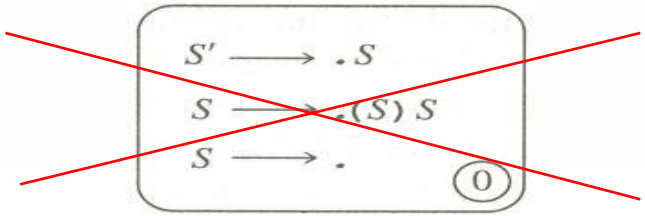
Análisis sintáctico LR(0)

- Ejemplo análisis:

	Pila de análisis sintáctico	Entrada	Acción
1	\$ 0	((a)) \$	desplazamiento
2	\$ 0 (3	(a)) \$	desplazamiento
3	\$ 0 (3 (3	a)) \$	desplazamiento
4	\$ 0 (3 (3 a 2	)) \$	reducción $A \rightarrow$ a
5	\$ 0 (3 (3 A 4	)) \$	desplazamiento
6	\$ 0 (3 (3 A 4) 5	) \$	reducción $A \rightarrow$ (A)
7	\$ 0 (3 A 4	) \$	desplazamiento
8	\$ 0 (3 A 4) 5	\$	reducción $A \rightarrow$ (A)
9	\$ 0 A 1	\$	aceptar

Conflictos en LR(0)

- Si un estado contiene un elemento completo (A→α.) :
  - Si además tiene un elemento "no completo" se produce un conflicto de reducción por 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).
  - Es decir, en LR(0) no puede haber estados mixtos.



Análisis sintáctico SLR(1)

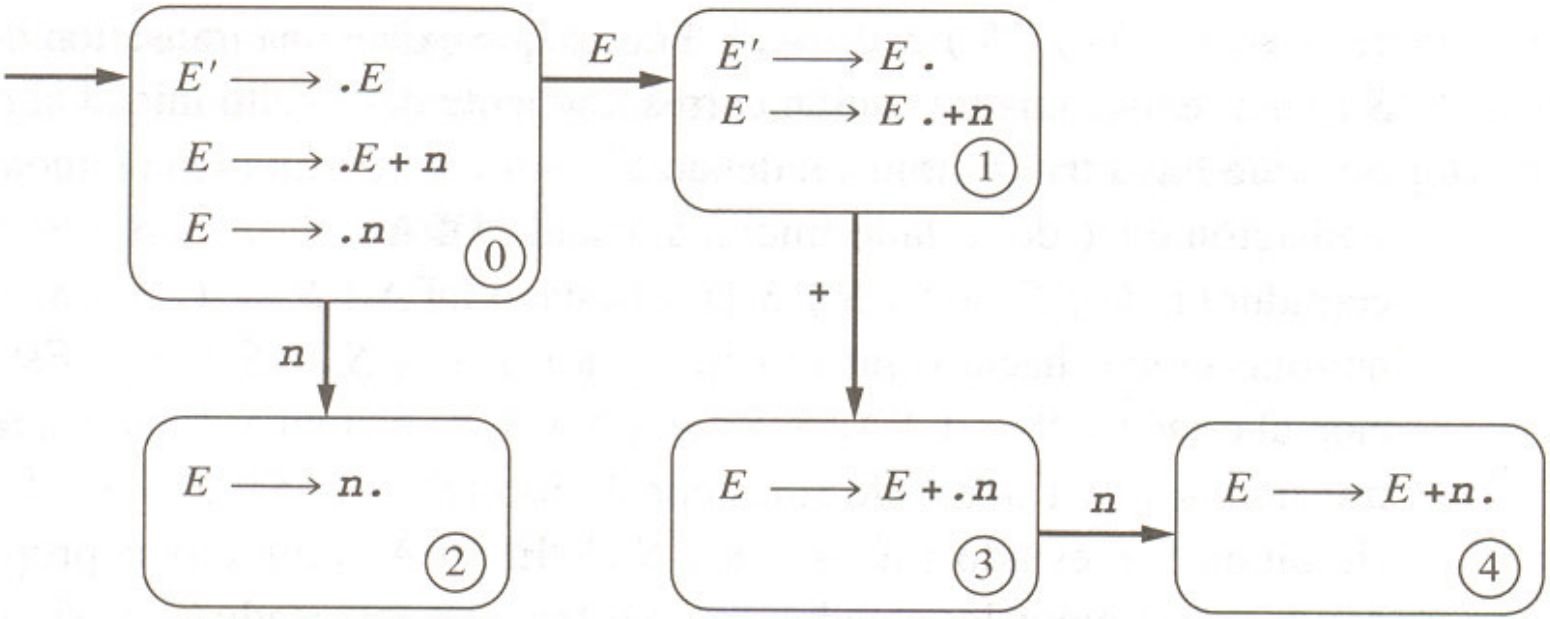
- Análisis sintáctico LR(1) simple o SLR(1).
- Capaz de analizar prácticamente todas las construcciones que se presentan en un lenguaje
  - No obstante, hay situaciones en las que no sirve y por tanto no es analizable.
- Un estado puede tener tanto reducciones como desplazamientos.
- Utiliza el autómata 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”
    - Si es una reducción con la forma A→XYZ. , incluir la regla gramatical A→XYZ 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)

