Compiladores
Análise Sintática Ascendente

Introdução

A ***análise sintática ascendente*** (***Bottom-Up***), também denominada de ***análise redutiva*** (ou ainda ***Shift-Reduce***), analisa uma sentença de entrada e tenta construir uma árvore de derivação, começando pelas ***folhas*** e prosseguindo para a ***raiz***, produzindo uma ***derivação mais à direita***, na ordem inversa. Caso seja obtida uma árvore cuja raiz tem por rótulo o símbolo inicial da gramática, e na qual a seqüência dos rótulos das folhas forma a sentença de entrada, então esta sentença pertence a linguagem gerada pela gramática, e a árvore obtida é a sua árvore de derivação.

Podemos pensar na análise ascendente como o processo de *reduzir* uma sentença de entrada a para o símbolo inicial S da gramática.

Por exemplo, seja a sentença ***abbcde*** e as produções gramaticais a seguir:

$S \rightarrow \mathbf{a}ABe$

$A \rightarrow Abc \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

Para reconhecer esta sentença devemos procurar por uma subcadeia que possa ser derivada por alguma das produções acima, e substituí-la pelo não-terminal do lado esquerdo da regra. O processo deve ser repetido até que a cadeia de entrada esteja reduzida ao símbolo inicial S.

Podemos demonstrar os passos de análise da cadeia acima através da tabela a abaixo.

Passos	Entrada	Regra Aplicada	Saída
1.	$a\underline{\mathbf{b}}bcde$	$A \rightarrow b$	$a\mathbf{A}bcde$
2.	$a\underline{\mathbf{Abc}}de$	$A \rightarrow Abc$	$a\mathbf{A}de$
3.	$aA\underline{\mathbf{d}}e$	$B \rightarrow d$	$aA\mathbf{B}e$
4.	$\mathbf{aABe}$	$S \rightarrow aABe$	$\mathbf{S}$

Na tabela podemos observar que o processo de redução corresponde exatamente a seguinte derivação mais a direita, na ordem inversa:

$$S \Rightarrow \underline{\mathbf{aABe}} \Rightarrow aA\underline{\mathbf{d}}e \Rightarrow aA\underline{\mathbf{bc}}de \Rightarrow a\underline{\mathbf{b}}bcdee$$

Também podemos observar que a análise da sentença de entrada é feita da esquerda para a direita, na tentativa de se encontrar uma subcadeia b em uma sentença agb, g contendo apenas símbolos terminais, tal que exista uma produção $d \rightarrow b$, cuja substituição de b por d permita a redução de adg, em zero ou mais passos, ao símbolo inicial. À forma sentencial $d \rightarrow b$, dá-se o nome de ***handle***. Se a subcadeia b estiver bem definida em agb e a produção $d \rightarrow b$ for clara no contexto, então dizemos que b é um *handle* de agb.

A tabela a seguir ilustra o *handle* correspondente a cada etapa de redução.

Passos	Entrada	Handle	Regra Aplicada	Saída
5.	$a\underline{\mathbf{b}}bcde$	b	$A \rightarrow b$	$a\mathbf{A}bcde$
6.	$a\underline{\mathbf{Abc}}de$	Abc	$A \rightarrow Abc$	$a\mathbf{A}de$
7.	$aA\underline{\mathbf{d}}e$	d	$B \rightarrow d$	$aA\mathbf{B}e$
8.	$\mathbf{aABe}$	$aABe$	$S \rightarrow aABe$	$\mathbf{S}$

Construção de Analisadores Ascendentes

A construção de *analisadores ascendentes* é feita através da implementação de *autômatos de pilha*, cujos controles são dirigidos por *tabelas de análise sintática*. Para tal é utilizado uma pilha sintática para guardar os símbolos gramaticais de uma sentença de entrada a ser decomposta. Usamos *$* para marcar o final da pilha e o final da sentença de entrada. Deste modo, no início do processo de análise deverá ser empilhado sobre a pilha o símbolo *$*. A sentença a ser analisada também deverá ser seguida de *$*. Assim se *a* é a cadeia de entrada, então teremos a seguinte configuração:

Pilha	Sentença de Entrada
$	*a*$

O processo de reconhecimento pelo analisador sintático, consiste em empilhar zero ou mais símbolos da sentença de entrada até que um *handle b* surja sobre o topo da pilha. Quando isto ocorre, reduz-se *b* para o não-terminal correspondente, empilhando-o. O processo se repete até que tenha sido detectado um erro ou que a pilha contenha em seu topo o não-terminal de partida seguido do símbolo *$*, e a entrada esteja vazia. Isto é:

