Análisis Ascendente

César Ignacio García Osorio
Área de Lenguajes y Sistemas Informáticos
Universidad de Burgos

Esquema

- Introducción
- Analizadores sintácticos por desplazamiento/reducción
- Conjuntos de ítems LR(0)
- Análisis sintáctico LR
- Obtención de tablas SLR(1)
- Conjuntos de ítems y tablas LR(1)
- Obtención de tablas LALR(1)
- Uso de gramáticas ambiguas
- Recuperación de errores

Introducción 1

- Un estilo general de análisis ascendente es el conocido como análisis sintáctico por desplazamiento/reducción
- Existen dos técnicas generales de análisis por desplazamiento/reducción:
  - Análisis sintáctico LR.
  - Análisis sintáctico por precedencia simple y de operadores.
- Existen tres tipos ampliamente usados de analizadores (parsers) LR por desplazamiento-reducción: SLR(k), LALR(k) y LR(k) canónico (y SLALR(k) o NQLALR(k)).

Introducción

- El análisis sintáctico por desplazamiento/reducción consiste en «*reducir*» una cadena *w* al símbolo inicial de la gramática.
- En cada paso de reducción se sustituye una subcadena determinada que concuerde con el lado derecho de una producción por el símbolo del lado izquierdo de dicha producción.
- Si en cada paso se elige correctamente la cadena, se consigue una derivación por la derecha en sentido inverso.

Ejemplo:

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Derivación más a la derecha:

$\underline{S} \Rightarrow aA\underline{B}e \Rightarrow a\underline{A}de \Rightarrow a\underline{A}bcde \Rightarrow abbcde$

$abbcde$

$aA\underline{bcde}$

$aA\underline{de}$

$\underline{aABe}$

S

Ejemplo

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow \mathbf{id}$

PILA	ENTRADA	SALIDA
\$	id + id * id\$	desplazar
\$id	+ id * id\$	reducir por $F \rightarrow$ id
\$$F$	+ id * id\$	reducir por $T \rightarrow F$
\$$T$	+ id * id\$	reducir por $E \rightarrow T$
\$$E$	+ id * id\$	desplazar
\$$E$ +	id * id\$	desplazar
\$$E$ + id	* id\$	reducir por $F \rightarrow$ id
\$$E$ + F	* id\$	reducir por $T \rightarrow F$
\$$E$ + T	* id\$	desplazar
\$$E$ + T *	id\$	desplazar
\$$E$ + T * id	\$	reducir por $F \rightarrow$ id
\$$E$ + T * F	\$	reducir por $T \rightarrow T * F$
\$$E$ + T	\$	reducir por $E \rightarrow E + T$
\$$E$	\$	aceptar

Introducción

- El análisis ascendente se basa en la identificación de asideros.
  - **Asidero**: subcadena de forma sentencial que concuerda con el lado derecho de una producción y cuya reducción al no terminal del lado izquierdo representa un paso a lo largo de la inversa de una derivación por la derecha.
  - **Prefijo viable**: Sea *xhy* una forma sentencial por la derecha con *h* un asidero. Se dice que *r* es un *prefijo viable* de *xhy* sii *xh=rs* (*s* puede ser ε). Es decir, *r* a lo sumo puede incluir *h*.

Analizador sintáctico por desplazamiento/reducción 1

- Un analizador sintáctico LR por desplazamiento/reducción consta de una entrada, una salida, una pila, un programa conductor y una tabla de análisis sintáctico con dos partes (*acción* y *goto*).
- El programa conductor es el mismo para todos los analizadores sintácticos LR; sólo cambian la forma de obtener las tablas de un tipo de análisis a otro.

Analizador sintáctico por desplazamiento/reducción 2

- El programa utiliza una pila para almacenar una cadena de la forma $s_0X_1s_1X_2s_2... X_ms_m$ Donde s_m está en la cima. Cada X_i es un símbolo gramatical y cada s_i es un símbolo llamado estado.
- Cada símbolo de estado resume la información contenida debajo de él en la pila, y se usan la combinación del símbolo de estado en la cima de la pila y el símbolo en curso de la entrada para indexar la tabla del análisis sintáctico y determinar la decisión de desplazamiento o reducción del analizador (los símbolos gramaticales en la práctica no son necesarios).

Analizador sintáctico por desplazamiento/reducción 3

- En cada paso cuatro opciones:
  - Poner el *token* actual en la cima de la pila y llamar al analizador léxico para obtener un nuevo *token* (SHIFT).
  - Decidir que los símbolos en la cima de la pila forman un asidero, y entonces desapilarlos y reemplazarlos por la parte izquierda de su producción (REDUCE).
  - Finalizar aceptando la cadena de entrada.
  - Finalizar señalando un error.
- Pueden surgir conflictos SHIFT-REDUCE y conflictos REDUCE-REDUCE que en algunos casos podemos resolver mirando el siguiente símbolo de la entrada.

Configuraciones LR(0) 1

- Configuraciones, ítems marcados o elementos del análisis sintáctico LR(0): es una producción con un punto en alguna posición del lado derecho.
  - Ejemplo: la producción $A \rightarrow XYZ$ da lugar a cuatro ítems marcados: $[A \rightarrow \bullet XYZ]$, $[A \rightarrow X \bullet YZ]$, $[A \rightarrow XY \bullet Z]$, $[A \rightarrow XYZ \bullet]$.
- Los ítems marcados se van a agrupar en estados de un autómata capaz de reconocer asideros en la cima de la pila.

Configuraciones LR(0) 2

core

- Los estados del autómata está compuesto de una serie de elementos nucleares y elementos no nucleares.
- Operaciones elementales para diseñar el autómata:
  - **Inicio**: Se amplia la gramática con un nuevo axioma y la producción $S' \rightarrow S\#$. Formar el núcleo del primer conjunto de items (estado del autómata).
    $[S' \rightarrow \bullet S\#] \in Set(0)$
  - **Cierre**: Completar el núcleo de un cto de items. Si el símbolo que precede la marca es un no terminal, se añaden sus reglas con la marca al principio.
    Si $[A \rightarrow x \bullet By] \in Set(n) \wedge B \in \mathcal{N} \Rightarrow [B \rightarrow \bullet w] \in Set(n) \ \forall \ B \rightarrow w \in \mathcal{P}$
  - **Lectura**: Obtener conjuntos de items a partir de otros conjuntos de items.
    $\forall [A \rightarrow x \bullet Xy] \in Set(n) \Rightarrow [A \rightarrow xX \bullet y] \in Set(n')$

Ej. obtención ctos items 1

1 $G \rightarrow E\#$
2 $E \rightarrow E+T$
3 $E \rightarrow E-T$
4 $E \rightarrow T$
5 $T \rightarrow a$
6 $T \rightarrow (E)$

