Compiladores e Intérpretes
Análisis Sintáctico III

Sebastian Gottifredi

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

Repaso

- Para expresar las reglas de sintaxis del lenguaje utilizamos **gramáticas libres de contexto**
- El **analizador sintáctico** es el encargado de **reconocer** si un programa sigue esas reglas, para eso:
  - La **gramática** tiene que ser **no ambigua**
  - **Simula** el proceso de **derivación** usando una estrategia:
    - **Descendente** (Top-Down)
    - **Ascendente** (Bottom-Up)

Repaso

- **Descendentes:** arrancando del no terminal Inicial reconstruir la derivación a izquierda hasta llegar a la cadena
- En vez de llegar a la cadena, **arrancamos del NT inicial** y vamos a ir **consumiendo** los **tokens actuales** del Léxico cuando corresponda y por lo tanto **reconoceremos** la cadena cuando lleguemos a la **forma vacía**
- Estrategias Descendentes: simular la derivación de
  - Mediante Tabla LL(1)
  - Recursiva

Intuición Análisis Sintáctico Ascendente

Intuición Análisis Sintáctico Ascendente

- Vamos **desde la cadena** a **hasta el símbolo inicial** de la gramática.
- En estos métodos **no necesitamos eliminar la recursión a izquierda ni factorizar** la gramática original
- **Extendemos** la gramática con el símbolo **$ de fin de archivo** como hacíamos para las LL(1)

$$S \rightarrow E\$$$
$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow (E) \mid id \mid num$$

Intuición Análisis Sintáctico Ascendente

- **Aplicamos** las producciones en **orden inverso**! (**Reducción**)
- La **entrada** la procesamos de **izquierda a derecha**

```
(v1)+5$
```

$S \rightarrow E\$$
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow (E) \mid id \mid num$

$(id)+num\$ \rightarrow (T)+num\$ \rightarrow (E)+num\$ \rightarrow T+num\$ \rightarrow E+num\$ \rightarrow E+T\$ \rightarrow E\$ \rightarrow S$

- Es la **derivación a derecha** vista al revés!

$S \Rightarrow E\$ \Rightarrow E+T\$ \Rightarrow E+num\$ \Rightarrow T+num\$ \Rightarrow (E)+num\$ \Rightarrow (T)+num\$ \Rightarrow (id)+num\$$

Intuición Análisis Sintáctico Ascendente

- El análisis sintáctico ascendente **procesa la cadena de izquierda a derecha** simulando la **derivación a derecha** de forma inversa.
- Estos analizadores sintácticos son llamados **LR(k)**
  - **L**: El programa se procesa de izquierda a derecha (**L**eft scannig)
  - **R**: derivación a derecha (**R**ightmost derivation)
  - **k**: cantidad de símbolos de predicción

Intuición Análisis Sintáctico Ascendente

- **Simular así la derivación** tiene una **implicancia importante** para el desarrollo del analizador sintáctico ascendente:
  - Si $\alpha\beta\delta$ es una forma sentencial
  - podemos reducir $Z \rightarrow \beta$
  - Entonces δ es una secuencia de terminales

¿Por qué vale esto?

Por que $\alpha Z\delta \Rightarrow \alpha\beta\delta$ es la derivación a derecha

- Esto nos permite **dividir la cadena de entrada** en dos partes

Intuición Análisis Sintáctico Ascendente

- La **parte futura** es una **secuencia de terminales** por analizar
  - Es decir, la **parte del programa** que nos queda **por analizar**

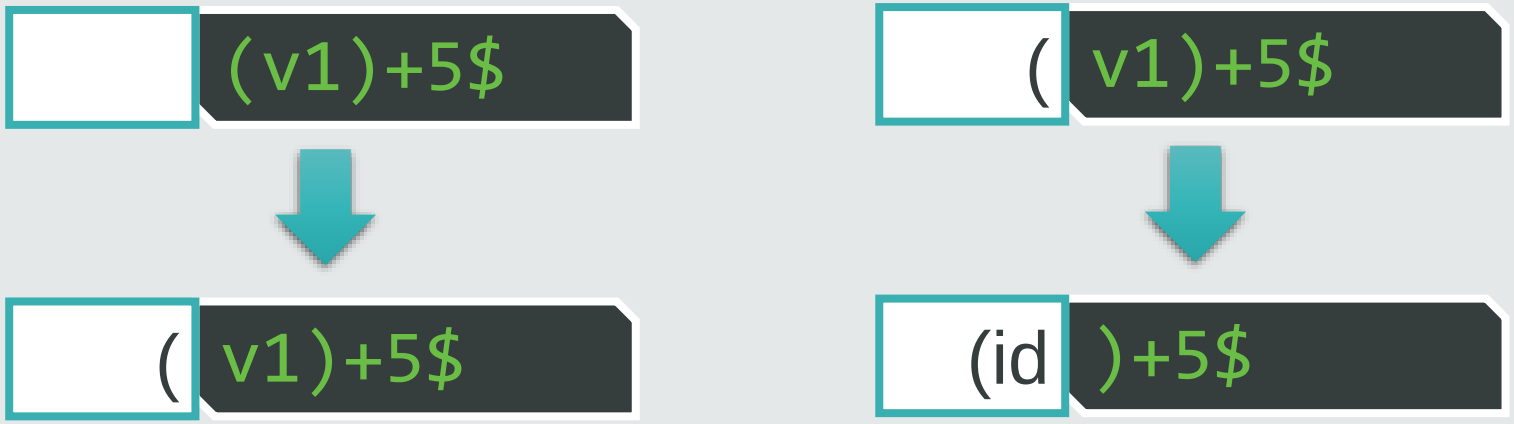
- La **parte actual** de **terminales y no terminales** con la que hemos ido **reduciendo y/o desplazando** para reducir