Pilha	Sentença de Entrada
$*S*	$

Existem efetivamente quatro ações possíveis de serem realizadas pelo analisador sintático:

1. ***Empilhar (Shift)***: esta ação é executada para se colocar o símbolo de entrada sobre a pilha;
2. ***Reduzir (Reduce)***: esta ação é executada quando um *handle* está sobre a pilha. Neste caso, o analisador o substitui pelo não terminal correspondente;
3. ***Aceitar***: esta ação anuncia o término, com sucesso, do reconhecimento da sentença de entrada;
4. ***Erro***: esta ação é executada quando um erro sintático é encontrado. Neste caso o analisador sintático deverá chamar uma rotina de recuperação de erros.

Considere como exemplo as regras gramaticais:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

e a sentença $id_1 + id_2 * id_3$. O processo de reconhecimento é demonstrado abaixo.

Passo	Pilha	Entrada	Handle	Ação
1.	\$	$id_1 + id_2 * id_3\$$		Empilhar
2.	$\$id_1$	$+ id_2 * id_3\$$	id_1	Reduzir por $E \rightarrow id$
3.	$\$E$	$+ id_2 * id_3\$$		Empilhar
4.	$\$E +$	$id_2 * id_3\$$		Empilhar
5.	$\$E + id_2$	$* id_3\$$	id_2	Reduzir por $E \rightarrow id$
6.	$\$E + E$	$* id_3\$$	$E + E$	Reduzir por $E \rightarrow E + E$
7.	$\$E$	$* id_3\$$		Empilhar
8.	$\$E *$	$id_3\$$		Empilhar
9.	$\$E * id_3$	\$	id_3	Reduzir por $E \rightarrow id$
10.	$\$E * E$	\$	$E * E$	Reduzir por $E \rightarrow E + E$
11.	$\$E$	\$		Aceitar

Observe pela tabela que para ocorrer o processo de redução o *handle* deverá sempre estar sobre o topo da pilha.

Análise de Precedência de Operadores

Os analisadores de precedência de operadores operam sobre a classe de gramáticas denominadas de ***gramáticas de operadores***, que são aquelas que não contêm produções do tipo $A \rightarrow \varepsilon$ e, no lado direito das produções, os não terminais aparecem sempre separados por símbolos terminais, ou seja, não há dois ou mais não terminais adjacentes, tal como $A \rightarrow \alpha BC\delta$.

Por exemplo, seja a seguinte gramática:

$$E \rightarrow EOE \mid (\ E\ ) \mid -E \mid \mathbf{id}$$

$$O \rightarrow + \mid - \mid * \mid / \mid \wedge$$

Esta gramática não é de operadores, devido à produção $E \rightarrow EOE$, a qual possui três terminais consecutivos. Entretanto, podemos transformá-la para uma gramática de operadores, substituindo o não terminal O pelos terminais por ele gerado, obtendo:

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

A análise de precedência de operadores é bastante eficiente e é aplicada, principalmente, no reconhecimento de expressões. Entretanto, este método apresenta algumas desvantagens, dentre elas a dificuldade de lidar com operadores iguais que tenham significados diferentes (como por exemplo, o operador "–", o qual pode ser tanto binário como unário) e ser aplicado a apenas uma classe restrita de gramáticas.

Relações de Precedência de Operador

A partir de uma gramática de operadores podemos construir um analisador sintático utilizando-se das ***relações de precedência de operadores*** existentes entre os *tokens* a serem analisados.

Existem três relações de precedência de operadores:

1. $a \lessdot b$: a tem ***precedência menor*** que b;
2. $a \gtrdot b$: a tem ***precedência maior*** que b;
3. $a \doteq b$: a e b têm a mesma precedência.

A utilidade destas relações na análise de uma sentença é a identificação do *handle*:

- $\lessdot$: identifica o limite esquerdo do *handle*;
- $\gtrdot$: identifica o limite direito do *handle*;
- $\doteq$: indica que os terminais pertencem ao mesmo *handle*.

Devemos ter cuidado ao interpretar esses operadores pois, diferentemente dos operadores maior, menor e igual da matemática, é perfeitamente possível termos $a \lessdot b$ e $a \gtrdot b$ simultaneamente.