Set(0)

$G \rightarrow \bullet E\#$
$E \rightarrow \bullet E+T$ $E \rightarrow \bullet E-T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet a$ $T \rightarrow \bullet (E)$

Set(1)

$T \rightarrow a\bullet$

Set(2)

$T \rightarrow (\bullet E)$
$E \rightarrow \bullet E+T$ $E \rightarrow \bullet E-T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet a$ $T \rightarrow \bullet (E)$

Set(3)

$G \rightarrow E\bullet\#$ $E \rightarrow E\bullet+T$ $E \rightarrow E\bullet-T$

Set(4)

$E \rightarrow T\bullet$

Set(5)

$T \rightarrow (E\bullet)$ $E \rightarrow E\bullet+T$ $E \rightarrow E\bullet-T$

Set(6)

$E \rightarrow E+\bullet T$
$T \rightarrow \bullet a$ $T \rightarrow \bullet (E)$

Set(7)

$E \rightarrow E-\bullet T$
$T \rightarrow \bullet a$ $T \rightarrow \bullet (E)$

Set(8)

$T \rightarrow (E)\bullet$

Set(9)

$E \rightarrow E+T\bullet$

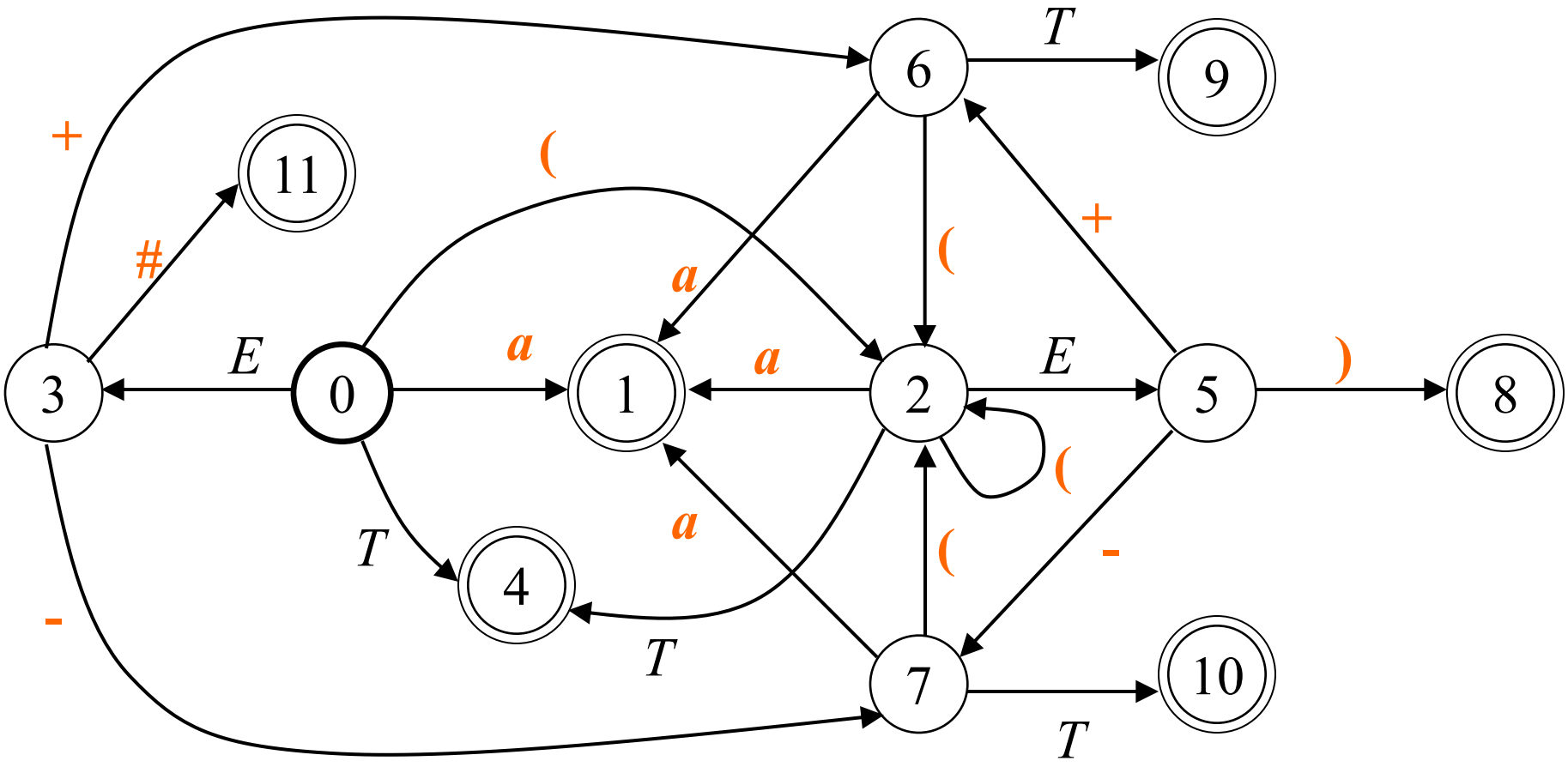
Set(10)

$E \rightarrow E-T\bullet$

Set(11)

$G \rightarrow E\#\bullet$

Ej. obtención ctos items 2



Algoritmo de análisis sintáctico LR 1

- La tabla *acción*, A[*s*,*a*], indica si desplazar, reducir, aceptar o señalar un error cuando el estado *s* está en la cima de la pila y *a* es el siguiente componente léxico de la entrada.
- La tabla *goto*, G[*s*,*X*], da el nuevo estado a poner en la cima de la pila. Esta función, para los terminales está codificada en la propia tabla acción.
- Para las acciones se van a utilizar los siguientes códigos:
  - ***di*** significa desplazar y meter en la pila el estado *i*,
  - ***rj*** significa reducir por la producción con número *j*,
  - ***acep*** significa aceptar (en alguna transparencia uso *A*!),
  - el espacio en blanco significa error.

Algoritmo de análisis sintáctico LR 2

Entrada: Una cadena *w* y las tablas de análisis sintáctico LR *acción* y *goto* para la gramática *G*.

Salida: Si *w* está en *L*(*G*), un análisis sintáctico ascendente de *w*; sino indica error.