- Dos **acciones** esenciales
  - **Desplazamiento**
  - **Reducción**

Implementación de Métodos Ascendentes LR

Desplazamiento

- Cuando **no podemos** aplicar ninguna reducción **desplazamos**
- **Movemos** un **terminal** de la parte **futura a la actual** (le pedimos el token al léxico)



S → E$
E → E + T | E - T | T
T → (E) | id | num

Reducción

- Analizando la **parte actual aplicamos** una **producción** en orden **inverso**
- La porción de la **parte actual** que se **reduce** se denomina **handle**

S → E$
E → E + T | E - T | T
T → (E) | id | num

Desplazamientos y Reducciones

- El método LR **desplaza tokens** hasta armar un **handle** con que aplica una **reducción**
- ¿Cómo **administramos** la **parte futura** de la cadena?
  - Con el **Analizador Léxico**
- ¿Cómo **modelamos** la **parte actual**?
  - Una **Pila**!

Desplazamientos y Reducciones

- Con la **Pila**…
- ¿Cómo modelamos el **desplazamiento**?
  - Apilamos el token!
- ¿Cómo modelamos la **reducción**?
  - **Desapilamos 0 o mas símbolos** que se corresponden con la **parte derecha de una producción** y apilamos el símbolo de la izquierda de esa producción

S → E$
E → E + T | | E - T | T
T → (E) | id | num

Handle

- Una de las **clave** del análisis sintáctico LR es decidir **cuando reducir y cuando desplazar**
- La intuición general es que:
  - mientras que **no tengamos un handle** (ie la parte derecha de una producción adecuada en la pila) **desplazamos**
  - Cuando tenemos un handle **reducimos**
- Para tomar esta decisión tenemos que **ver como armamos los handles** a medida que vamos desplazando
  - Considerando **todas las posibles producciones** para las que podríamos estar armando un handle!

Handle

- La **pila** siempre tiene **pedazos de la parte izquierda** de la producción
  - Cuando la tiene **completa** tenemos el **handle**!

- Lo importante es que estos pedazos son siempre **prefijos** de las partes derechas!

S → E$
E → E + T | E - T | T
T → (E) | id | num

Handle

- Entonces, tenemos que:
- llevar nota del **conjunto de producciones** para las que tenemos un **posible prefijo**
- Llevar algún **indicador de donde** en esas **producciones estamos**
- Para esto vamos a **anotar las producciones** e ir llevando un **estado** con todas las **producciones anotadas** que se **corresponden al prefijo**

Conjuntos de Ítems

- Las **producciones anotadas** son denominadas **Ítems**
- Un **Item** es una producción con un • en alguna parte de su lado derecho
- Para la producción E → E + T los siguientes son ítems validos

$$E \rightarrow \bullet\, E + T$$
$$E \rightarrow E \bullet + T$$
$$E \rightarrow E + \bullet\, T$$
$$E \rightarrow E + T \bullet$$

Intuitivamente: el • es una marca que usamos para identificar cuanto llevamos armado del prefijo.

Cuando el • esta al final tenemos un Handle!

Conjuntos de Ítems

- Tenemos que llevar cuenta de **todas** las **producciones** para las que tenemos un **posible prefijo**
- Para eso vamos a llevar un **conjunto con todos los ítems**…
- ¿Qué pasa cuando un ítem tiene • delante de un **NT**?
  - Tenemos **prefijos** para todas las **producciones que comienzan** con **ese no terminal**!
- Por ejemplo
  - si en el conjunto tenemos E → • T deberíamos tener

T → • (E)
T → • id
T → • num

Conjuntos de Ítems

- Partiendo de un conjunto de **Items dado** su **clausura** contendrá **todos los ítems** de la forma X → • α, si en la clausura hay un ítem de la forma Z → β•Xδ (donde β y δ pueden no estar)
  - Partiendo del/los Item/s dado/s agregamos ítems hasta que no podamos agregar mas
- Por ejemplo partiendo de E → • E + T

