CI3725 - Traductores e Interpretadores
Abril-Julio 2006
Prof. Ernesto Hernández-Novich

Laboratorio Semana X

FIRST

Sea $G =< N, \Sigma, P, S >$ una gramática libre de contexto. Construiremos el conjunto $FIRST(X)$ donde $X \in N \cup \Sigma$ recursivamente como:

1. $X \in \Sigma \Rightarrow FIRST(X) = \{X\}$
2. $X \to \lambda \in P \Rightarrow FIRST(X) = FIRST(X) \cup \{\lambda\}$
3. Si $X \Rightarrow Y_1 Y_2 \ldots Y_k$ entonces
   a) $(\forall j : 1 \leq j < i \wedge \lambda \in FIRST(Y_j)) \wedge a \in FIRST(Y_i) \Rightarrow FIRST(X) = FIRST(X) \cup \{a\}$
   b) $(\forall j : \lambda \in FIRST(Y_j)) \Rightarrow FIRST(X) = FIRST(X) \cup \{\lambda\}$

FOLLOW

Sea $G =< N, \Sigma, P, S >$ una gramática libre de contexto. Construiremos el conjunto $FOLLOW(X)$ donde $X \in N$ recursivamente como:

1. $\$ \in FOLLOW(S)$[1]
2. $A \to \alpha B \beta \in P \Rightarrow FOLLOW(B) = FOLLOW(B) \cup FIRST(\beta)$ excepto λ.
3. $A \to \alpha B \in P \vee (A \to \alpha B \beta \in P \wedge \lambda \in FIRST(\beta)) \Rightarrow FOLLOW(B) = FOLLOW(B) \cup FOLLOW(A)$

Ejemplo 1

Calculemos $FIRST$ y $FOLLOW$ para todos los símbolos pertinentes de la gramática

$$
\begin{array}{ll}
(1) & E \to TE' \\
(2) & E' \to +TE' \\
(3) & E' \to \lambda \\
(4) & T \to FT' \\
(5) & T' \to *FT' \\
(6) & T' \to \lambda \\
(7) & F \to (E) \\
(8) & F \to \mathbf{id}
\end{array}
$$

Para el cálculo del $FIRST$:

- Por la producción (1) y la regla (3a) de $FIRST \Rightarrow FIRST(E) = FIRST(T)$.
- Por la producción (4) y la regla (3a) de $FIRST \Rightarrow FIRST(T) = FIRST(F)$.
- Por la producción (7) y la regla (3a) de $FIRST \Rightarrow FIRST(F) \supset FIRST(($, y por la regla (1) de $FIRST$ inmediatamente $FIRST(F) \supset \{($.
- Por la producción (8) y la regla (3a) de $FIRST \Rightarrow FIRST(F) \supset FIRST(\mathbf{id})$, y por la regla (1) de $FIRST$ inmediatamente $FIRST(F) \supset \{\mathbf{id}\}$.

[1] El símbolo \$ es un terminal adicional que representa el final de la entrada.

- Por la producción (2) y la regla (3a) de $FIRST \Rightarrow FIRST(E') \supset FIRST(+)$ y por la regla (1) de $FIRST$ inmediatamente $FIRST(E') \supset \{+\}$.
- Por la producción (3) y la regla (2) de $FIRST \Rightarrow FIRST(E') \supset \{\lambda\}$.
- Por la producción (5) y la regla (3a) de $FIRST \Rightarrow FIRST(T') \supset FIRST(*)$ y por la regla (1) de $FIRST$ inmediatamente $FIRST(T') \supset \{*\}$.
- Por la producción (6) y la regla (2) de $FIRST \Rightarrow FIRST(T') \supset \{\lambda\}$.