Método: Inicialmente, en la pila está el estado inicial 0, y *w*# en la entrada.

apuntar *ae* al primer símbolo de *w*#

repeat forever begin

sea *s* el estado en la cima de la pila y *a* el símbolo apuntado por *ae*;

if A[*s*, *a*]=*desplazar s'* **then** meter *a* y después *s'* en la cima de la pila;
avanzar *ae* al siguiente símbolo de entrada;

else if A[*s*, *a*]=*reducir* $A \rightarrow \beta$ **then** sacar $2 \times |\beta|$ símbolos de la pila;
sea *s'* el estado que ahora está en la cima de la pila;
meter *A* y después G[*s'*, *A*] en la cima de la pila;
emitir la producción $A \rightarrow \beta$;

else if A[*s*, *a*]=*aceptar* **then return**

else *error*()

end

Algoritmo de análisis sintáctico LR

Ejemplo

Estado	Acción							Goto		
	a	+	-	(	)	#		*G*	*E*	*T*
0	*d*1			*d*2					3	4
1	*r*5	*r*5	*r*5	*r*5	*r*5	*r*5				
2	*d*1			*d*2					5	4
3		*d*6	*d*7			*d*11				
4	*r*4	*r*4	*r*4	*r*4	*r*4	*r*4				
5		*d*6	*d*7		*d*8					
6	*d*1			*d*2						9
7	*d*1			*d*2						10
8	*r*6	*r*6	*r*6	*r*6	*r*6	*r*6				
9	*r*2	*r*2	*r*2	*r*2	*r*2	*r*2				
10	*r*3	*r*3	*r*3	*r*3	*r*3	*r*3				
11	*a*	*c*	*e*	*p*	*t*	*a*	*r*			

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

Algoritmo de análisis sintáctico LR

Ejemplo

Estado	Acción						Goto		
	a	+	-	(	)	#	*G*	*E*	*T*
0	*d*1			*d*2				3	4
1		*r*5	*r*5		*r*5	*r*5			
2	*d*1			*d*2				5	4
3		*d*6	*d*7			*acep*			
4		*r*4	*r*4		*r*4	*r*4			
5		*d*6	*d*7		*d*8				
6	*d*1			*d*2					9
7	*d*1			*d*2					10
8		*r*6	*r*6		*r*6	*r*6			
9		*r*2	*r*2		*r*2	*r*2			
10		*r*3	*r*3		*r*3	*r*3			
11	//////////////////								

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

Ejemplo conflicto SHIFT-REDUCE

1 $S' \rightarrow S\#$
2 $S \rightarrow Sa$
3 $S \rightarrow B$
4 $B \rightarrow b$
5 $B \rightarrow bB$

Set(0)

$S' \rightarrow \bullet S\#$
$S \rightarrow \bullet Sa$ $S \rightarrow \bullet B$ $B \rightarrow \bullet b$ $B \rightarrow \bullet bB$

Set(1)

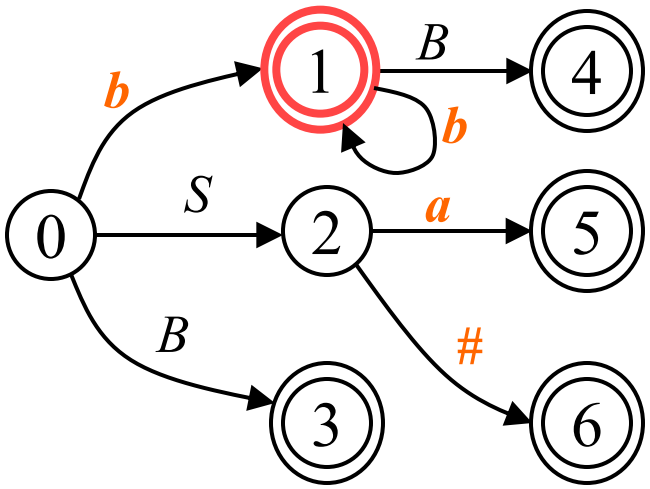
$B \rightarrow b\bullet$ $B \rightarrow b\bullet B$
$B \rightarrow \bullet b$ $B \rightarrow \bullet bB$

Set(2)

$S' \rightarrow S\bullet\#$ $S \rightarrow S\bullet a$

Set(3)

$S \rightarrow B\bullet$



Conflicto SHIFT-REDUCE

Set(4)

$B \rightarrow bB\bullet$

Set(5)

$S \rightarrow Sa\bullet$

Set(6)

$S' \rightarrow S\#\bullet$

Conflictos durante el análisis sintáctico por desplazamiento/reducción

- Conflicto SHIFT/REDUCE

prop → **if** *expr* **then** *prop*
| **if** *expr* **then** *prop* **else** *prop*
| **otro**

- Ante una situación como:

PILA	ENTRADA
...**if** *expr* **then** *prop*	**else** ... **$**

- Además esta gramática es ambigua, y en general, ninguna gramática ambigua puede ser LR(*k*) para ningún *k*.

- Conflicto REDUCE/REDUCE

prop → **id(***lista_params***)**
| *expr* **:=** *expr*
lista_params → *lista_params* **,** *parámetro*
| *parámetro*
parámetro → **id**
expr → **id(***lista_expr***)**
| **id**
lista_expr → *lista_expr* **,** *expr*
| *expr*

- Ante una situación como:

PILA	ENTRADA
...**id (id**	**,id)**... **$**

Análisis SLR 1

- Es la técnica más sencilla que permite un análisis LR con previsión de *LookAhead* de un símbolo.
- Clase gramatical LR(1) más simple. ($SLR(1) \subset LR(1)$)
- Ante un conflicto SHIFT-REDUCE: un ítem $[A \rightarrow \alpha_1 \bullet X \alpha_2]$ y otro $[B \rightarrow \delta \bullet]$ en el mismo conjunto, se calcula el $FOLLOW(B)$, si no incluye el símbolo de detrás de la marca en el ítem $[A \rightarrow \alpha_1 \bullet X \alpha_2]$ puedo resolver el conflicto.
- Ante un conflicto REDUCE-REDUCE: $[A \rightarrow \delta \bullet]$ y $[B \rightarrow \gamma \bullet]$ en el mismo conjunto, si son disjuntos $FOLLOW(A)$ y $FOLLOW(B)$ puedo resolverlo.
  ($S \Rightarrow^* \phi A t \Rightarrow \phi \delta t$ pero $S \Rightarrow^* \phi B u \Rightarrow \phi \gamma u$)

Construcción tabla de Análisis SLR

Entrada: Un conjunto $C=\{C_0, C_1, \ldots C_m\}$ de conjuntos de ítems marcados $LR(0)$

Salida: A(*acción*) A: $\mathcal{Q}\times\Sigma \longrightarrow \{d, acep, e\} \cup \{ri\}$

G(*goto*) G: $\mathcal{Q}\times (\Sigma\cup\mathcal{N}) \longrightarrow \{E\}\cup\mathcal{Q}$

Método:

1. Los estados $\mathcal{Q}=\{0, 1, ..., m\}$ se construyen a partir de C ($C_i \rightarrow i$).
2. **forall** $i \in \mathcal{Q}$

   if Ci —lectura $x \rightarrow Cj$ **then** G[i,x] = j

   if $[A \rightarrow a \bullet x\beta] \in Ci \ \wedge \ x \in \Sigma$

   then A[i,x] $\supset \{d\, j\}$

   if $[A \rightarrow \gamma \bullet] \in Ci \ \wedge A \rightarrow \gamma$ es la producción j-ésima

   then forall $x \in FOLLOW(A)$ A[i,x] $\supset \{rj\}$

   if $[S' \rightarrow S \bullet \#] \in Ci$

   then A[i, #] $\supset \{acep\}$

LR(1) canónico 1

- Configuraciones, ítems marcados o elementos del análisis sintáctico LR(1): es un objeto con dos componentes: $[A \rightarrow \alpha \bullet \beta \ , \ u]$
  - una producción con marca (ítem LR(0))
  - un conjunto u de símbolos de (previsión/*LookAhead*)
- La posición de la marca va indicando los prefijos viables. Los desplazamientos de la marca van aumentando la longitud del prefijo viable. Lo que está detrás del asidero debe corresponderse con algún elemento de u.

LR(1) canónico 2

- La construcción de ítem LR(1) es muy parecida a la de ítems LR(0) ya vista. En la lectura, me olvido de la *u*, simplemente la arrastro. En el inicio, hago $[S' \rightarrow \bullet S\#, \#] \in Set(0)$. La que cambia es la operación de cierre.
- **Cierre**: Completar el núcleo de un cto de ítems.

  Si $[A \rightarrow x \bullet By, u] \in Set(n) \wedge B \in \mathcal{N} \wedge B \rightarrow w \in \mathcal{P}$

  $\Rightarrow [B \rightarrow \bullet w, v] \in Set(n)\ \forall v \in FIRST(yu)$

  Si $[A \rightarrow x \bullet By; u_1, \ldots, u_k] \in Set(n) \wedge B \in \mathcal{N} \wedge B \rightarrow w \in \mathcal{P}$

  $\Rightarrow [B \rightarrow \bullet w; FIRST(yu_1) \cup \ldots \cup FIRST(yu_k)] \in Set(n)$

Ej. obtención conjuntos de ítems LR(1)

1 $G \rightarrow S\#$
2 $S \rightarrow CC$
3 $C \rightarrow cC$
4 $C \rightarrow d$

Set(0)

$G \rightarrow \bullet S\#,\ \#$
$S \rightarrow \bullet CC,\ \#$ $C \rightarrow \bullet cC,\ c$ $C \rightarrow \bullet cC,\ d$ $C \rightarrow \bullet d,\ c$ $C \rightarrow \bullet d,\ d$

Set(1)

$G \rightarrow S\bullet\#,\ \#$

Set(2)

$S \rightarrow C\bullet C,\ \#$
$C \rightarrow \bullet cC,\ \#$ $C \rightarrow \bullet d,\ \#$

Set(3)

$C \rightarrow c\bullet C,\ c$ $C \rightarrow c\bullet C,\ d$
$C \rightarrow \bullet cC,\ c$ $C \rightarrow \bullet cC,\ d$ $C \rightarrow \bullet d,\ c$ $C \rightarrow \bullet d,\ d$

Set(4)

$C \rightarrow d\bullet,\ c$ $C \rightarrow d\bullet,\ d$

Set(5)

$S \rightarrow CC\bullet,\ \#$

Set(6)

$C \rightarrow c\bullet C,\ \#$
$C \rightarrow \bullet cC,\ \#$ $C \rightarrow \bullet d,\ \#$

Set(7)

$C \rightarrow d\bullet,\ \#$

Set(8)

$C \rightarrow cC\bullet,\ c$ $C \rightarrow cC\bullet,\ d$

Set(9)

$C \rightarrow cC\bullet,\ \#$

Ej. obtención conjuntos de ítems LR(1)

Si $[A \rightarrow x \bullet By;\ u_1,\ \ldots,\ u_k] \in Set(n) \wedge B \in \mathcal{N} \wedge B \rightarrow w \in \mathcal{P}$
$\Rightarrow [B \rightarrow \bullet w;\ FIRST(yu_1) \cup \ldots \cup FIRST(yu_k)] \in Set(n)$

1 $G \rightarrow S\#$
2 $S \rightarrow CC$
3 $C \rightarrow cC$
4 $C \rightarrow d$

Set(0)

$G \rightarrow \bullet S\#;\ \#$
$S \rightarrow \bullet CC,\ \#$ $C \rightarrow \bullet cC;\ c,\ d$ $C \rightarrow \bullet d;\ c,\ d$

Set(1)

$G \rightarrow S \bullet \#;\ \#$

Set(2)

$S \rightarrow C \bullet C;\ \#$
$C \rightarrow \bullet cC;\ \#$ $C \rightarrow \bullet d;\ \#$

Set(3)

$C \rightarrow c \bullet C;\ c,\ d$
$C \rightarrow \bullet cC;\ c,\ d$ $C \rightarrow \bullet d;\ c,\ d$

Set(4)

$C \rightarrow d \bullet;\ c,\ d$

Set(5)

$S \rightarrow CC \bullet;\ \#$

Set(6)

$C \rightarrow c \bullet C;\ \#$
$C \rightarrow \bullet cC;\ \#$ $C \rightarrow \bullet d;\ \#$

Set(7)

$C \rightarrow d \bullet;\ \#$

Set(8)

$C \rightarrow cC \bullet;\ c,\ d$

Set(9)

$C \rightarrow cC \bullet;\ \#$

LR(1) canónico

Entrada: Un conjunto $C=\{C_0, C_1, \ldots C_m\}$ de conjuntos de ítems marcados $LR(1)$

Salida: A(*acción*) A: $\mathcal{Q}\times\Sigma \longrightarrow \{d,acep, e\} \cup \{ri\}$

G(*goto*) G: $\mathcal{Q}\times (\Sigma\cup N) \longrightarrow \{E\}\cup\mathcal{Q}$

Método:

1. Los estados $\mathcal{Q}=\{0, 1, ..., m\}$ se construyen a partir de C ($C_i \rightarrow i$).
2. **forall** $i \in \mathcal{Q}$

   if Ci —lectura $x\rightarrow Cj$ **then** $G[i,x] = j$

   if $[A \rightarrow a\bullet x\beta, u] \in Ci \ \wedge \ x\in\Sigma$

   then $A[i,x] \supset \{d\, j\}$

   if $[A \rightarrow \gamma\bullet, u] \in Ci \ \wedge A\rightarrow\gamma$ es la producción j-ésima

   then $A[i,u] \supset \{rj\}$

   if $[S' \rightarrow S\bullet\#, \#] \in Ci$

   then $A[i,\#] \supset \{acep\}$

LR(1) canónico

Entrada: Un conjunto $C=\{C_0, C_1, \ldots C_m\}$ de conjuntos de ítems marcados $LR(1)$