S → E$
E → E + T | E - T | T
T → (E) | id | num

E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

Conjuntos de Ítems

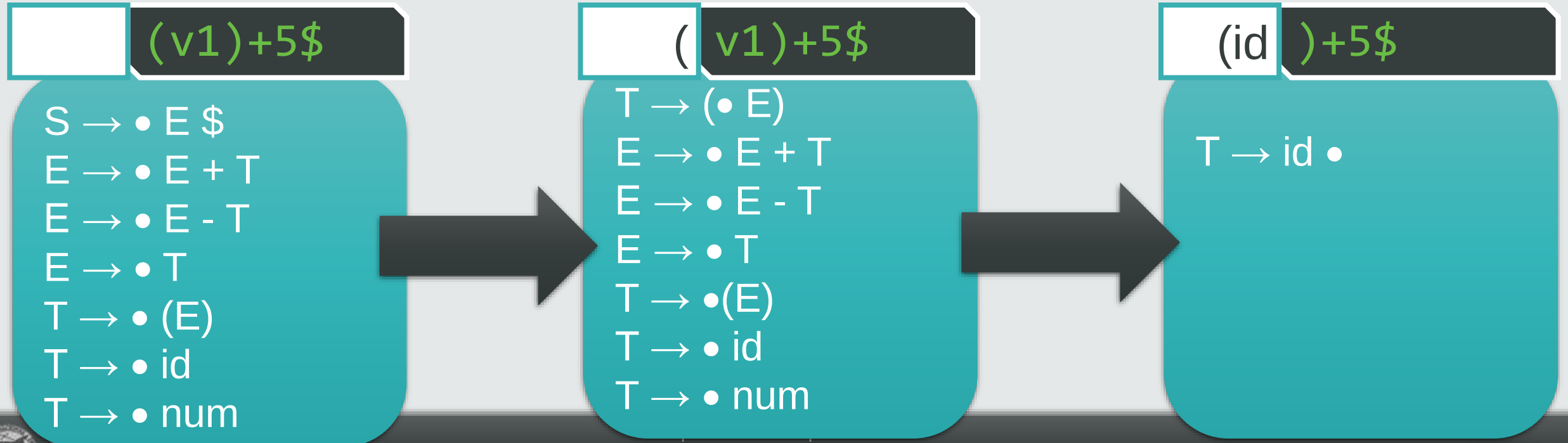
- En los **analizadores sintácticos LR** los **estados** se representan con conjuntos de **ítems clausurados**
- La intuición es que un **estado** representa los **posible prefijos** que estamos armando
- ¿Como sería el estado inicial e0?

S → E$
E → E + T | E - T | T
T → (E) | id | num

S → • E $
E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

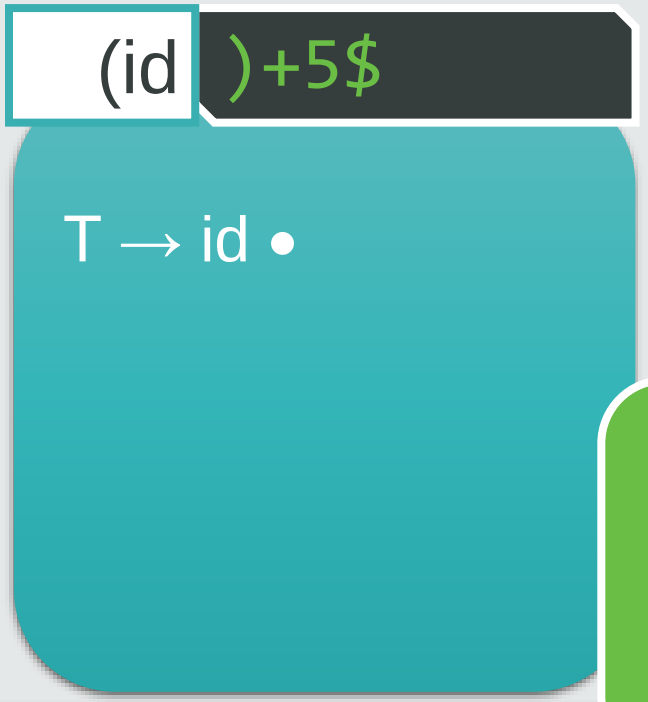
Estados LR

- Cuando **desplazamos** vamos a pasar de un **estado** a donde los **ítems** sean los **resultantes mover el •** en los ítems del **estado originario** según el **token apilamos**



Estados LR

- Cuando llegamos a un estado donde hay un ítem que **tiene el •** **al final podemos reducir**!



¿A que estado vamos?