- Construcción de la tabla de análisis para el siguiente autómata:



Análisis sintáctico SLR(1)

- Siguiente(E') = { $ }
- Siguiente(E) = { + , $ }

				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 de una entrada SLR(1)

	Pila de análisis sintáctico	Entrada	Acción
1	\$ 0	n + n + n \$	desplazamiento 2
2	\$ 0 n 2	+ n + n \$	reducción $E \rightarrow n$
3	\$ 0 E 1	+ n + n \$	desplazamiento 3
4	\$ 0 E 1 + 3	n + n \$	desplazamiento 4
5	\$ 0 E 1 + 3 n 4	+ n \$	reducción $E \rightarrow E + n$
6	\$ 0 E 1	+ n \$	desplazamiento 3
7	\$ 0 E 1 + 3	n \$	desplazamiento 4
8	\$ 0 E 1 + 3 n 4	\$	reducción $E \rightarrow E + n$
9	\$ 0 E 1	\$	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 $A \rightarrow \alpha.X\beta$ en un estado E con un terminal X no hay elemento completo $B \rightarrow \varphi.$ en E con X en Siguiente(B)

  2. Para cualesquiera dos elementos completos A y B en un estado E,

     $$Siguiente(A) \cap Siguiente(B) = \emptyset$$

- Conflictos:
  - Reducción por 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: mayor potencia.

Análisis sintáctico SLR(1)

- Los conflictos en SLR(1) pueden eliminarse mediante reglas de eliminación de la ambigüedad:
  - Reducción por 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)