Todo esto resulta en

$$\begin{aligned} FIRST(E) = FIRST(T) = FRIST(F) &= \{(, \mathbf{id}\} \\ FIRST(E') &= \{+, \lambda\} \\ FIRST(T') &= \{*, \lambda\} \end{aligned}$$

Para el cálculo del $FOLLOW$:

- Por la producción (1) y la regla (1) de $FOLLOW \Rightarrow FOLLOW(E) \supset \{\$\}$
- Por la producción (7) y la regla (2) de $FOLLOW \Rightarrow FOLLOW(E) \supset FIRST())$
- Por la producción (1) y la regla (3) de $FOLLOW \Rightarrow FOLLOW(E') \supset FOLLOW(E) \setminus \{\lambda\}$
- Por la producción (4) y la regla (3) de $FOLLOW \Rightarrow FOLLOW(T) \supset FOLLOW(T') \setminus \{\lambda\}$
- Por la producción (2) y la regla (3) de $FOLLOW \Rightarrow FOLLOW(T) \supset FIRST(E') \setminus \{\lambda\}$
- Por la producción (4) y la regla (2) de $FOLLOW \Rightarrow FOLLOW(F) \supset FIRST(T') \setminus \{\lambda\}$
- Por la producción (5) y la regla (3) de $FOLLOW \Rightarrow FOLLOW(F) \supset FOLLOW(T') \setminus \{\lambda\}$

Todo esto resulta en

$$\begin{aligned} FOLLOW(E) = FOLLOW(E') &= \{\$,)\} \\ FOLLOW(T) = FOLLOW(T') &= \{\$,), +\} \\ FOLLOW(F) &= \{\$,), +, *\} \end{aligned}$$

Construcción de Tablas de Parsing $SLR(1)$

El algoritmo de construcción de tablas de parsing $LR(0)$ que estudiamos la semana pasada, sólo es capaz de generar tablas para gramáticas $LR(0)$ que son muy pocas. El siguiente algoritmo, denominado también *Simple LR Parsing Table Generation* permite procesar más gramáticas. La diferencia principal radica en la forma en que se establecen las acciones *reduce*: en lugar de llenar toda la fila de la tabla de acciones con *reduce*, sólo se ocupan las columnas donde el terminal corresponda al $FOLLOW$ del terminal que se está reduciendo.

El algoritmo de construcción queda como:

1. Aumentar la gramática original con un nuevo símbolo inicial S' tal que agregue la producción $S' \rightarrow S$ donde S es el símbolo inicial original de la gramática.
2. Enumerar contando desde cero cada producción de la gramática aumentada comenzando por la nueva producción inicial.
3. Calcular el autómata de prefijos viables a partir de los items $LR(0)$ de la gramática. Cada conjunto de items I_i servirá para construir el i-ésimo estado de la máquina parser $SLR(1)$.
4. Construir la tabla de *goto* usando una columna por cada símbolo no-terminal de la gramática original y una fila por cada estado. Si en el autómata de prefijos viables hay una transición desde I_i hasta I_j con el símbolo no terminal X entonces $goto[i, X] \leftarrow j$.

5. Construir la tabla de *acciones* usando una columna por cada símbolo terminal de la gramática aumentada incluyendo una para \$ y una fila por cada estado:

   a) Si en el autómata de prefijos viables hay una transición desde I_i hasta I_j con el símbolo terminal **x**, entonces $acciones[i, \mathbf{x}] \leftarrow shift\,\mathbf{j}$.

   b) Si en el autómata de prefijos viables el i-ésimo conjunto contiene un item de la forma $A \rightarrow \alpha\cdot$ y $A \rightarrow \alpha$ es la n-ésima producción de la gramática ($n > 0$), entonces $\forall \mathbf{a} \in FOLLOW(A) \wedge A \neq S' \Rightarrow acciones[i, \mathbf{a}] \leftarrow reduce\,\mathbf{n}$, i.e. llenar con *reduce* solamente las columnas que correspondan al $FOLLOW(A)$.

   c) Si en el autómata de prefijos viables el i-ésimo conjunto contiene un item de la forma $S' \rightarrow S\$\cdot$, entonces $acciones[i, \$] \leftarrow accept$.

   d) El resto de la tabla de *acciones* se llena con *error*.