Tenemos que volver a un punto atrás en el tiempo donde nos desplazamos por id…

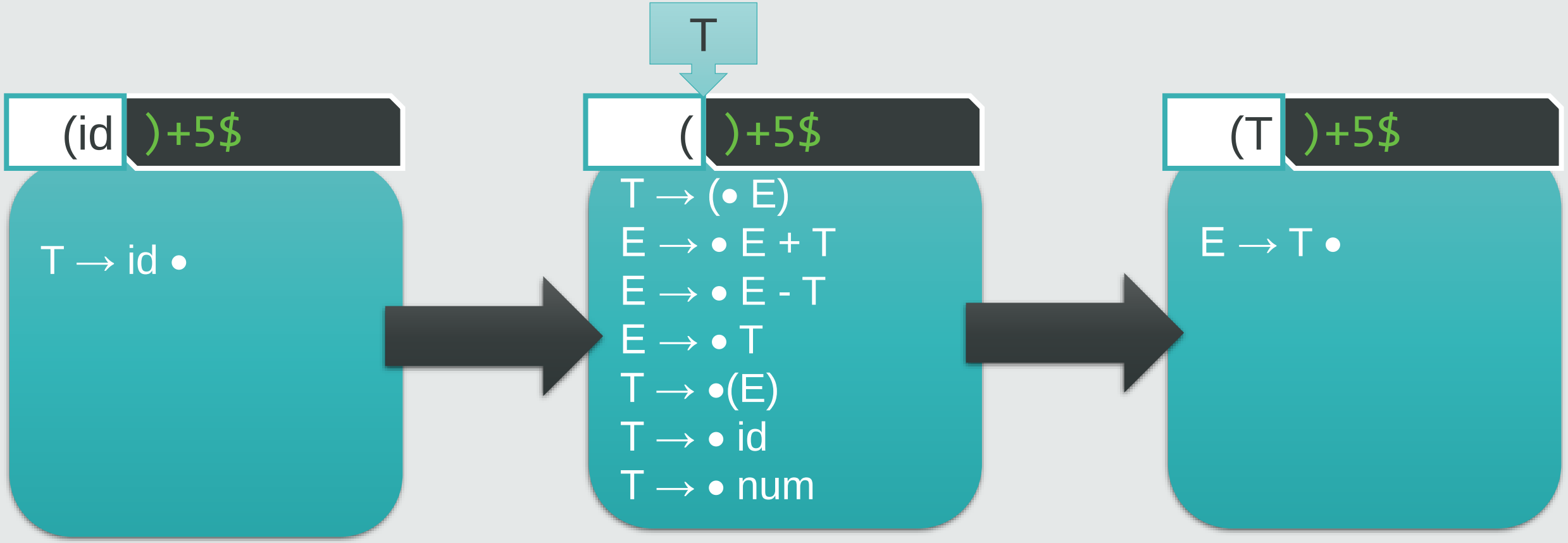
T → (• E)
E → • E + T
E → • E - T
E → • T
T → •(E)
T → • id
T → • num

Pero además, ya desplazamos por T…

Estados LR

- Cuando **reducimos apilamos el NT** de la izquierda de la producción
- Básicamente lo que estamos haciendo es **desplazando ese NT** en la pila…
- Entonces cuando **reducimos** estamos haciendo **dos acciones**
  - La **reducción** en si
  - El **desplazamiento por el NT**

Estados LR



Estados LR

- Dado que cuando **reducimos** tenemos que volver a **estados anteriores**

- ¿Cómo hacemos para **saber** a que **estado volver**?

Apilamos los estados también!

Estados LR

e_0

S → • E $
E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

e_1

T → (• E)
E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

e_2

T → id •

e_3

E → T •

e_0	(v1)+5$
e_0 (	v1)+5$
e_0 (e_1	v1)+5$
e_0 (e_1 id	)+5$
e_0 (e_1 id e_2	)+5$
e_0 (e_1 T	)+5$
e_0 (e_1 T e_3	)+5$
e_0 (e_1 E	)+5$

Tabla LR

- Según el **estado** y el **símbolo desplazado** se toman decisiones…
- ¿Cómo podemos **almacenar esta información**?
- Una **tabla**!
  - **Columnas T o NT desplazado**
  - **Filas estado actual**
  - Las **entradas** pueden indicar:
    - a que **estado** nos **desplazamos**,
    - si en ese estado hay que **reducir** y con que **producción lo hacemos**
    - Si hay que **aceptar** la cadena (el programa es valido!)

Tabla LR

- Por ejemplo, la **porción de la tabla** considerando solo los estados que veníamos viendo en el ejemplo:

e_0

S → • E $
E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

e_1

T → (• E)
E → • E + T
E → • E - T
E → • T
T → • (E)
T → • id
T → • num

e_2

T → id •

e_3

E → T •

	(	)	id	num	+	$	S	E	T
e_0	$D(e_1)$		$D(e_2)$						$D(e_3)$
e_1			$D(e_2)$						$D(e_3)$
e_2	R(T → id)								
e_3	R(E → T)								