Salida: A(*acción*) A: $\mathcal{Q}\times\Sigma \longrightarrow \{d,acep, e\} \cup \{ri\}$

G(*goto*) G: $\mathcal{Q}\times (\Sigma\cup N) \longrightarrow \{E\}\cup\mathcal{Q}$

Método:

1. Los estados $\mathcal{Q}=\{0, 1, ..., m\}$ se construyen a partir de C ($C_i \rightarrow i$).
2. **forall** $i \in \mathcal{Q}$

   if Ci —lectura $x\rightarrow Cj$ **then** G[i,x] = j

   if $[A \rightarrow a\bullet x\beta; u_1, \ldots, u_k] \in Ci \ \wedge \ x\in\Sigma$

   then A[i,x] $\supset \{d\, j\}$

   if $[A \rightarrow \gamma\bullet; u_1, \ldots, u_l, \ldots, u_k] \in Ci \ \wedge A\rightarrow\gamma$ es la producción j-ésima

   then A[i,u_l] $\supset \{rj\}$ para $l = 1, \ldots, k$

   if $[S' \rightarrow S\bullet\#; \#] \in Ci$

   then A[i,#] $\supset \{acep\}$

Tablas LR(1)

Estado	Acción			Goto	
	c	*d*	#	*S*	*C*
0	*d*3	*d*4		1	2
1			*acep*		
2	*d*6	*d*7			5
3	*d*3	*d*4			8
4	*r*4	*r*4			
5			*r*2		
6	*d*6	*d*7			9
7			*r*4		
8	*r*3	*r*3			
9			*r*3		

1 $G \rightarrow S\#$

2 $S \rightarrow CC$

3 $C \rightarrow cC$

4 $C \rightarrow d$

LALR(1) 1

- El análisis LR(1) canónico requiere muchos más estados que el análisis SLR(1). En lenguajes de programación reales la diferencia puede ser un factor de 10. Por el contrario, hay algunas construcciones de los lenguajes de programación que dan problemas en el análisis SLR(1).
- El análisis LALR(1) combina las ventajas de los dos mundos. Puede tratar con las gramáticas problemáticas para el análisis SLR(1), y requiere menos estados que en el análisis LR(1) canónico.

LALR(1) 2

- Tras obtener los conjuntos LR(1), se identifican aquellos en los que los ítems del núcleo sólo difieren en los *lookahead* y se fusionan, haciendo que su *lookahead* sea la unión.
- A continuación, se aplica a los nuevos conjuntos de ítems marcados el mismo algoritmo de obtención de tablas que el visto para LR(1) canónico.
- Pueden surgir conflictos REDUCE/REDUCE que antes no existían en un conjunto C si

$$[X \to \alpha\bullet, l_1] \in C \wedge [Y \to \beta\bullet, l_2] \in C \wedge \ l_1 \cap l_2 \neq \varnothing.$$

- Si surgen conflictos, la gramática no es LALR(1).

Uso de gramáticas ambiguas: introducción

- En ocasiones es útil usar construcciones ambiguas siempre que se haga de manera estrictamente controlada.
- Por ejemplo, una gramática para expresiones ambigua proporciona una especificación más natural y corta que cualquier gramática no ambigua equivalente.
- Otro ejemplo, la ambigüedad del *else* «descolgado».
- Las gramáticas ambiguas producen conflictos (desplazamiento/reducción o reducción/reducción), pero podemos utilizar información adicional para resolverlos.

Uso de gramática ambiguas: expresiones (1)

- Considérese las expresiones en los lenguajes de programación.

$$E \rightarrow E{+}E \mid E{*}E \mid (E) \mid \mathbf{id}$$

$$E \rightarrow E{+}T \mid T$$
$$T \rightarrow T{*}F \mid F$$
$$F \rightarrow (E) \mid \mathbf{id}$$

- Se prefiere la primera:
  - Se pueden cambiar asociatividades y precedencias sin interferir en las producciones.
  - El analizador para la segunda consume un tiempo importante para las producciones $E \rightarrow T$ y $T \rightarrow F$, cuya única función es asegurar la asociatividad y precedencia.

Uso de gramática ambiguas: expresiones (2)

I_0

$E' \rightarrow \bullet E$

$E \rightarrow \bullet E+E$

$E \rightarrow \bullet E*E$

$E \rightarrow \bullet(E)$

$E \rightarrow \bullet \mathbf{id}$

I_1

$E' \rightarrow E\bullet$

$E \rightarrow E\bullet+E$

$E \rightarrow E\bullet*E$

I_2

$E \rightarrow (\bullet E)$

$E \rightarrow \bullet E+E$

$E \rightarrow \bullet E*E$

$E \rightarrow \bullet(E)$

$E \rightarrow \bullet \mathbf{id}$

I_3

$E \rightarrow \mathbf{id}\bullet$

I_4

$E \rightarrow E+\bullet E$

$E \rightarrow \bullet E+E$

$E \rightarrow \bullet E*E$

$E \rightarrow \bullet(E)$

$E \rightarrow \bullet \mathbf{id}$

I_5

$E \rightarrow E*\bullet E$

$E \rightarrow \bullet E+E$

$E \rightarrow \bullet E*E$

$E \rightarrow \bullet(E)$

$E \rightarrow \bullet \mathbf{id}$

I_6

$E \rightarrow (E\bullet)$

$E \rightarrow E\bullet+E$

$E \rightarrow E\bullet*E$

I_7

$E \rightarrow E+E\bullet$

$E \rightarrow E\bullet+E$

$E \rightarrow E\bullet*E$

I_8

$E \rightarrow E*E\bullet$

$E \rightarrow E\bullet+E$

$E \rightarrow E\bullet*E$

I_9

$E \rightarrow (E)\bullet$

FOLLOW(E')={$}

FOLLOW(E)={+,*,),$}

Uso de gramática ambiguas: expresiones (3)

	id	+	*	(	)	$	E
0	d3			d2			1
1		d4	d5			A!	
2	d3			d2			6
3		r4	r4		r4	r4	
4	d3			d2			7
5	d3			d2			8
6		d4	d5		d9		
7		**d4/r1**	**d5/r1**		r1	r1	
8		**d4/r2**	**d5/r2**		r2	r2	
9		r3	r3		r3	r3	

1. $E \rightarrow E+E$
2. $E \rightarrow E*E$
3. $E \rightarrow (E)$
4. $E \rightarrow \mathbf{id}$