Si durante la construcción de la tabla se producen conflictos en el llenado de la tabla de *acciones* entonces la gramática **no** es $SLR(1)$ y es imposible utilizar el parser $SLR(1)$ para procesarla.

Ejemplo 2

Construyamos la tabla de parser $SLR(1)$ para la gramática

$$\begin{array}{lcl} E & \rightarrow & E{+}T \\ E & \rightarrow & T \\ T & \rightarrow & T{*}F \\ T & \rightarrow & F \\ F & \rightarrow & (E) \\ F & \rightarrow & \mathbf{id} \end{array}$$

Calculamos

$$\begin{array}{rcl} FOLLOW(E) & = & \{\$, +,)\} \\ FOLLOW(T) = FOLLOW(F) & = & \{\$, +,), *\} \end{array}$$

necesarios para construir la tabla de acciones.

Continuamos por aumentar la gramática con un nuevo símbolo no terminal inicial S. Enseguida enumeramos las producciones según describe el algoritmo, para tener

$$\begin{array}{ll} (0) & S \rightarrow E \\ (1) & E \rightarrow E{+}T \\ (2) & E \rightarrow T \\ (3) & T \rightarrow T{*}F \\ (4) & T \rightarrow F \\ (5) & F \rightarrow (E) \\ (6) & F \rightarrow \mathbf{id} \end{array}$$

Construimos el autómata de prefijos viables calculando los conjuntos de items $LR(0)$ de la gramática, que nos quedan

$$\begin{array}{lcl} I_0 & : & S \rightarrow \cdot E \\ & & E \rightarrow \cdot E{+}T \\ & & E \rightarrow \cdot T \\ & & T \rightarrow \cdot T{*}F \end{array}$$

$$
\begin{array}{rcl}
 & & T \to \cdot F \\
 & & F \to \cdot(E) \\
 & & F \to \cdot \mathbf{id} \\
I_1 & : & S \to E \cdot \\
 & & E \to E \cdot + T \\
I_2 & : & E \to T \cdot \\
 & & T \to T \cdot * F \\
I_3 & : & T \to F \cdot \\
I_4 & : & F \to (\cdot E) \\
 & & E \to \cdot E + T \\
 & & E \to \cdot T \\
 & & T \to \cdot T * F \\
 & & T \to \cdot F \\
 & & F \to \cdot(E) \\
 & & F \to \cdot \mathbf{id} \\
I_5 & : & F \to \mathbf{id} \cdot \\
I_6 & : & E \to E + \cdot T \\
 & & T \to \cdot T * F \\
 & & T \to \cdot F \\
 & & F \to \cdot(E) \\
 & & F \to \cdot \mathbf{id} \\
I_7 & : & T \to T * \cdot F \\
 & & F \to \cdot(E) \\
 & & F \to \cdot \mathbf{id} \\
I_8 & : & F \to (E \cdot) \\
 & & E \to E \cdot + T \\
I_9 & : & E \to E + T \cdot \\
 & & T \to T \cdot * F \\
I_{10} & : & T \to T * F \cdot \\
I_{11} & : & F \to (E) \cdot
\end{array}
$$