No esta completa!
Tenemos que calcular todos los estados y sus transiciones

Algoritmo LR

- **Algoritmo LR** basado en tabla:

```
pila.apilar(e0)
tkActual = Lexico.nextToken()
while(true)
        if(tabla[tkActual, pila.tope()] es D(ex))
                pila.apilar(tkActual)
                pila.apilar(ex)
                tkActual = Lexico.nextToken()
        else if(tabla[tkActual, pila.tope()] es R(X → β))
                for(desde 0 hasta (tamaño de β)*2)
                        pila.desapilar()
                estadoTope = pila.tope()
                pila.apilar(X)
                pila.apilar(tabla[X, estadoTope])
        else if(tabla[tkActual, pila.tope()] es aceptar)
                return Aceptar!
        else
                ERROR!! (ver el detalle en la entrada)
```

Tipos de Analizadores Sintácticos LR basados en Tabla

Tabla LR(0)

- Para **armar la tabla** necesitamos calcular **todos los estados** y sus **transiciones** (los desplazamientos)
- Para esto **partimos del estado inicial**
  - Se construye **clausurando** a partir del **ítem con el • al principio de la producción inicial** del la gramática
- Tomo un estado $\mathbf{e_i}$ **clausurado** y para cada conjunto de ítems de la forma $X \rightarrow \alpha \bullet \beta\ \delta$ (donde α y δ pueden no estar y β es un T o NT)
  - Genero **una transición por** β de $\mathbf{e_i}$ a un estado $\mathbf{e_j}$ **clausurado** a partir de los ítems $X \rightarrow \alpha\ \beta \bullet \delta$ (si **no existe el estado lo creo** y lo **clausuro**)
- Hago esto hasta que **no pueda crear mas estados nuevos**

$S \rightarrow E\$$

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow (E) \mid id \mid num$

Tabla LR(0)

$S \rightarrow E\$$
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow (E) \mid id \mid num$

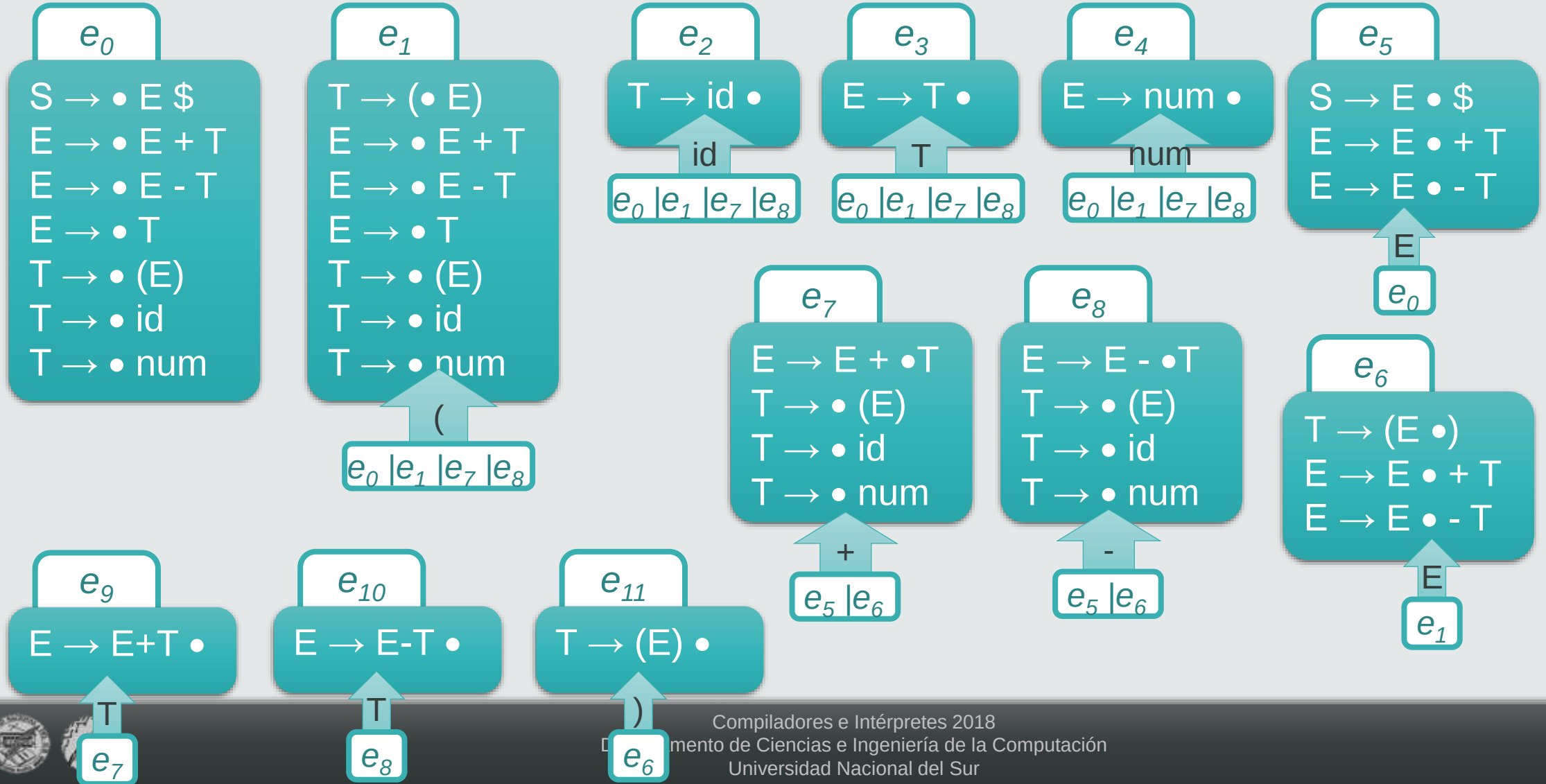


Tabla LR(0)