Os analisadores sintáticos de precedência de operadores são dirigidos por uma ***tabela de precedência***, cujas relações definem o movimento que o analisador deve fazer: *empilhar*, *reduzir*, *aceitar* ou *chamar uma rotina para tratamento de erros*. Esta tabela é uma matriz quadrada que relaciona todos os terminais da gramática mais o marcador *$*. Os terminais nas ***linhas*** representam ***terminais no topo da pilha***, e os terminais nas ***colunas*** representam ***terminais sob a cabeça de leitura***.

Por exemplo, sejam as produções a seguir:

$$E \rightarrow E \text{ Ú } E \mid E \text{ Ù } E \mid \text{(E)} \mid id$$

A tabela de precedência de operadores para esta gramática é indicada abaixo.

	id	Ú	Ù	**(**	**)**	**$**
id		$\gtrdot$	$\gtrdot$		$\gtrdot$	$\gtrdot$
Ú	$\lessdot$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
Ù	$\lessdot$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\gtrdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	
)		$\gtrdot$	$\gtrdot$		$\gtrdot$	$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$		Aceita

Basicamente, um analisador de precedência funciona da seguinte forma. Seja a o terminal mais ao topo da pilha, desprezando-se qualquer não terminal que possa ocorrer, e b o terminal sob a cabeça de leitura:

1. Se $a \lessdot b$ ou $a \doteq b$, então *empilha*;
2. Se $a \gtrdot b$, então procure um *handle* na pilha, o qual deverá estar delimitado pelas relações $\lessdot$ e $\gtrdot$, e o substitui pelo não terminal correspondente.

Deve ser observado que o *handle* vai desde o topo da pilha até o primeiro terminal a, inclusive, que tem abaixo de si um terminal b, tal que $b \lessdot a$.

A Figura 1 ilustra como funciona um analisador deste tipo. Nela podemos identificar, de acordo com a tabela de precedência definida anteriormente, que sobre o topo da pilha temos o *handle* E Ù E, pois, neste caso, Ù é o terminal mais ao topo da pilha, ***$*** é o terminal sob a cabeça de leitura e Ù ***$***. Logo, o *handle* vai desde o topo da pilha até o não terminal E que antecede o terminal Ú, pois Ú Ù.

Para ilustrar o funcionamento de um analisador de precedência de operadores, suponha que se deseje reconhecer a sentença: id_1 Ú id_2 Ù id_3; considerando a tabela de precedência de operadores definida anteriormente. Os movimentos efetuados pelo analisador sintático são indicados abaixo.

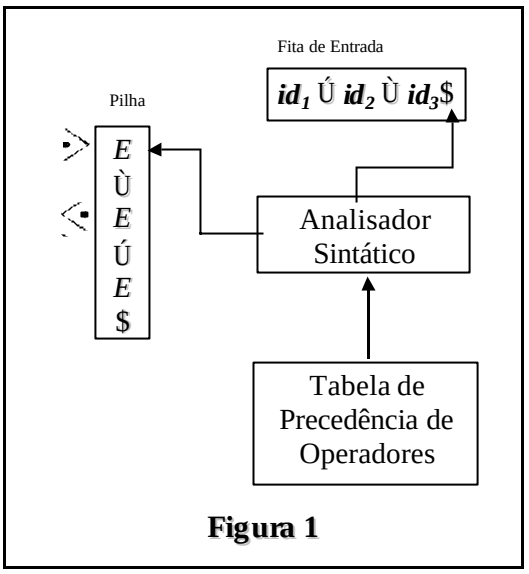

Figura 1

Passo	Pilha	Relação	Entrada	Handle	Ação
1.	$		id_1 Ú id_2 Ù $id_3$$		Empilhar
2.	$$id_1$		Ú id_2 Ù $id_3$$	id_1	Reduzir por $E \rightarrow id$
3.	$$E$		Ú id_2 Ù $id_3$$		Empilhar
4.	$$E$ Ú		id_2 Ù $id_3$$		Empilhar
5.	$$E$ Ú id_2		Ù $id_3$$	id_2	Reduzir por $E \rightarrow id$
6.	$$E$ Ú E		Ù $id_3$$		Empilhar
7.	$$E$ Ú E Ù		$id_3$$		Empilhar
8.	$$E$ Ú E Ù id_3		$	id_3	Reduzir por $E \rightarrow id$
9.	$$E$ Ú E Ù E		$	E Ù E	Reduzir por $E \rightarrow E$ Ù E
10.	$$E$ Ú E		$	E Ú E	Reduzir por $E \rightarrow E$ Ú E
11.	$$E$		$		Aceitar