siendo el autómata resultante de la forma

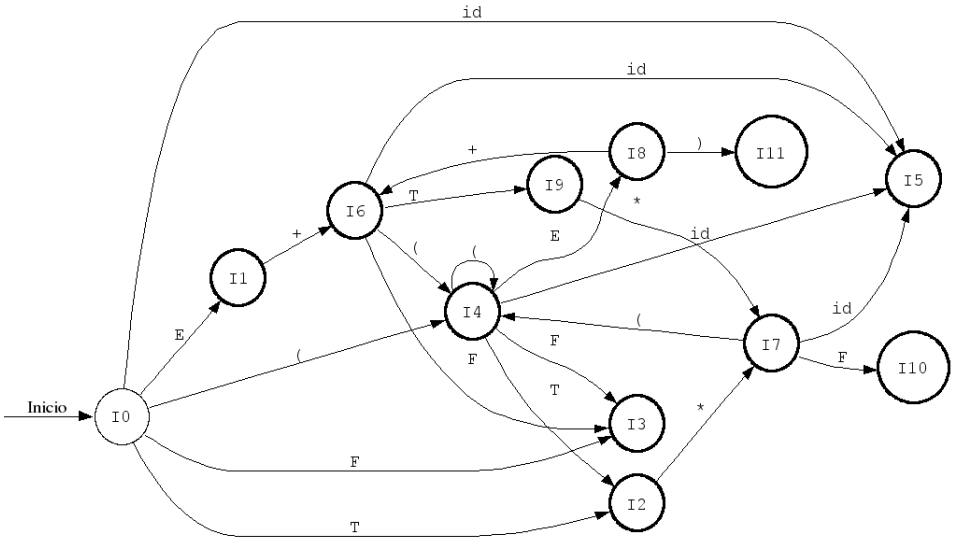

Figura 1: Autómata de Prefijos Viables para el Ejemplo 2

Ahora podemos construir la tabla de parsing $SLR(1)$ según el algoritmo descrito previamente

	+	*	(	)	**id**	$	E	T	F
0			$shift$ **4**		$shift$ **5**		1	2	3
1	$shift$ **6**					*accept*			
2	*reduce* **2**	$shift$ **7**		*reduce* **2**		*reduce* **2**			
3	*reduce* **4**	*reduce* **4**		*reduce* **4**		*reduce* **4**			
4			$shift$ **4**		$shift$ **5**		8	2	3
5	*reduce* **6**	*reduce* **6**		*reduce* **6**		*reduce* **6**			
6			$shift$ **4**		$shift$ **5**			9	3
7			$shift$ **4**		$shift$ **5**				10
8	$shift$ **6**			$shift$ **11**					
9	*reduce* **1**	$shift$ **7**		*reduce* **1**		*reduce* **1**			
10	*reduce* **3**	*reduce* **3**		*reduce* **3**		*reduce* **3**			
11	*reduce* **5**	*reduce* **5**		*reduce* **5**		*reduce* **5**			

Cuadro 1: Tabla de Parsing $SLR(1)$ para el Ejemplo 2

Cálculo de Items $LR(1)$

Consideremos items acompañados de un símbolo de *lookahead* que nos permitirá caracterizar mejor el avance del proceso del parser dependiendo no solamente del estado actual de la máquina, sino del próximo caracter de la entrada **sin** consumirlo.

Ahora los items son de la forma $[A \to \alpha \cdot \beta, \mathbf{a}]$ donde $\mathbf{a} \in \Sigma \cup \{\$\}$ es el *lookahead*. Si bien todos los items contendrán *lookahead* y deben ser tomados en cuenta para el cálculo de la clausura de items, sólo serán relevantes en los items de la forma $[A \to \alpha\cdot, \mathbf{a}]$ para que la acción *reduce* sólo tenga lugar si el siguiente elemento de la entrada es **a**. La presencia del *lookahead* alter la forma en que se calcula la clausura del conjunto de items y el correspondiente autómata de prefijos.

Ahora, para calcular $CLAUSURA(I)$:

1. Consideraremos todos los items $[A \to \alpha \cdot B\beta, \mathbf{a}] \in I$.
2. Para cada producción $B \to \gamma$ y $\forall b \in FIRST(\beta\mathbf{a})$ agregaremos $[B \to \cdot\gamma, \mathbf{b}]$ a $CLAUSURA(I)$.

3. Repetimos los pasos anteriores hasta que no se pueda agregar nada más.

Para hacer más compacta la representación, si en un conjunto de items están presentes los items $[A \to \alpha \cdot X\beta, \mathbf{a}]$ y $[A \to \alpha \cdot X\beta, \mathbf{b}]$, abreviaremos escribiendo $[A \to \alpha \cdot X\beta, \mathbf{a/b}]$. En el mismo orden de ideas, diremos que $A \to \alpha \cdot X\beta$ es el **núcleo** del conjunto de items.