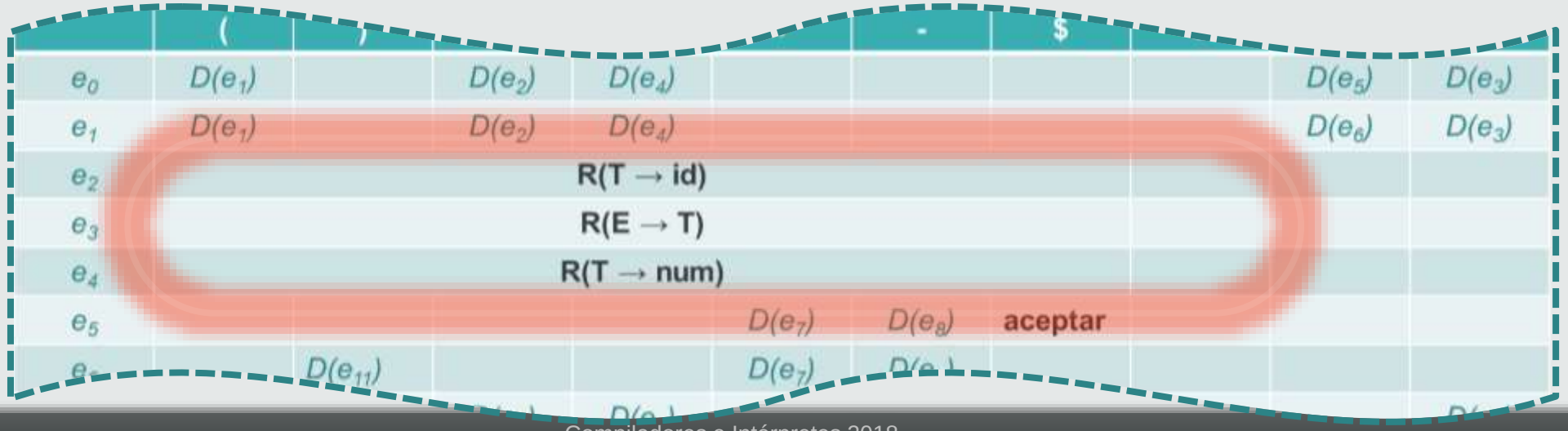
	(	)	id	num	+	-	$	S	E	T
e_0	$D(e_1)$		$D(e_2)$	$D(e_4)$					$D(e_5)$	$D(e_3)$
e_1	$D(e_1)$		$D(e_2)$	$D(e_4)$					$D(e_6)$	$D(e_3)$
e_2	**R(T → id)**									
e_3	**R(E → T)**									
e_4	**R(T → num)**									
e_5					$D(e_7)$	$D(e_8)$	**aceptar**			
e_6		$D(e_{11})$			$D(e_7)$	$D(e_8)$				
e_7	$D(e_1)$		$D(e_2)$	$D(e_4)$						$D(e_9)$
e_8	$D(e_1)$		$D(e_2)$	$D(e_4)$						$D(e_{10})$
e_9	**R(E → E+T)**									
e_{10}	**R(E → E-T)**									
e_{11}	**R(T → (E))**									

Tabla LR(0)

- Al igual que en los **métodos descendentes** si estoy en una situación donde el **algoritmo indexa** una **entrada “vacía”** estamos en presencia de un **error sintáctico**
- En la entrada podemos **guardar el mensaje** adecuado de **error**
- A **diferencia** de los métodos **LL(1)** tenemos **muchas mas entradas**, lo que nos permite **identificar mas contextos**
  - Podemos brindar mensajes de **error mas sofisticados/adecuados**

SLR(0) o LR(0)

- La **estrategia** que utilizamos para **construir la tabla** es denominada **SLR(0) o LR(0)**
- El **0** es por que **no hacemos lookahead** al reducir, es decir **reducimos a ciegas**



SLR(0) o LR(0)

- Este método es **muy eficiente**, pero bastante **limitado**
- Requiere que los **estados** con **ítems** que tienen • **al final** (ítem completo) sean **unitarios**

SLR(0) o LR(0)