Uso de gramática ambiguas: expresiones (3)

	id	+	*	(	)	$	E
0	*d*3			*d*2			1
1		*d*4	*d*5			*A*!	
2	*d*3			*d*2			6
3		*r*4	*r*4		*r*4	*r*4	
4	*d*3			*d*2			7
5	*d*3			*d*2			8
6		*d*4	*d*5		*d*9		
7		***r*1**	***d*5**		*r*1	*r*1	
8		***r*2**	***r*2**		*r*2	*r*2	
9		*r*3	*r*3		*r*3	*r*3	

1. $E \rightarrow E+E$
2. $E \rightarrow E*E$
3. $E \rightarrow (E)$
4. $E \rightarrow \mathbf{id}$

Uso de gramática ambiguas: ambigüedad del *else* (1)

- Considérese la siguiente gramática para las proposiciones condicionales.

$$S \rightarrow iS \mid iSeS \mid a$$

prop → **if** ***expr*** **then** ***prop***

| **if** ***expr*** **then** ***prop*** **else** ***prop***

| **otra** (otros tipos de proposición)

- Veamos que ocurre al calcular los conjuntos de ítems.

Uso de gramática ambiguas: ambigüedad del *else* (2)

I_0

$S' \rightarrow \bullet S$

$S \rightarrow \bullet iSeS$

$S \rightarrow \bullet iS$

$S \rightarrow \bullet a$

I_1

$S' \rightarrow S\bullet$

I_2

$S \rightarrow i \bullet SeS$

$S \rightarrow i \bullet S$

$S \rightarrow \bullet iSeS$

$S \rightarrow \bullet iS$

$S \rightarrow \bullet a$

I_3

$S \rightarrow a\bullet$

I_4

$S \rightarrow iS \bullet eS$

$S \rightarrow iS\bullet$

I_5

$S \rightarrow iSe \bullet S$

$S \rightarrow \bullet iSeS$

$S \rightarrow \bullet iS$

$S \rightarrow \bullet a$

I_6

$S \rightarrow iSeS\bullet$

$\text{FOLLOW}(S) = \{\$, e\}$

Uso de gramática ambiguas: ambigüedad del *else* (3)

	i	*e*	*a*	$	*S*
0	*d*2		*d*3		1
1				*A*!	
2	*d*2		*d*3		4
3		*r*3		*r*3	
4		***d*5/*r*2**		*r*2	
5	*d*2		*d*3		6
6		*r*1		*r*1	

1. $S \rightarrow iSeS$
2. $S \rightarrow iS$
3. $S \rightarrow a$

Uso de gramática ambiguas: ambigüedad del *else* (3)

	i	*e*	*a*	$	*S*
0	*d*2		*d*3		1
1				*A*!	
2	*d*2		*d*3		4
3		*r*3		*r*3	
4		***d*5**		*r*2	
5	*d*2		*d*3		6
6		*r*1		*r*1	

1. $S \rightarrow$ ***i*S*e*S**
2. $S \rightarrow$ ***i*S**
3. $S \rightarrow$ ***a***

La ambigüedad se resuelve eligiendo desplazar. Se puede ver que esta solución funciona analizando lo que ocurriría con la palabra ***iiaea***.

Recuperación de errores (1)

Introducción

- Un analizador sintáctico LR detectará un error cuando consulte la tabla de acciones de análisis sintáctico y encuentre una entrada vacía. Los errores nunca se detectan consultando la tabla de transiciones *ir_a*.
- Un analizador LR anunciará un error tan pronto como no haya una continuación válida para la parte de entrada examinada hasta entonces.
- Un analizador sintáctico LR canónico nunca hará ni una sola reducción antes de anunciar un error. Los analizadores SLR y LALR pueden hacer varias reducciones antes de anunciar un error, pero nunca desplazarán un símbolo de entrada erróneo a la pila.

Recuperación de errores (2)

Recuperación en modo pánico

- Se eliminan elementos de la pila hasta encontrar un estado *s* con un valor de *ir_a* para algún no terminal *A*.
- Entonces, se desechan cero o más símbolos de entrada hasta encontrar un símbolo *a* que pueda seguir legalmente a *A*.
- Entonces, el analizador sintáctico mete en la pila *A* y el estado *ir_a*[*s*,*A*] y prosigue el análisis sintáctico.
- **Puede haber más de una opción para el no terminal *A*.** Generalmente, estos serían no terminales que representan las partes más importantes de un programa, como una expresión, una proposición o un bloque. Por ejemplo, si *A* es el no terminal *prop*, entonces *a* puede ser el símbolo de punto y coma o el componente léxico **end**.

Recuperación de errores (3)

Recuperación en modo pánico

Siendo *s* el símbolo de tope de pila y *e* el siguiente símbolo de la entrada:

```
if action[s, e] is empty:
    for a in input:
        for s in reversed(stack):
            for A in 𝒩:
                if goto[goto[s, A], a] is not empty:
                    descartar input hasta a
                    sacar elementos de la pila hasta que s quede en top
                    meter A en la pila
                    meter goto[s, A] en la pila
                    continuar el análisis
```

Estado	Acción						Goto		
	a	**+**	**-**	**(**	**)**	**$**	*G*	*E*	*T*
0	*d*1			*d*2				3	4
1	*r*5	*r*5	*r*5	*r*5	*r*5	*r*5			
2	*d*1			*d*2				5	4
3		*d*6	*d*7			*acep*			
4	*r*4	*r*4	*r*4	*r*4	*r*4	*r*4			
5		*d*6	*d*7		*d*8				
6	*d*1			*d*2					9
7	*d*1			*d*2					10
8	*r*6	*r*6	*r*6	*r*6	*r*6	*r*6			
9	*r*2	*r*2	*r*2	*r*2	*r*2	*r*2			
10	*r*3	*r*3	*r*3	*r*3	*r*3	*r*3			

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

PILA	ENTRADA	MENSAJE DE ERROR Y ACCIONES
0	***a*** + + ***a*** **$**	
0***a***1	+ + ***a*** **$**	
0*T*4	+ + ***a*** **$**	$T \rightarrow a$
0*E*3	+ + ***a*** **$**	$E \rightarrow T$
0*E*3+6	+ ***a*** **$**	Busco en la pila un estado *s* y símbolo *A*, tales que el estado *s'*=*ir_a*(*s*,*A*) tiene una transición definida para algún símbolo de los pendientes de leer.