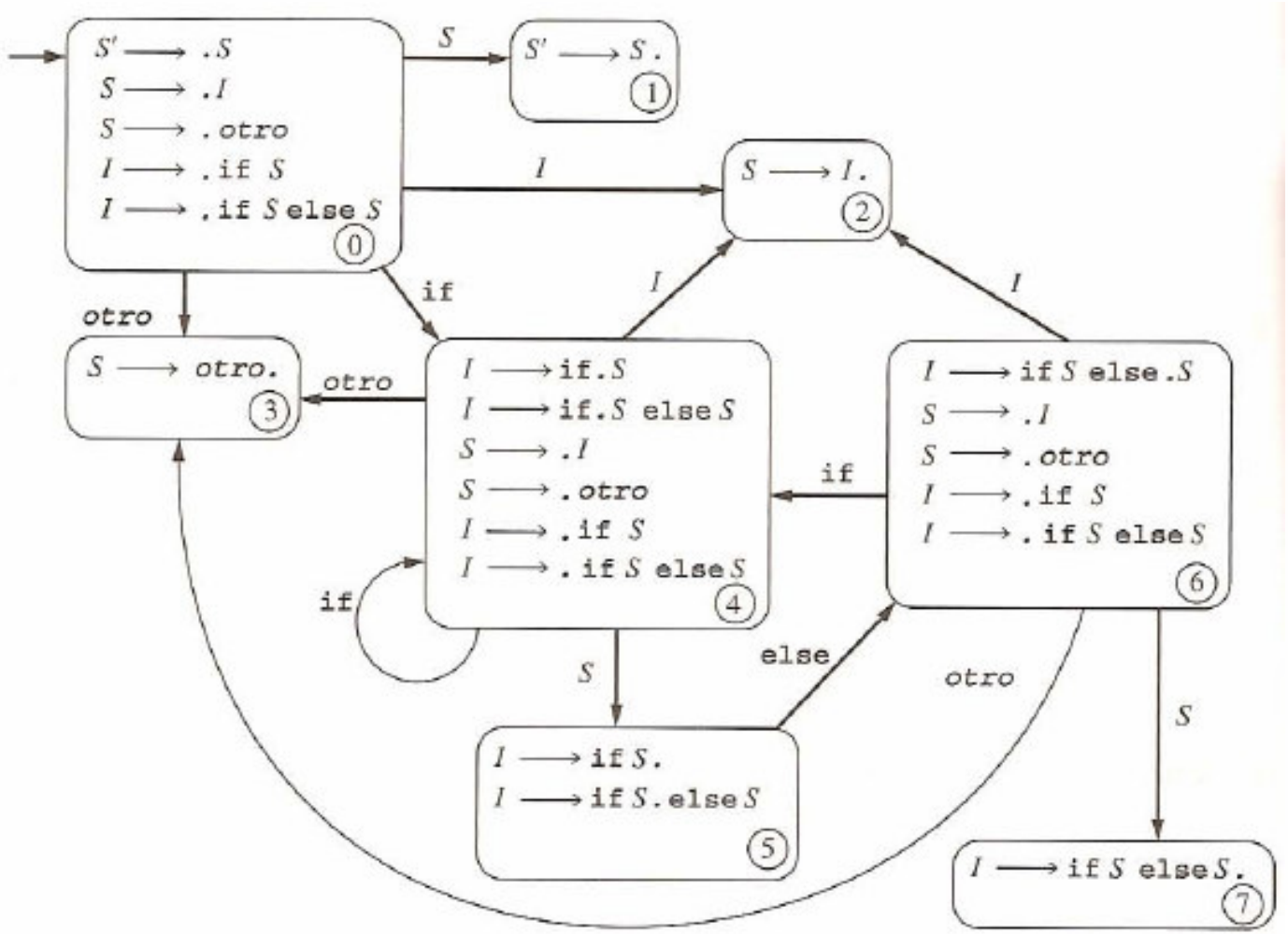
- Simplificada y aumentada:

```
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 SLR(1)



Análisis sintáctico SLR(1)

Estado	Entrada				Ir a	
	`if`	`else`	`otro`	`$`	*S*	*I*
0	s4		s3		1	2
1				aceptar		
2		r1		r1		
3		r2		r2		
4	s4		s3		5	2
5		s6		r3		
6	s4		s3		7	2
7		r4		r4		

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)
  - 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:

$$[A \rightarrow \alpha.\beta, a]$$

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

Análisis sintáctico LALR(1)

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

- Transiciones ε (se hace la cerradura) según (dentro del estado):

  - Dado un elemento [A→α.Bφ, a], 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(φa). Nota: φa ≡ 'φ' concatenado con 'a'

  - El elemento [A→α.Bφ, a] indica que podríamos querer reconocer una B pero sólo si va seguida de una cadena derivable de "φa".

- El símbolo inicial es [S'→.S,$]

Análisis sintáctico LALR(1)

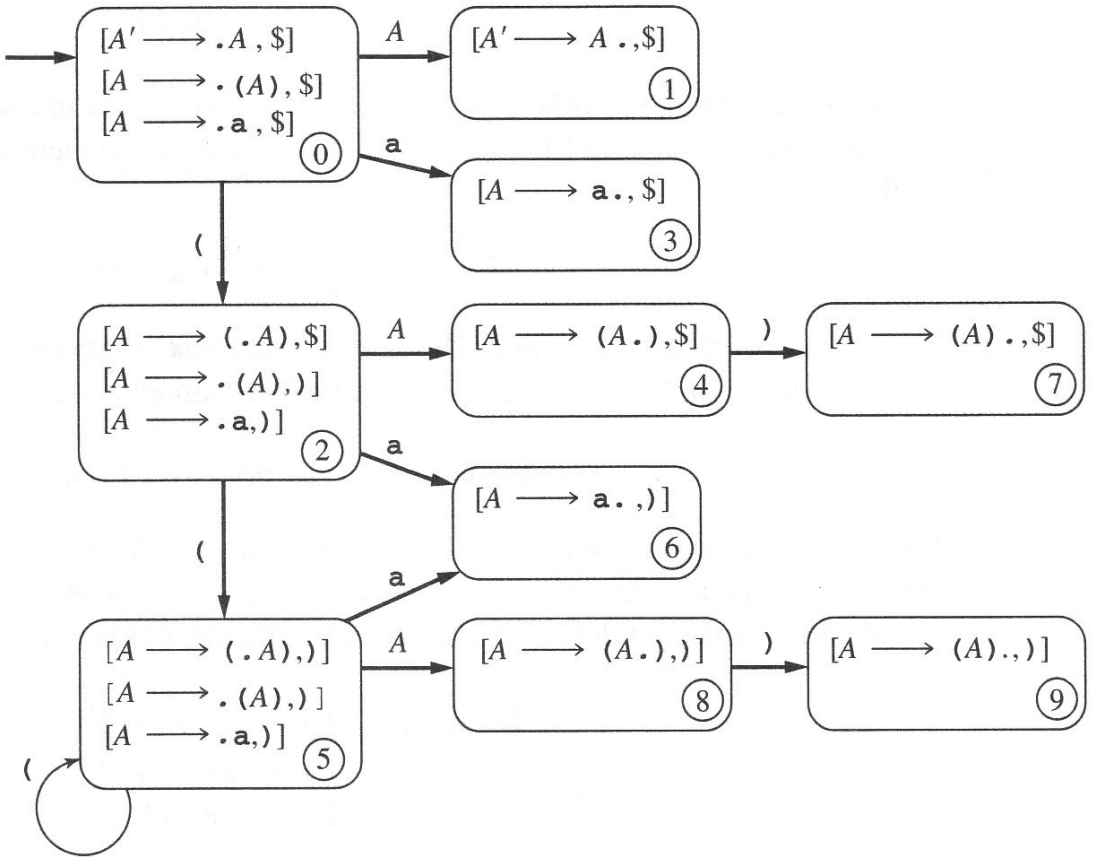
- Gramática: A→(A)|a
- Estado 0:
  - [A' → .A, $]
  - [A → .(A), $]
  - [A → .a, $]
- Estado 1: (Estado 0 con transición "A")
  - [A' → A., $]
- Estado 2: (Estado 0 con transición "(")
  - [A → (.A), $]