Os movimentos de um analisador de precedência de operadores são efetuados de acordo com um algoritmo, o qual recebe como entrada uma tabela de precedência de operadores, ***t***, e uma cadeia ***w*** a ser analisada. Este algoritmo é indicado a seguir.

```
Algoritmo PrecedênciaOperadores(t, w)
Inicio
     Seja w$ a cadeia de entrada
     Seja S o símbolo de partida
     Repita Sempre
          Se   $S está no topo da pilha
          e    $ está sob a cabeça de leitura, então
               Aceite a cadeia de entrada e pare
          Senão
               Seja a o terminal ao topo da Pilha
               Seja b o terminal sob a cabeça de leitura
               Se   a ⋖· b ou a ≐ b, então
                    Empilhar b
                    Avançar a cabeça de leitura
               Senão
                    Se   a ·⋗ b, então
                         Repita
                              Desempilhar
                         Até encontrar a relação ⋖· entre o
                         terminal do topo da pilha e o último
                         terminal desempilhado
                         // Seja a produção X → α, sendo α o
                         // handle desempilhado
                         // Neste caso, X deverá ser empilhado
                         Empilhar o não terminal correspondente
                    Senão
                         Chamar a rotina de tratamento de erros
                    Fim  Se
               Fim  Se
          Fim  Se
     Fim  Repita
Fim
```

Construção da Tabela de Precedência de Operadores

Para construir a tabela de precedência de operadores, podemos utilizar dois métodos, os quais nos permitem computar as relações de precedência entre os símbolos terminais de uma gramática de operadores: o método ***intuitivo*** e o ***mecânico***.

Método Intuitivo

Este método permite obter as relações de equivalência a partir do conhecimento prévio da associatividade e precedência dos operadores da gramática.

Sejam dois operadores θ_1 e θ_2. O algoritmo funciona do seguinte modo, lembrando-se que o lado direito da relação corresponde à linha da tabela (terminal mais ao topo da pilha) e o lado direito corresponde à coluna da tabela (terminal sob a cabeça de leitura):

1. Se o operador θ_1 tem maior precedência sobre o operador θ_2, então fazemos $\theta_1 \dot{>} \theta_2$ e $\theta_2 \dot{<} \theta_1$.
2. Se os operadores θ_1 e θ_2, têm a mesma precedência, isto é, precedência igual, então:
   2.1. Se são associativos à esquerda, então fazemos $\theta_1 \dot{>} \theta_2$ e $\theta_2 \dot{>} \theta_1$;
   2.2. Se são associativos à direita, então fazemos $\theta_1 \dot{<} \theta_2$ e $\theta_2 \dot{<} \theta_1$.
3. As relações entre os operadores e os demais *tokens* (operandos e delimitadores) são fixas. Para todo operador θ, temos:

   $\theta \dot{<} id$ e $id \dot{>} \theta$,
   $\theta \dot{<} ($ e $(\dot{<} \theta$,
   $\theta \dot{>})$ e $) \dot{>} \theta$,
   $\theta \dot{>} \$$ e $\$ \dot{<} \theta$;
4. As relações entre os *tokens* que não são operadores também são fixas:

   $) \dot{>})$ e $(\dot{<} (,$
   $id \dot{>})$ e $(\dot{<} id,$
   $id \dot{>} \$$ e $\$ \dot{<} id,$
   $) \dot{>} \$$ e $\$ \dot{<} (,$
   $(\doteq).$

Seja a gramática de operadores a seguir:

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E \wedge E \mid (E) \mid id$$

Têm-se as seguintes precedências e associatividade entre os operadores:

1. **^**: tem maior precedência e é associativo à direita;
2. ***** e **/**: tem precedência intermediária e são associativos à esquerda;
3. **+** e **-**: tem menor precedência e são associativos à esquerda.

A tabela de precedência de operadores para esta a gramática é indicada abaixo. As posições em branco são assinaladas como ***erro***.

	id	**+**	*****	**–**	**/**	**^**	**(**	**)**	**$**
id		$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$		$\cdot>$	$\cdot>$
+	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
*****	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
–	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$	$<\cdot$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
/	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
^	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$	$\cdot>$
(	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$\doteq$	
)		$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$	$\cdot>$		$\cdot>$	$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$	$<\cdot$		Aceita

Observe que na gramática acima a produção: $E \rightarrow - E$; foi eliminada, impedindo assim que o operador "–" seja utilizado como binário e unário. Para termos os dois significados para o mesmo operador, o analisador léxico deve distinguir entre os dois tipos, por exemplo, representando o operador unário por "–u", o qual é o caso quando o operador é precedido de outro operador, de abre parênteses, de vírgula ou de um símbolo de atribuição.