Ejemplo 3

Calculemos el conjunto de items $LR(1)$ y el autómata de prefijos viables para la gramática

$$\begin{array}{lcl} S & \to & CC \\ C & \to & \mathbf{c}C \\ C & \to & \mathbf{d} \end{array}$$

comenzando por aumentar la gramática con un nuevo símbolo inicial S' y la producción $S' \to S$. Continuamos por enumerarlas comenzando desde cero por la producción aumentada, quedando

$$\begin{array}{ll} (0) & S' \to S \\ (1) & S \to CC \\ (2) & C \to \mathbf{c}C \\ (3) & C \to \mathbf{d} \end{array}$$

El cálculo de los items $LR(1)$ comienza por el item $[S' \to \cdot S, \$]$ procediendo con el cálculo de la clausura, con lo cual nos queda

$$\begin{array}{lcl} I_0 & : & [S' \to \cdot S, \$] \\ & & [S \to \cdot CC, \$] \\ & & [C \to \cdot \mathbf{c}C, \mathbf{c/d}] \\ & & [C \to \cdot \mathbf{d}, \mathbf{c/d}] \\ I_1 & : & [S' \to S \cdot, \$] \\ I_2 & : & [S \to C \cdot C, \$] \\ & & [C \to \cdot \mathbf{c}C, \$] \\ & & [C \to \cdot \mathbf{d}, \$] \\ I_3 & : & [C \to \mathbf{c} \cdot C, \mathbf{c/d}] \\ & & [C \to \cdot \mathbf{c}C, \mathbf{c/d}] \\ & & [C \to \cdot \mathbf{d}, \mathbf{c/d}] \\ I_4 & : & [C \to \mathbf{d} \cdot, \mathbf{c/d}] \\ I_5 & : & [S \to CC \cdot, \$] \\ I_6 & : & [C \to \mathbf{c} \cdot C, \$] \\ & & [C \to \cdot \mathbf{c}C, \$] \\ & & [C \to \cdot \mathbf{d}, \$] \\ I_7 & : & [C \to \mathbf{d} \cdot, \$] \\ I_8 & : & [C \to \mathbf{c}C \cdot, \mathbf{c/d}] \\ I_9 & : & [C \to \mathbf{c}C \cdot, \$] \end{array}$$

siendo el autómata resultante de la forma

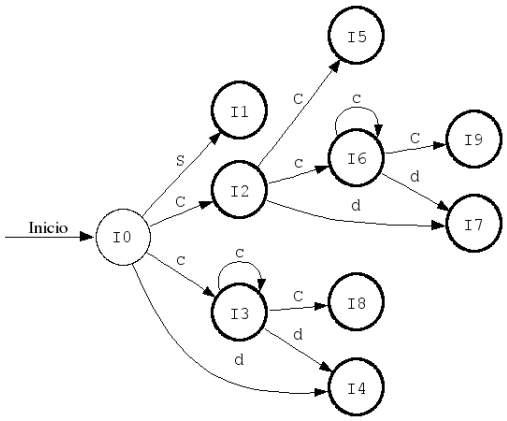

Figura 2: Autómata de Prefijos Viables para el Ejemplo 3

Observemos detalladamente el cálculo de la clausura del item I_0:

- El item inicial es $[S' \to \cdot S, \$]$.
- Usando las reglas para el cálculo de la clausura debemos considerar todos los items de la forma $[A \to \alpha \cdot B\beta, \mathbf{a}]$ y en este caso corresponde

$$\begin{aligned} A &= S' \\ \alpha &= \lambda \\ B &= S \\ \beta &= \lambda \\ \mathbf{a} &= \$ \end{aligned}$$