  - [A → .(A),)]
  - [A → .a,)]
- Estado 3: (Estado 0 con transición "a")
  - [A → a., $]

Análisis sintáctico LALR(1)

- 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)

- Autómata:



Construcción del autómata LALR(1)

- El algoritmo de construcción del autómata será:
  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 Є. (Las transiciones Є ya no se realizan porque son representadas por la cerradura).
        a. 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)

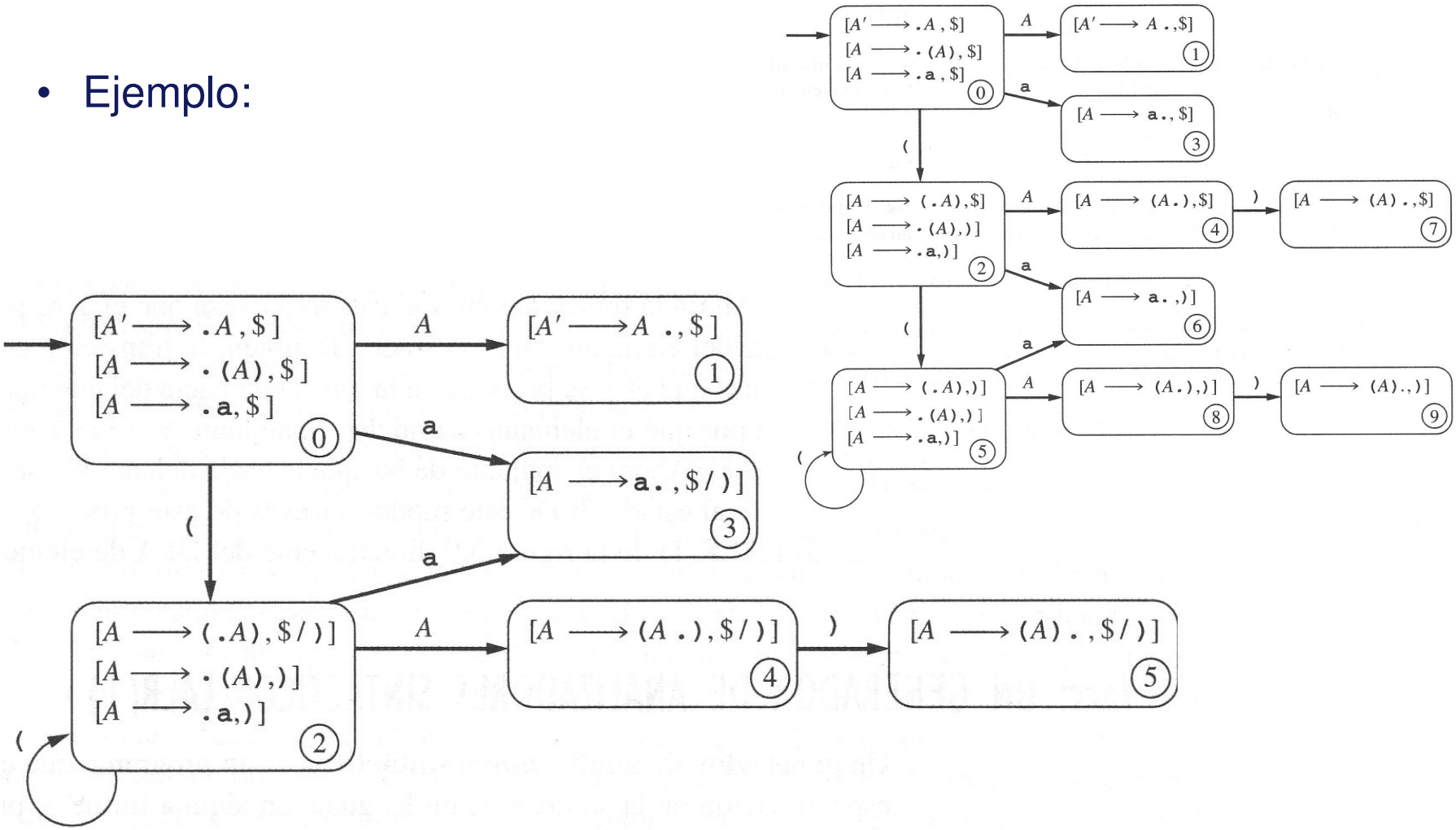
- El análisis LALR(1) se basa en que el tamaño del autómata LR(1) puede simplificarse unificando todos los estados con igual núcleo para los que existen varios elementos de búsqueda hacia delante.

- 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 S1 y S2 del autómata LR(1) con el mismo núcleo y,
  - Si hay una transición con un símbolo X desde S1 hasta T1,
  - Entonces también hay una transición con un símbolo X desde S2 hasta T2 y T1 y T2 tienen el mismo núcleo.
  - Por tanto se unifican S1 y S2 y T1 y T2.

Análisis sintáctico LALR(1)

- Ejemplo:



Análisis sintáctico LALR(1)

- El algoritmo de análisis es similar al SLR(1) pero utilizando 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 α 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)

