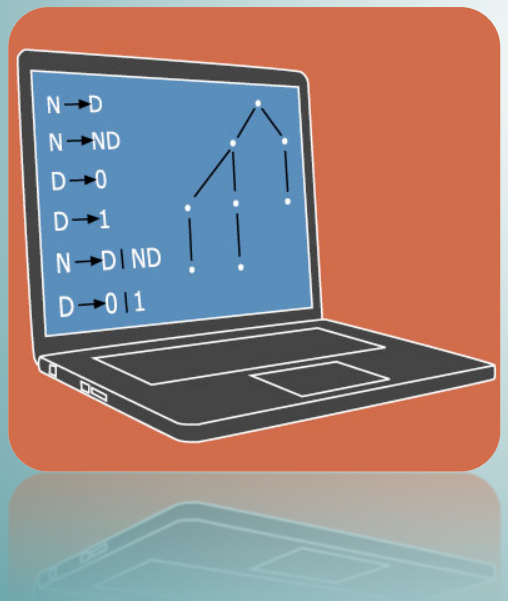
Procesadores de Lenguaje

Analizadores LALR



Cristina Tirnauca

Domingo Gómez Pérez

Dpto. de Matemáticas,

Estadística y Computación

El Analizador LALR

El método $LR(1)$ es el método más potente que hemos visto para realizar parsing. Ventajas:

- Es un método muy conocido, con muchas optimizaciones.
- Los métodos $LR(k)$ se pueden reducir algoritmicamente a $LR(1)$.
- Se puede combinar con el análisis semántico y proporciona un recorrido en postorden del árbol de derivacion.

Inconvenientes::

- Aparecen muchos estados en el autómata $LR(1)$.

El Analizador LALR

Bison y Yacc y varios otros

A base de ítems LR(1).

Pero empleando estados LR(0).

- Si lo miramos desde el LR canónico:
  - Construyes el analizador LR canónico,
  - y fundes entre sí los estados que sólo difieran en los *look-ahead*.
  - O lo que es lo mismo, fundes entre sí los estados cuya parte LR(0) coincida.
  - Obtienes estados LR(0), pero con ítems LR(1) que traen los *look-ahead* correspondientes.

El Analizador LALR

Bison y Yacc y varios otros

A base de ítems LR(1).

Pero empleando estados LR(0).

- Si lo miramos desde el LR canónico:
  - Construyes el analizador LR canónico,
  - y fundes entre sí los estados que sólo difieran en los *look-ahead*.
  - O lo que es lo mismo, fundes entre sí los estados cuya parte LR(0) coincida.
  - Obtienes estados LR(0), pero con ítems LR(1) que traen los *look-ahead* correspondientes.
- Si lo miramos desde el LR(0):
  - Construyes el analizador LR(0),
  - y amplías cada ítem LR(0) en cada uno de los estados con el *look-ahead* que obtendrías si hubieras usado ítems LR(1).

El Analizador LALR

Bison y Yacc y varios otros

A base de ítems LR(1).

Pero empleando estados LR(0).

- Si lo miramos desde el LR canónico:
  - Construyes el analizador LR canónico,
  - y fundes entre sí los estados que sólo difieran en los *look-ahead*.
  - O lo que es lo mismo, fundes entre sí los estados cuya parte LR(0) coincida.
  - Obtienes estados LR(0), pero con ítems LR(1) que traen los *look-ahead* correspondientes.
- Si lo miramos desde el LR(0):
  - Construyes el analizador LR(0),
  - y amplías cada ítem LR(0) en cada uno de los estados con el *look-ahead* que obtendrías si hubieras usado ítems LR(1).
  - (Esta última tarea no es nada trivial.)

LR canónico y LALR

La construcción "laboriosa" de LALR

Un clásico:

Dos trenes; cada uno de ellos es una secuencia no acotada (quizá vacía) de vagones, tirados por una locomotora.

```
%token VAG LOC
%%
S : dostrenes $
dostrenes : tren tren
tren : VAG tren | LOC
```

- En realidad es un lenguaje regular, y el analizador LR(0) puede con él.
- Pero es un buen ejemplo de cómo se construye el analizador LALR a partir del LR canónico.