  Luego calculamos $FIRST(\beta\mathbf{a}) = FIRST(\lambda\$) = \{\$\}$, y debemos agregar a $CLAUSURA(I_0)$ los items de la forma $[B \to \cdot\gamma, \$]$, i.e. $[S \to \cdot CC, \$]$.
- Al haber agregado $[S \to \cdot CC, \$]$, debemos repetir el cálculo de clausura sobre el nuevo item de la forma $[A \to \alpha \cdot B\beta, \mathbf{a}]$ y en este caso corresponde

$$\begin{aligned} A &= S \\ \alpha &= \lambda \\ B &= C \\ \beta &= C \\ \mathbf{a} &= \$ \end{aligned}$$

  Luego calculamos $FIRST(\beta\mathbf{a}) = FIRST(C\$) = \{\mathbf{c}, \mathbf{d}\}$ y debemos agregar a $CLAUSURA(I_0)$ los items de la forma $[B \to \cdot\gamma, \mathbf{c/d}]$, i.e. $[C \to \cdot\mathbf{c}C, \mathbf{c/d}]$ y $[C \to \cdot\mathbf{d}, \mathbf{c/d}]$.
- Al haber agregado cuatro nuevos items, debemos repetir el cálculo de clausura sobre cada uno de ellos. Sin embargo, como a la derecha del $\cdot$ no aparecen símbolos no terminales, no hay nada que agregar.

Construcción de Tablas de Parsing $LALR(1)$

El algoritmo de construcción queda como:

1. Aumentar la gramática original con un nuevo símbolo inicial S' tal que agregue la producción $S' \to S$ donde S es el símbolo inicial original de la gramática.
2. Enumerar contando desde cero cada producción de la gramática aumentada comenzando por la nueva producción inicial.

3. Calcular el autómata de prefijos viables a partir de los items $LR(1)$ de la gramática. Cada conjunto de items I_i servirá para construir el i-ésimo estado de la máquina parser $LALR(1)$.

4. Identificar conjuntos de items I_i e I_j que tengan **núcleo** idéntico, i.e. que los **núcleos** de **todos** los items contenidos en I_i sean los mismos (en número y forma) que los **núcleos** de **todos** los items contenidos en I_j. Eliminar ambos conjuntos y reemplazarlos por el conjunto I_{ij} que tenga los mismos **núcleos**, pero que combine los *lookaheads* de los conjuntos previos.

5. Construir la tabla de *acciones* usando una columna por cada símbolo terminal de la gramática aumentada incluyendo una para \$ y una fila por cada estado:

   a) Si en el autómata de prefijos viables hay una transición desde I_i hasta I_j con el símbolo terminal $\mathbf{x}$, entonces $acciones[i, \mathbf{x}] \leftarrow shift\,\mathbf{j}$.

   b) Si en el autómata de prefijos viables el i-ésimo conjunto contiene un item de la forma $[A \rightarrow \alpha\cdot, \mathbf{a}]$ y $A \rightarrow \alpha$ es la n-ésima producción de la gramática ($n > 0 \wedge A \neq S'$), entonces $acciones[i, \mathbf{a}] \leftarrow reduce\,\mathbf{n}$, i.e. llenar con *reduce* solamente las columnas que correspondan al *lookahead* calculado.

   c) Si en el autómata de prefijos viables el i-ésimo conjunto contiene un item de la forma $[S' \rightarrow S\cdot, \$]$, entonces $acciones[i, \$] \leftarrow accept$.

   d) El resto de la tabla de *acciones* se llena con *error*.

6. Construir la tabla de *goto* usando una columna por cada símbolo no-terminal de la gramática original y una fila por cada estado. Si en el autómata de prefijos viables hay una transición desde I_i hasta I_j con el símbolo no terminal X entonces $goto[i, X] \leftarrow j$.