- Tabla de análisis:

Estado	Entrada				Ir a
	(	a	)	$	A
0	s2	s3			1
1				aceptar	
2	s5	s6			4
3				r2	
4			s7		
5	s5	s6			8
6			r2		
7				r1	
8			s9		
9			r1		

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 métodos

- 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); más potente que LR(0) pero menos que LR(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 como CUP, Bison, y Yacc reconocen gramáticas LALR(1).
- Una gramática ambigua no es LL(1) ni SLR ni LR(1).

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.

- Los compiladores a la hora de informar del error, muestran la línea donde se comete el error y un mensaje explicativo del diagnóstico.

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.
- Es un método muy sencillo, y no puede producir bucles infinitos de recuperación, pero desecha muchos símbolos de la entrada.

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:

$$= \quad \rightarrow \quad :=$$

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

Recuperación de errores

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

  `A → error`

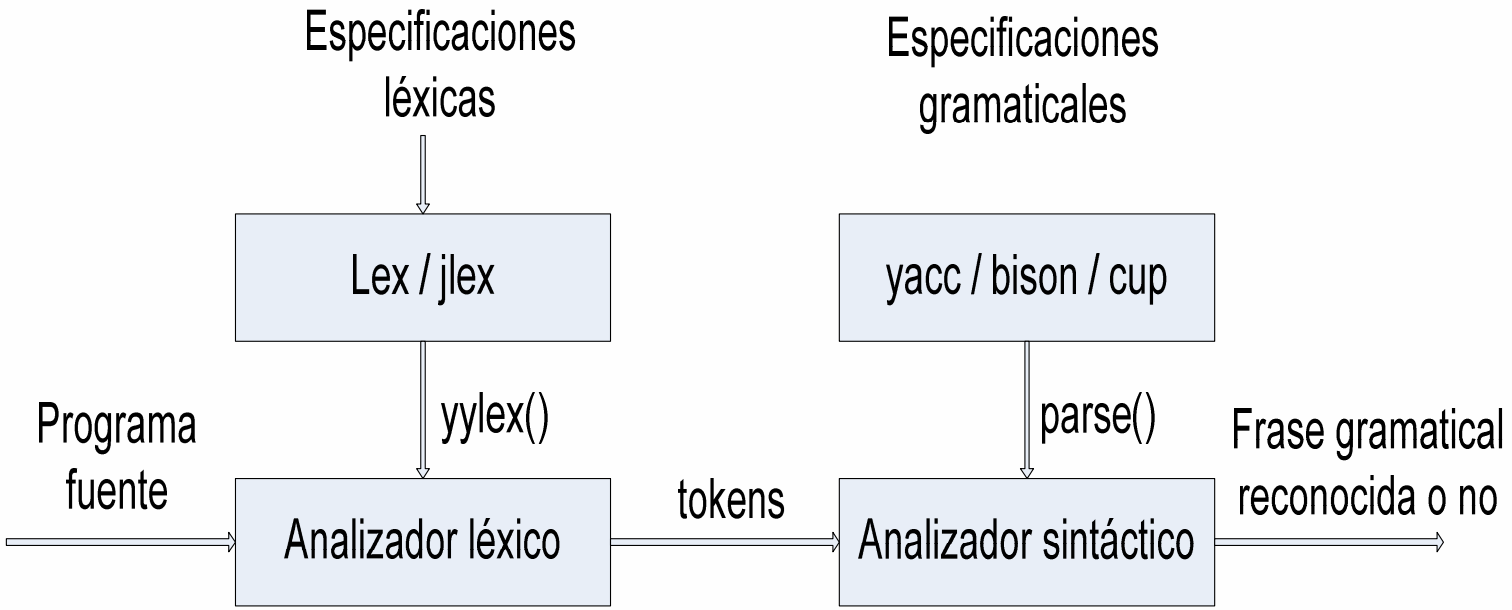