Estado	Acción						Goto		
	a	+	-	(	)	$	*G*	*E*	*T*
0	*d*1			*d*2				3	4
1	*r*5	*r*5	*r*5	*r*5	*r*5	*r*5			
2	*d*1			*d*2				5	4
3		*d*6	*d*7			*acep*			
4	*r*4	*r*4	*r*4	*r*4	*r*4	*r*4			
5		*d*6	*d*7		*d*8				
6	*d*1			*d*2					9
7	*d*1			*d*2					10
8	*r*6	*r*6	*r*6	*r*6	*r*6	*r*6			
9	*r*2	*r*2	*r*2	*r*2	*r*2	*r*2			
10	*r*3	*r*3	*r*3	*r*3	*r*3	*r*3			

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

Pila	Entrada	Mensaje de Error y Acciones
0	*a* + + *a* $	
0*a*1	+ + *a* $	
0*T*4	+ + *a* $	$T \rightarrow a$
0*E*3	+ + *a* $	$E \rightarrow T$
0*E*3+6	+ *a* $	Busco en la pila un estado *s* y símbolo *A*, tales que el estado $s'=ir\_a(s,A)$ tiene una transición definida para algún símbolo de los pendientes de leer.
0*E*3	+ *a* $	
0*E*3+6	*a* $	
0*E*3+6*a*1	$	$T \rightarrow a$
0*E*3+6*T*9	$	$E \rightarrow E + T$
0*E*3	$	aceptar

Pila	Entrada	Mensaje de Error y Acciones
0	*a* + (+ *a*) $	
0*a*1	+ (+ *a*) $	
0*T*4	+ (+ *a*) $	$T \rightarrow a$
0*E*3	+ (+ *a*) $	$E \rightarrow T$
0*E*3+6	(+ *a*) $	
0*E*3+6(2	+ *a*) $	Busco en la pila un estado *s* y símbolo *A*, tales que el estado *s'*=*ir_a*(*s*,*A*) tiene una transición definida para algún símbolo de los pendientes de leer.

Estado	Acción						Goto		
	a	+	-	(	)	$	*G*	*E*	*T*
0	*d*1			*d*2				3	4
1	*r*5	*r*5	*r*5	*r*5	*r*5	*r*5			
2	*d*1			*d*2				5	4
3		*d*6	*d*7			*acep*			
4	*r*4	*r*4	*r*4	*r*4	*r*4	*r*4			
5		*d*6	*d*7		*d*8				
6	*d*1			*d*2					9
7	*d*1			*d*2					10
8	*r*6	*r*6	*r*6	*r*6	*r*6	*r*6			
9	*r*2	*r*2	*r*2	*r*2	*r*2	*r*2			
10	*r*3	*r*3	*r*3	*r*3	*r*3	*r*3			

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

Estado	Acción						Goto		
	a	+	-	(	)	$	*G*	*E*	*T*
0	*d*1			*d*2				3	4
1	*r*5	*r*5	*r*5	*r*5	*r*5	*r*5			
2	*d*1			*d*2				5	4
3		*d*6	*d*7			*acep*			
4	*r*4	*r*4	*r*4	*r*4	*r*4	*r*4			
5		*d*6	*d*7		*d*8				
6	*d*1			*d*2					9
7	*d*1			*d*2					10
8	*r*6	*r*6	*r*6	*r*6	*r*6	*r*6			
9	*r*2	*r*2	*r*2	*r*2	*r*2	*r*2			
10	*r*3	*r*3	*r*3	*r*3	*r*3	*r*3			

1 $G \rightarrow E\#$

2 $E \rightarrow E+T$

3 $E \rightarrow E-T$

4 $E \rightarrow T$

5 $T \rightarrow a$

6 $T \rightarrow (E)$

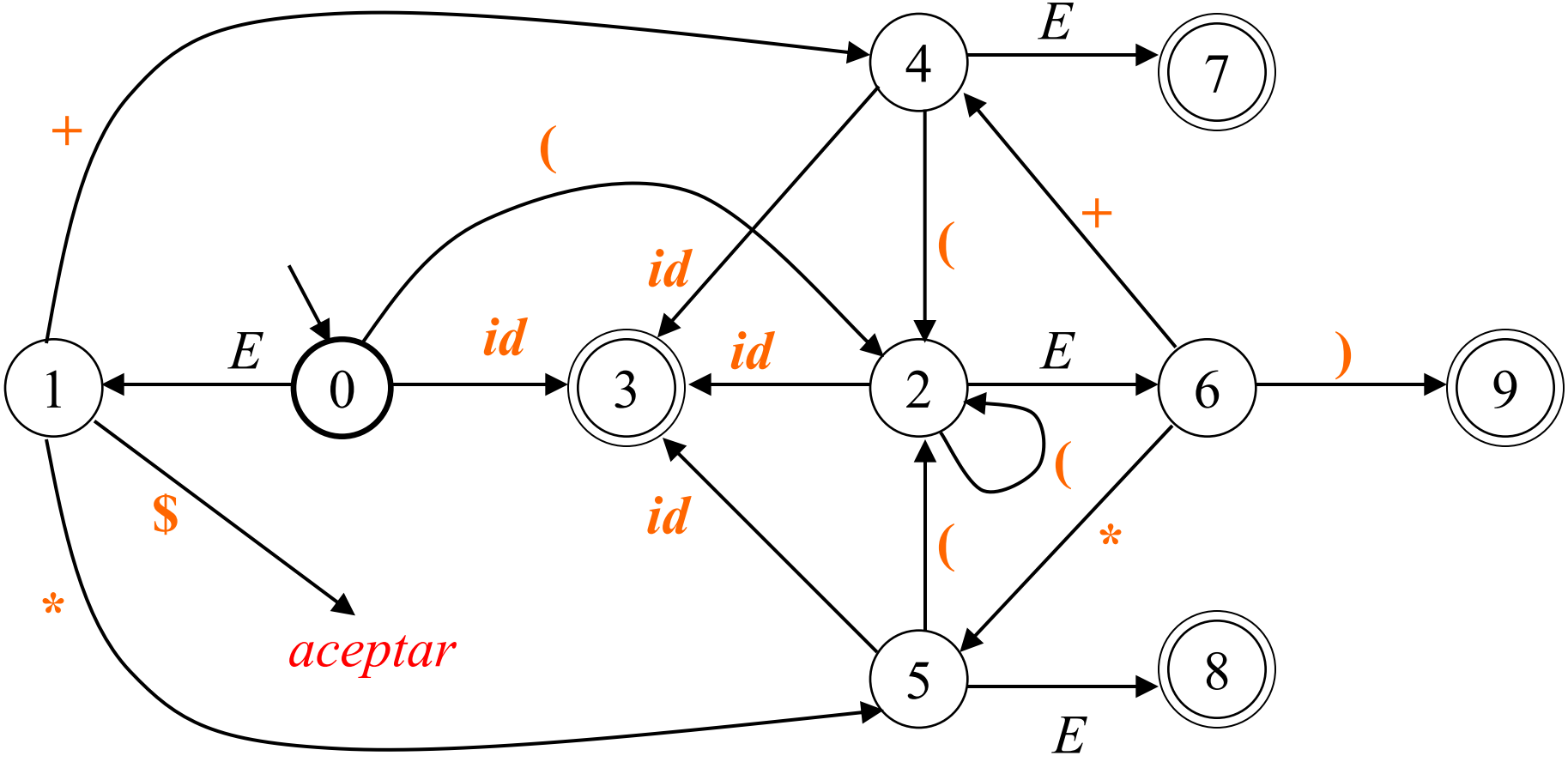
PILA	ENTRADA	MENSAJE DE ERROR Y ACCIONES
0	*a* + (+ *a*) $	
0*a*1	+ (+ *a*) $	
0*T*4	+ (+ *a*) $	$T \rightarrow a$
0*E*3	+ (+ *a*) $	$E \rightarrow T$
0*E*3+6	(+ *a*) $	
0*E*3+6(2	+ *a*) $	Busco en la pila un estado *s* y símbolo *A*, tales que el estado *s'*=*ir_a*(*s*,*A*) tiene una transición definida para algún símbolo de los pendientes de leer.
0*E*3+6(2*E*5	+ *a*) $	
0*E*3+6(2*E*5+6	*a*) $	
0*E*3+6(2*E*5+6*a*1	) $	$T \rightarrow a$
0*E*3+6(2*E*5+6*T*4	) $	$E \rightarrow T$
... ...	...	