Método Mecânico

Este método obtém as relações de precedência diretamente a partir da gramática de operadores, a qual não pode ser ambígua.

O algoritmo funciona do seguinte modo:

1. Defina ***leading(A)***[1] para o não terminal ***A*** como sendo o conjunto dos terminais ***a***, tal que ***a*** é o terminal ***mais à esquerda*** em alguma forma sentencial derivada de ***A***;
2. Defina ***trailing(A)***[2] para o não terminal ***A*** como sendo o conjunto dos terminais ***a***, tal que ***a*** é o terminal ***mais à direita*** em alguma forma sentencial derivada de ***A***;
3. Para cada dois terminais ***a*** e ***b***, temos:
   - 3.1. $a \doteq b$, se existe um lado direito de produção da forma $\alpha a \beta b \gamma$, onde β é ε ou é um não terminal e α e γ são arbitrários;
   - 3.2. $a \lessdot b$. se existe um lado direito de produção da forma $\alpha a A \gamma$, e ***b*** está em ***leading(A)***;
   - 3.3. $a \gtrdot b$, se existe um lado direito de produção da forma $\alpha A b \gamma$ e ***a*** está em ***trailing(A)***;
   - 3.4. $\$ \lessdot b$, se ***S*** é o símbolo de partida da gramática e, para qualquer ***b***, $b \in$ ***leading(S)***;
   - 3.5. $a \gtrdot \$$, se ***S*** é o símbolo de partida da gramática e, para qualquer ***a***, $a \in$ ***trailing(S)***;

Como exemplo de aplicação do algoritmo, suponha a gramática de operadores:

$$E \rightarrow E + E \mid E * E \mid E \wedge E \mid (E) \mid id$$

Esta gramática é ambígua. Portanto, não podemos aplicar o algoritmo sem antes eliminarmos a ambigüidade. Isto pode ser feito transformando as produções para expressar a precedência e a associatividade dos operadores. Consegue-se isto introduzindo um símbolo não terminal para cada nível de precedência. No caso de

---

[1] A tradução para ***leading***, neste contexto, seria *à frente*. Portanto, ***leading*(*A*)** seria o conjunto formado por todos os símbolos terminais à frente de uma forma sentencial derivada de ***A***, isto é, os primeiros símbolos terminais (não confundir com o conjunto *Primeiro*(*A*)).

[2] A tradução para ***trailing***, neste contexto, seria *finaliza*. Portanto, ***trailing*(*A*)** seria o conjunto formado por todos os símbolos terminais que finalizam uma forma sentencial derivada de ***A***, isto é, os últimos símbolos terminais.

associatividade à esquerda, a avaliação será feita da esquerda para a direita. Já para a associatividade à direita, a avaliação será feita da direita para a esquerda. Portanto, temos a seguinte gramática:

$E \rightarrow E + T \mid T$ // *Operador de menor precedência e associativo à esquerda*
$T \rightarrow T * F \mid F$ // *Operador de precedência intermediária e associativo à esquerda*
$F \rightarrow P \wedge F \mid P$ // *Operador de maior precedência e associativo à direita*
$P \rightarrow (\ E\ ) \mid id$ // *Operandos*

Os conjuntos ***leading*** e ***trailing*** são indicados pela tabela abaixo.

	Leading	***Trailing***
E	**+, *, ^, (, *id***	**+, *, ^,), *id***
T	***, ^, (, *id***	***, ^,), *id***
F	**^, (, *id***	**^,), *id***
P	**(, *id***	**), *id***

Agora, calculemos as relações de precedência:

1. ***a*** $\doteq$ ***b***: examinar todos os lados direitos das produções procurando por formas $\alpha a \beta b \gamma$, onde β é ε ou é um não terminal e α e γ são arbitrários. Neste caso, a única produção que se enquadra é $P \rightarrow (\ E\ )$. Portanto, temos:
   ($\doteq$ **)**

2. ***a*** $\lessdot$ ***b***: examinar todos os lados direitos das produções procurando por formas $\alpha a A \gamma$, tal que ***b*** está em ***leading(A)***. Neste caso, temos as produções:

   $P \rightarrow E + T$ (par + T): o que nos dá a relação: **+** $\lessdot$ **{*, ^, (, *id*}**