- Si nuestra gramática hubiese sido

$S \rightarrow E\$$
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow (E) \mid id \mid id(E) \mid num$

e_2

$T \rightarrow id \bullet$
$T \rightarrow id \bullet (E)$

- En este **estado** en la **tabla** tendríamos **entradas múltiples**:
  - **Reducir** por $T \rightarrow id$
  - **Desplazar** por (

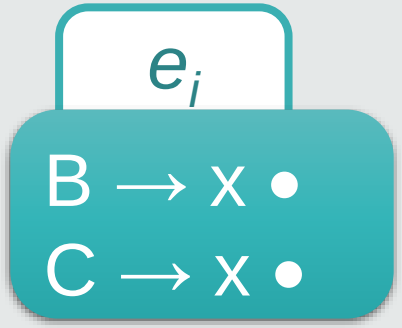
conflicto desplazamiento-reduccion

Esa gramática no es SLR(0)

SLR(0) o LR(0)

- Otro ejemplo de una gramática **no SLR(0)** se cuando hay un conflicto **reducción-reducción**

$S \rightarrow A\$$
$A \rightarrow Bw \mid Cz$
$B \rightarrow x$
$C \rightarrow x$



- En este caso **no sabemos si reducir** por $B \rightarrow x$ o por $C \rightarrow x$

SLR(1)

- El **método SLR(1)** es el siguiente en **capacidad de reconocimiento**
- Básicamente **utiliza Lookahead** para **decidir** cuando si es adecuado o no aplicar una **reducción**
- Intuitivamente si estamos en un **estado** con un **ítem de la forma X** $\rightarrow \alpha \bullet$ **reducimos** por X **si el lookahead** se corresponde con algún terminal en **Siguientes(X)**
  - En la **tabla** a diferencia del SLR(0) en el SLR(1) **solo vamos** a indicar la **reducción X** en las entradas de los **Siguientes(X)**

SLR(1)

- Hay ciertos **requisitos** para que una **gramática sea SLR(1)**
- Un **estado e_i** con un ítems completo **X** $\rightarrow \alpha \bullet$ debe **cumplirse que:**
  - **No haya en e_i un ítem** de la forma Z $\rightarrow \alpha \bullet \beta$ tal que hay algún elemento en **Primeros(β)** que este en **Siguiente(X)**

SLR(1)

- Hay ciertos **requisitos** para que una **gramática sea SLR(1)**
- Un **estado e_i** con un ítems completo **X** $\rightarrow \alpha \bullet$ debe **cumplirse que**:
  - **No haya en e_i un ítem** de la forma Z $\rightarrow \alpha \bullet \beta$ tal que hay algún elemento en **Primeros(β)** que este en **Siguiente(X)**
  - **No haya en e_i un ítem** de la forma Y $\rightarrow \alpha \bullet$ tal que hay algún elemento en **Siguiente(Y)** que este en **Siguiente(X)**

SLR(1)

S → E$
E → E + T | E - T | T
T → (E) | id | id(E) | num

e_2

T → id •
T → id • (E)

Siguientes(T) = {$,)}
Primeros((E)) = {(}

S → A$
A → Bw | Cz
B → x
C → x

e_i

B → x •
C → x •

Siguientes(B) = {w}
Siguientes(C) = {z}

SLR(1)

- Con las gramáticas SLR podemos capturar la mayoría de los constructores de en Lenguajes de Programación
- Pero algunos no…

S → D$
D → L T
L → id | id L
T → id | type

e_i

L → id • L
L → id •

Primeros(L) = {id}
Siguientes(L) = {id,type}

Conflicto desplazamiento-reduccion

LR(1)

- Los métodos **LR(1)** son **mas poderosos** que los **SLR(1)**
- Mirar **todos los siguientes** de un no terminal para decidir si aplicar una reducción, puede ser que lleve a **conflictos**
- Los **LR(1)** mejoran **mirando solo los siguientes locales** a un estado

LR(1)

- Para esto, los **ítems** en el **método LR(1)** se construyen **indicando** además la producción con • el simbolo de **lookahead** local lh.
  - Por ejemplo [X → α • b δ , lh]
- El **lookahead local** se **actualiza** en los ítems resultantes de **clausurar un estado**, de la siguiente manera
  - Al **clausurar** por un ítem [Z → α•Cβ , lh] y teniendo la producción C→ δ
  - **Generamos** un ítem [C→ • δ, t] por cada t en Primeros(βlh)

LR(1)

- Los **símbolos de lookahead** son los son utilizados en un **estado con ítems completos** para:
  - decidir **cuando** hacer una **reducción** y
  - **controlar** si tenemos **conflictos** de **desplazamiento-reducción** o **reducción-reducción**, es decir si tenemos $[Y \rightarrow \delta \bullet, t]$ en el estado controlar que:
    - No haya un ítem $[V \rightarrow \delta \bullet \gamma, t]$ con t en **Primeros(γ)**
    - No haya un ítem $[W \rightarrow \delta \bullet, t]$