Recuperación de errores (3)

Recuperación a nivel de frase

- Se implementa examinando cada entrada de error en la tabla de análisis sintáctico LR y decidiendo, basándose en el uso del lenguaje, los errores de los programadores que más probablemente darían lugar a situación.
- Entonces, se puede construir un procedimiento de recuperación apropiado; presumiblemente, el tope de la pila o los primeros símbolos de entrada o ambos se modificarían de la forma adecuada a cada entrada de error.

1. $E \rightarrow E+E$
2. $E \rightarrow E*E$
3. $E \rightarrow (E)$
4. $E \rightarrow \mathbf{id}$



Recuperación de errores (4)

Ejemplo para expresiones

- Primero, las entradas erróneas para estados que llevan a cabo una reducción se sustituyen por dicha reducción.
  - Este cambio puede posponer la detección del error y permitir reducciones posteriores, pero el error se detectará antes de que tenga lugar ningún desplazamiento.
- Las restantes entradas en blanco se sustituyen por llamadas a las siguientes rutinas de error.

  ***e*1**: inserta un token **id** en la pila y lo cubre con el estado 3 (el estado de destino para los estado 0, 2, 4 y 5 con **id**).
  - se invoca desde los estados 0, 2, 4 y 5 que esperan el principio de un operando, ya sea un identificador o un paréntesis de apertura, en vez de eso se encuentran un operador o el fin de la entrada.
  - Mensaje emitido: «*operando ausente*»

Recuperación de errores (5)

Ejemplo para expresiones

***e*2**: elimina el paréntesis derecho de la entrada

- Se invoca desde los estados 0, 1, 2, 4 y 5 cuando encuentra un paréntesis derecho.
- Mensaje emitido «*paréntesis derecho desemparejado*»

***e*3**: introduce el *token* + en la pila y lo cubre con el estado 4

- Se invoca desde los estados 1 o 6 que esperan un operador y en vez de eso se encuentra el *token* **id** o un paréntesis derecho.
- Mensaje emitido: «*operador ausente*»

***e*4**: introduce un paréntesis derecho en la pila y lo cubre con el estado 9

- Se invoca desde el estado 6 cuando se encuentra el final de la entrada, este estado estaba esperando un paréntesis derecho.
- Mensaje emitido: «*paréntesis derecho omitido*»

Recuperación de errores (6)

Ejemplo para expresiones

	id	+	*	(	)	$	E
0	*d*3	**e1**	**e1**	*d*2	**e2**	**e1**	1
1	**e3**	*d*4	*d*5	**e3**	**e2**	*acc*	
2	*d*3	**e1**	**e1**	*d*2	**e2**	**e1**	6
3	***r4***	*r*4	*r*4	***r4***	*r*4	*r*4	
4	*d*3	**e1**	**e1**	*d*2	**e2**	**e1**	7
5	*d*3	**e1**	**e1**	*d*2	**e2**	**e1**	8
6	**e3**	*d*4	*d*5	**e3**	*d*9	**e4**	
7	***r1***	***r1***	***d5***	***r1***	*r*1	*r*1	
8	***r2***	***r2***	***r2***	***r2***	*r*2	*r*2	
9	***r3***	*r*3	*r*3	***r3***	*r*3	*r*3	

1. $E \rightarrow E+E$
2. $E \rightarrow E*E$
3. $E \rightarrow (E)$
4. $E \rightarrow$ **id**

Veamos que pasa con la siguiente entrada errónea: **id +) $**

Recuperación de errores (7)

Ejemplo para expresiones

PILA	ENTRADA	MENSAJE DE ERROR Y ACCIONES
0	id +) $	
0id3	+) $	
0*E*1	+) $	
0*E*1+4	) $	
0*E*1+4	$	«paréntesis derecho desemparejado»
		*e*2 elimina el paréntesis derecho
0*E*1+4id3	$	«operando ausente»
		*e*1 mete en la pila el token id y el estado 3
0*E*1+4*E*7	$	
0*E*1	$	

¿LL(1) o LALR(1)? (1)

- Simplicidad:
  - Ambos métodos son simples, pero LL(1) es más intuitivo.
  - Es más fácil escribir una gramática que sea LALR(1).
  - La mayoría de las construcciones habituales son LL(1).
- Generalidad: toda gramática LL(1) es LALR(1).
- Tratamiento de «símbolos de acción» (para el tratamiento semántico):
  - LL(1) permite ponerlos en cualquier parte de las reglas.
  - LALR(1) sólo al final.
    - Hay «apaños» (pero pueden generar conflictos).
- Recuperación de errores:
  - LL(1) tiene en la pila lo que deseamos encontrar.
  - LALR(1) contiene lo que hemos encontrado.

¿LL(1) o LALR(1)? (2)

- Tamaño del analizador: se ha calculado que, para los lenguajes habituales,
  - $|\text{LALR}(1)| = 2 \times |\text{LL}(1)|$
- **Conclusión**:
  - LL(1) parece tener más ventajas, si la gramática se puede hacer fácilmente LL(1).
  - Hay que manejar bien las dos técnicas.
  - LL(1) es una mejor primera aproximación.
  - Todo depende de la disponibilidad de buenas herramientas
    - LL(*k*): JavaCC, ANTLR, Grammatica, ….
    - LALR(1): yacc, bison, beaver, CUP, YooParse, LRgen, ….