Tabla de Análisis LALR

El rol de los estados al construir el analizador

Precisamos los estados únicamente para construir la tabla de análisis.

Cada entrada de la tabla:

Corresponde a un estado E y a un *token* que viene a continuación.

- Indica `desplazar` si hay un ítem en el estado que lleva el punto justo delante del mismo *token*.
- Indica `reducir` si hay un ítem en el estado que lleva el punto justo al final, y que incluye ese mismo *token* en su *look-ahead*.

LR canónico tiene tablas más grandes. LR(0), SLR y LALR usan los mismos estados y sólo se diferencian en qué entradas de la tabla "mandan reducir".

Núcleos de Estados LR(0)

La esencia que describe el estado

El estado inicial tiene siempre el item $[S' \to \bullet S\$]$ o equivalente. Todo lo demás es consecuencia de éste.

Dos maneras de generar ítems LR(0):

1. A partir de un ítem que ya existe, en el que el punto está frente a un símbolo no terminal, se generan nuevos ítems en el mismo estado, con ese símbolo como parte izquierda, y el punto al comienzo de la parte derecha.
2. A partir de un ítem que ya existe, "el punto avanza" y causa la incorporación del nuevo ítem a un nuevo estado:
   - el punto "avanza" un *token* al desplazar;
   - el punto "avanza" un símbolo no terminal tras una reducción.

Llamamos núcleo del estado a los ítems que se han generado de la manera 2.

Núcleos de Estados LR(0)

La esencia que describe el estado

El estado inicial tiene siempre el item $[S' \to \bullet S\$]$ o equivalente. Todo lo demás es consecuencia de éste.

Dos maneras de generar ítems LR(0):

1. A partir de un ítem que ya existe, en el que el punto está frente a un símbolo no terminal, se generan nuevos ítems en el mismo estado, con ese símbolo como parte izquierda, y el punto al comienzo de la parte derecha.
2. A partir de un ítem que ya existe, "el punto avanza" y causa la incorporación del nuevo ítem a un nuevo estado:
   - el punto "avanza" un *token* al desplazar;
   - el punto "avanza" un símbolo no terminal tras una reducción.

Llamamos núcleo del estado a los ítems que se han generado de la manera 2. Ojo: el ítem causante puede no ser un ítem del núcleo del estado de partida.

Fuentes de *Look-ahead*, I

Pueden provenir de dos vías

Dos maneras de transformar ítems LR(0) en ítems LR(1):

1. *look-aheads* que se generan y
2. *look-aheads* que se propagan.

Idea inicial: se generan "dentro" del mismo estado al incluír ítems que no son núcleo; se propagan al causar ítems núcleo en otros estados.

Fuentes de *Look-ahead*, I

Pueden provenir de dos vías

Dos maneras de transformar ítems LR(0) en ítems LR(1):

1. *look-aheads* que se generan y
2. *look-aheads* que se propagan.

Idea inicial: se generan "dentro" del mismo estado al incluír ítems que no son núcleo; se propagan al causar ítems núcleo en otros estados.
Pero esta idea no se emplea así. Se gestiona evitando poner en juego los ítems que no son núcleo.

Fuentes de *Look-ahead*, II

Pueden provenir de dos vías

Evitaremos calcular *look-aheads* fuera del núcleo.
Tendremos en cuenta los ítems que no son del núcleo únicamente cuando sea imprescindible.

Redefinimos las nociones para usar únicamente ítems núcleo:

1. *look-aheads* que se generan "de un núcleo a otro" y
2. *look-aheads* que se propagan "de un núcleo a otro".

La propagación es la misma de antes pero sólo entre núcleos. Una propagación que viene de un ítem no núcleo se obtiene al considerar conjuntamente la generación en ese ítem.

La generación va "a través de un ítem no núcleo" intermedio.

Grafo de Propagación

Preparar la propagación antes de realizarla

Si generamos y propagamos a la vez los *look-ahead*, las cosas se complican.