Continuando con el Ejemplo 3 comenzamos por notar que los conjuntos I_3 e I_6 tienen el mismo **núcleo**. Por tanto podemos reemplazar ambos conjuntos por el conjunto

$$\begin{array}{rcl} I_{36} & : & [C \rightarrow C \cdot C, \mathbf{c}/\mathbf{d}/\$] \\ & & [C \rightarrow \cdot\mathbf{c}C, \mathbf{c}/\mathbf{d}/\$] \\ & & [C \rightarrow \cdot\mathbf{d}, \mathbf{c}/\mathbf{d}/\$] \end{array}$$

y notamos que todas las transiciones que llegaban al estado I_3 ahora llegan al estado I_{36}, así como también las transiciones que llegaban al estado I_6. Del mismo modo, las transiciones que salían del estado I_3 ahora salen desde el estado I_{36} con el mismo destino, así como también las transiciones que salían del estado I_6. Una observación similar podemos hacer con los conjunto I_4 e I_7 que son reemplazados por el conjunto

$$I_{47} \quad : \quad [C \rightarrow \mathbf{d}\cdot, \mathbf{c}/\mathbf{d}/\$]$$

y con los conjuntos I_8 e I_9 que son reemplazados por el conjunto

$$I_{89} \quad : \quad [C \rightarrow \mathbf{c}C\cdot, \mathbf{c}/\mathbf{d}/\$]$$

Nos queda entonces el conjunto reducido de items,

$$\begin{array}{rcl} I_0 & : & [S' \rightarrow \cdot S, \$] \\ & & [S \rightarrow \cdot CC, \$] \\ & & [C \rightarrow \cdot\mathbf{c}C, \mathbf{c}/\mathbf{d}] \\ & & [C \rightarrow \cdot\mathbf{d}, \mathbf{c}/\mathbf{d}] \\ I_1 & : & [S' \rightarrow S\cdot, \$] \\ I_2 & : & [S \rightarrow C \cdot C, \$] \\ & & [C \rightarrow \cdot\mathbf{c}C, \$] \\ & & [C \rightarrow \cdot\mathbf{d}, \$] \\ I_{36} & : & [C \rightarrow \mathbf{c} \cdot C, \mathbf{c}/\mathbf{d}/\$] \end{array}$$

$$
\begin{aligned}
& [C \to \cdot \mathbf{c}C, \mathbf{c}/\mathbf{d}/\$] \\
& [C \to \cdot \mathbf{d}, \mathbf{c}/\mathbf{d}/\$] \\
I_{47} \quad : \quad & [C \to \mathbf{d}\cdot, \mathbf{c}/\mathbf{d}/\$] \\
I_5 \quad : \quad & [S \to CC\cdot, \$] \\
I_{89} \quad : \quad & [C \to \mathbf{c}C\cdot, \mathbf{c}/\mathbf{d}/\$]
\end{aligned}
$$

y el autómata reducido de prefijos viables tiene la forma

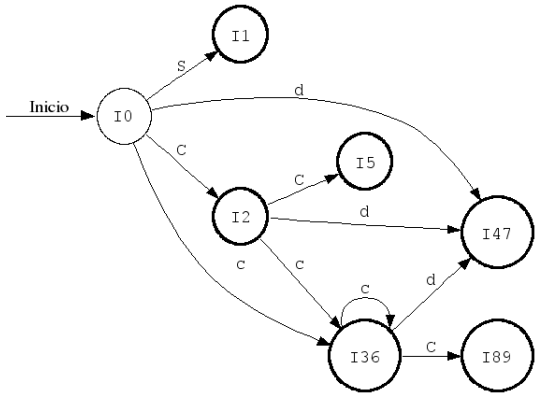

Figura 3: Autómata Reducido de Prefijos Viables para el Ejemplo 3

Finalmente construimos la tabla de parsing $LALR(1)$ según el algoritmo, quedando

	c	**d**	$	S	C
0	$shift$ **36**	$shift$ **47**		1	2
1			*accept*		
2	$shift$ **36**	$shift$ **47**			5
36	$shift$ **36**	$shift$ **47**			89
47	$reduce$ **3**	$reduce$ **3**	$reduce$ **3**		
5			$reduce$ **1**		
89	$reduce$ **2**	$reduce$ **2**	$reduce$ **2**		

Cuadro 2: Tabla de Parsing $LALR(1)$ para el Ejemplo 3