LALR(1)

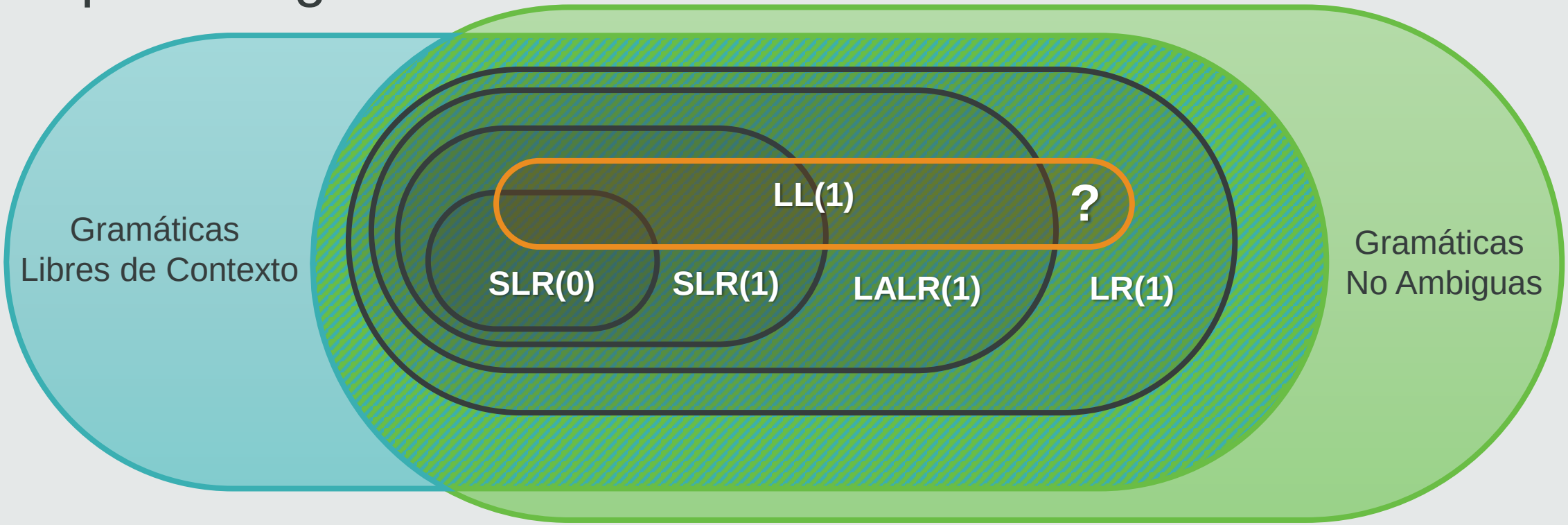
- Los analizadores **LR(1) son los mas poderosos** dentro de los métodos LR con lookahead 1
- El costo es que **tienen muchos mas estados** que los **SLR(1)**, por lo tanto sus **tablas son mucho mas grandes**
- Una **optimización** esta dada por **los métodos LALR(1)**
  - Son **menos poderosos** que los **LR(1), mas** que los **SLR(1)** pero tienen **muchos menos estados** que los **LR(1)**
  - Se **construyen fusionando estados LR(1)** que comparten ítems de igual corazón (producción con •)

Métodos y Gramáticas LR

- La **estrategia general** de los **Anlizadores Sintacticos LR** es la básicamente misma (el algoritmo que vimos antes)
- Para **poder determinar** que **acción** seguir la estrategia de los métodos LR se basan en la Tabla.
- Cada **método LR** tiene una forma distinta de **construir la tabla**
- De **mas simple** a **mas sofisticada**
- SLR(0) o LR(0) → SLR(1) → LALR(1) → LR(1)

Métodos y Gramáticas LR

- **Métodos LR** y las **gramáticas libres de contexto** para las que pueden generar **tablas sin conflictos**



Métodos y Gramáticas LR

- Si para un **método LR** una gramática es tal que en **un estado** de la pila **no podemos decir que hacer**, ya sea por que:
  - Es factible tanto reducir como desplazar (conflicto **desplazamiento-reducción**)
  - Es factible reducir por mas de una producción (conflicto **reducción-reducción**
- Esto va a terminar en una **tabla con entradas múltiples**, y
- Decimos que esa gramática **NO es de ese tipo LR**
- **Posiblemente** sea necesario utilizar **un método LR mas sofisticado** capaz de decidir en esos casos.

Métodos y Gramáticas LR

- Los **métodos LR** son los **mas adecuados** para **reportar errores**
- **Mantienen** mejor la **estructura de la gramática** original respecto a los LL(1)
- Son **eficientes**, pero sus **tablas** suelen ser **muy grandes**
- En los **lenguajes comerciales** la construcción de las **tablas a mano es complejo**
  - Se utilizan **herramientas automatizadas**
    - se especifica la **gramática** y la **herramienta construye automáticamente** el analizador sintactico