- Marca un contexto en el que los *tokens* de error pueden eliminarse hasta que se encuentre uno correcto.
- Se amplia la gramática con reglas de error que aparezcan con frecuencia y en cada regla se aplica una recuperación especifica.
- Principal método disponible en YACC/CUP para la recuperación de errores.

Generadores automáticos

- Un generador automático de analizadores sintácticos es un programa que toma como entrada la especificación de un lenguaje y proporciona un procedimiento de análisis sintáctico para el mismo.
- Históricamente denominados “compiladores de compiladores”:
  - YACC: *Yet another compiler-compiler*
- Implementaciones de dominio público (GNU):
  - Bison (C/C++)
  - CUP (Java)

Generadores automáticos

- Esquema de funcionamiento combinado:



Bibliografía

- *Básica:*
  - *Compiladores: principios, técnicas y herramientas*. A.V. Aho, R. Sethi, J.D. Ullman. Addison-Wesley Iberoamerica. 1990.
  - *Construcción de compiladores. Principios y práctica.* Kenneth C. Louden. Thomson-Paraninfo. 2004.

- *Complementaria:*
  - *Modern Compiler Implementation in Java*, A.W. Appel, Cambridge University Press, NY, ISBN 0-521-58388-8
  - *Introduction to compiling techniques: a first course using ANSI C, LEX and YACC. Bennet. McGraw-Hill International (UK). 1990.*