Comparando ítems podemos ver qué *look-aheads* se generan, pero no cuáles se propagan.

Grafo de Propagación

Preparar la propagación antes de realizarla

Si generamos y propagamos a la vez los *look-ahead*, las cosas se complican.

Comparando ítems podemos ver qué *look-aheads* se generan, pero no cuáles se propagan.

Pero sí vemos "por dónde" se propagan. Por tanto, al recorrer los ítems, anotamos *look-aheads* que se generan y, a la vez, construimos un grafo dirigido que nos indicará cómo se propagan.

Cálculo de los *Look-ahead*

Se descompone en cuatro tareas

Organizamos cuatro tareas en tres fases, porque las dos primeras tareas se realizan a la vez.

Fases:

1. A la vez, sobre todos los ítems núcleo,
   - calcular los *look-aheads* que se generan y
   - construir el grafo de propagación.
2. Propagar por el grafo los *look-aheads* generados.
3. Construir la tabla de análisis, calculando *look-aheads* de ítems no núcleo únicamente cuando son necesarios.

Generación de *Look-aheads*

La primera de las cuatro fases

Empleamos un "falso *look-ahead*" que nos sirve de marcador para construir el grafo de propagación.

Algoritmo:

Para cada ítem núcleo $[A \to \alpha \bullet \beta]$ del estado E:

1. lo convertimos en LR(1) añadiendo el "falso *look-ahead*" (por ejemplo "#");
2. calculamos todos los ítems λ-accesibles desde ese ítem LR(1), $[A \to \alpha \bullet \beta,\ \#]$ (él mismo y λ-transiciones), y
3. para cada nuevo ítem resultante, $[B \to \gamma \bullet X\delta,\ a]$:
   - si $a = \#$, anotamos que los *look-ahead* de $[A \to \alpha \bullet \beta]$ se habrán de propagar a $[B \to \gamma X \bullet \delta]$: ponemos el correspondiente arco en el grafo de propagación;
   - si $a \neq \#$, anotamos a como *look-ahead* generado en el ítem $[B \to \gamma X \bullet \delta]$ del estado transición desde E con X.

Construcción de la Tabla LALR

Proceso similar al caso SLR

Cuatro tareas en tres fases:

- Preproceso de *look-aheads* en un único algoritmo:
  - Cálculo de *look-aheads* generados y
  - Construcción del grafo de propagación de *look-aheads*;
- Propagación de *look-aheads* (cálculo propiamente dicho de los *look-aheads*);
- Construcción de la tabla.

Construcción de la Tabla LALR

Proceso similar al caso SLR

Cuatro tareas en tres fases:

- Preproceso de *look-aheads* en un único algoritmo:
  - Cálculo de *look-aheads* generados y
  - Construcción del grafo de propagación de *look-aheads*;
- Propagación de *look-aheads* (cálculo propiamente dicho de los *look-aheads*);
- Construcción de la tabla.
  - Parte Acción de la tabla (solemos pintarla a la izquierda):
    - a qué estado vamos con los *shift* y
    - qué reglas aplicamos para reducir, en cada estado y con cada *token* de la entrada.
  - Parte GoTo de la tabla: al reducir y apilar un símbolo no terminal, cuál es el nuevo estado.

Ejemplo de Autómata LALR

Lenguaje de Dyck sin palabra vacía

Excluyendo la palabra vacía:

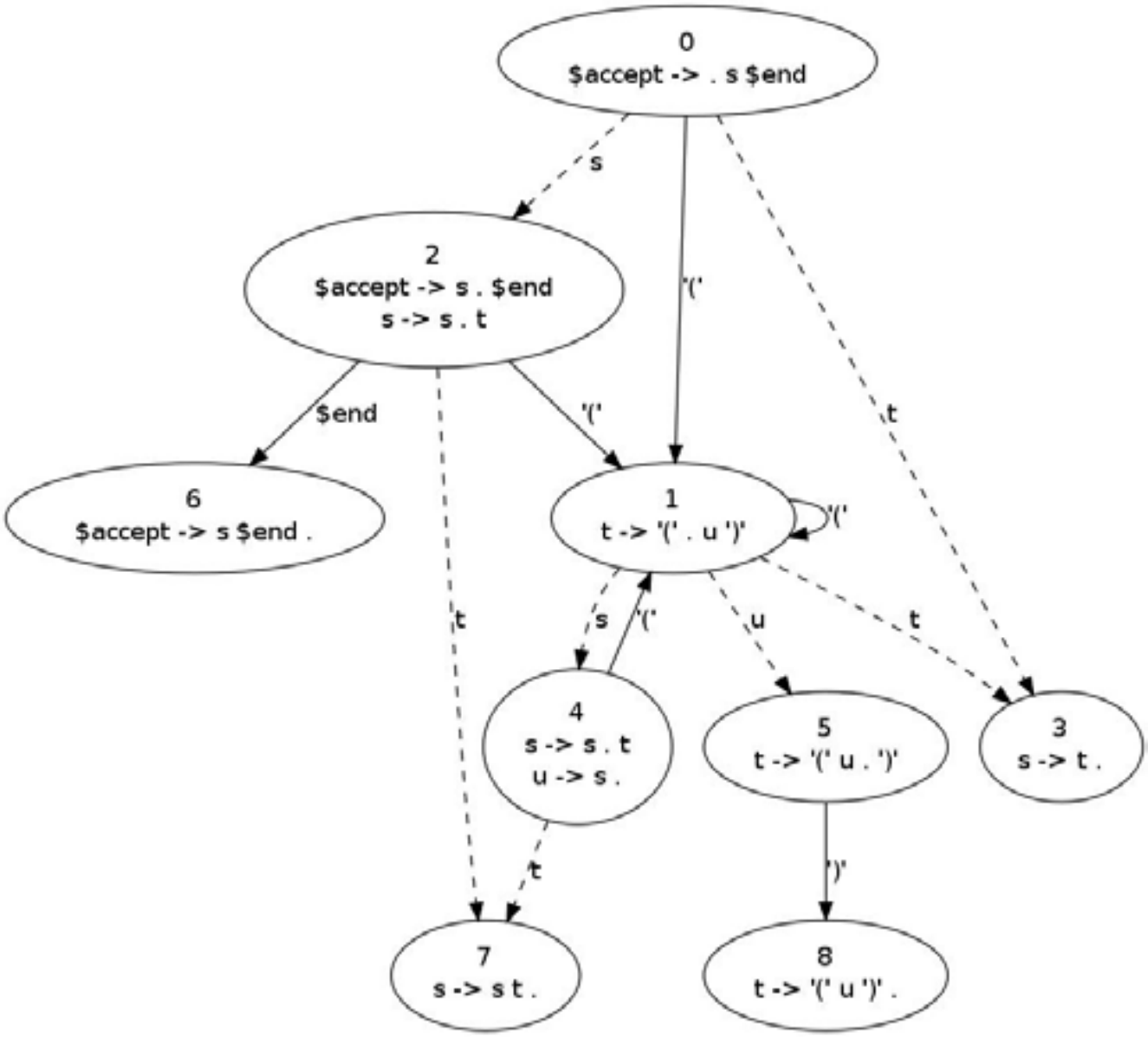
```
%%

s : s t | t;

t : '(' u ')' ;

u : s | /* empty */;
```

Automata LR(0)



Símbolos Propagados

Estados	Estados a donde se propaga	Símbolos generados
0	1,2,3	($ accept->.s$ end,$) (s->.st,(,$)
1	5	
2	1,6,7	
4	1,7	(u->s.$,))
5	8	

Tabla LALR

Tal como la obtenemos de Bison

estado	'('	')'	$	s	t	u
0	shift: 1			2	3	
1	shift: 1	u:λ	u:λ	4	3	5
2	shift: 1		shift: 6		7	
3	s:t	s:t	s:t			
4	shift: 1	u:s	u:s		7	
5		shift: 8				
6	accept	accept	accept			
7	s:s t	s:s t	s:s t			
8	t:'(' u ')'	t:'(' u ')'	t:'(' u ')'			