   $T \rightarrow T * F$ (par * F): o que nos dá a relação: ***** $\lessdot$ **{^, (, *id*}**

   $F \rightarrow P \wedge F$ (par ^ F): o que nos dá a relação: **^** $\lessdot$ **{^, (, *id*}**

   $F \rightarrow (\ E\ )$ (par '(' E): o que nos dá a relação: **(** $\lessdot$ **{+, *, ^, (, *id*}**

3. $a \overset{\bullet}{>} b$: examinar todos os lados direitos das produções procurando por formas $\alpha A b \gamma$, tal que e a está em ***trailing(A)***. Neste caso, temos as produções:

   $P \rightarrow E + T$ (par E +): o que nos dá a relação: {**+, *, ^,), *id***} $\overset{\bullet}{>}$ **+**

   $T \rightarrow T * F$ (par T *): o que nos dá a relação: {***, ^,), *id***} $\overset{\bullet}{>}$ *****

   $F \rightarrow P \wedge F$ (par P ^): o que nos dá a relação: {**), *id***} $\overset{\bullet}{>}$ **^**

   $F \rightarrow (\ E\ )$ (par F '**)**'): o que nos dá a relação: {**+, *, ^,), *id***} $\overset{\bullet}{>}$ **)**

4. **$** $\overset{\bullet}{<}$ ***b***, se ***S*** é o símbolo de partida da gramática e, para qualquer ***b***, ***b*** $\in$ ***leading(S)***. Como **$** tem precedência menor que qualquer terminal em ***leading(E)***, então temos:

   $ $\overset{\bullet}{<}$ {**+, *, ^, (, *id***}

5. ***a*** $\overset{\bullet}{>}$ ***$***, se ***S*** é o símbolo de partida da gramática e, para qualquer ***a***, ***a*** $\in$ ***trailing(S)***. Como qualquer terminal em ***trailing(E)*** possui precedência maior que $, então temos:

   {**+, *, ^,), *id***} $\overset{\bullet}{>}$ ***$***

A tabela de precedência obtida a partir deste algoritmo é indicada abaixo.

	id	**+**	*****	**^**	**(**	**)**	**$**
id		$\overset{\bullet}{>}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$		$\overset{\bullet}{>}$	$\overset{\bullet}{>}$
+	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$
*****	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$
^	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$
(	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{=}$	
)		$\overset{\bullet}{>}$	$\overset{\bullet}{>}$	$\overset{\bullet}{>}$		$\overset{\bullet}{>}$	$\overset{\bullet}{>}$
$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$	$\overset{\bullet}{<}$		Aceita

Funções de Precedência

Podemos usar funções para indicar ao analisador a precedência de operadores, substituindo assim a tabela de precedência. As funções f e g mapeiam terminais em inteiros. Estes inteiros indicam a precedência. Sejam a e b terminais:

- $f(a) < g(b)$ sempre que $a \lessdot b$;
- $f(a) = g(b)$ sempre que $a \doteq b$;
- $f(a) > g(b)$ sempre que $a \gtrdot b$.

O método apresenta algumas desvantagens, como não representar as entradas de erros. Além disto, nem sempre é possível obter as funções f e g.

O algoritmo para obter as funções de precedência é o seguinte:

1. Criar símbolos fa e ga para cada a elemento de V_T e $\$$.
2. Distribuir os símbolos criados em grupos, tal que:
   2.1. Se $a \doteq b$, então fa e gb estão no mesmo grupo;
   2.2. Se $a \doteq b$ e $c \doteq b$, fa e fc deverão ficar no mesmo grupo que gb;
   2.3. Se, ainda, $c \doteq d$, então fa, fc, gb e gd deverão ficar no mesmo grupo, mesmo que $a \doteq d$ não ocorra.
3. Gerar um grafo dirigido cujos **nós** são os grupos formados anteriormente. Para quaisquer a e b:
   3.1. Se $a \lessdot b$, construir um arco do grupo fa para o grupo gb; e
   3.2. Se $a \gtrdot b$ fazer um arco de gb para fa.
4. Se o grafo contiver ciclos, então as funções de precedência não existem. Se não houver ciclos, então $f(a)$ é igual ao caminho mais longo iniciando em fa e $g(a)$ é igual ao caminho mais longo iniciando em ga.

Por exemplo, seja a gramática:

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

A tabela de precedência de operadores para esta gramática é indicada abaixo.

	id	**+**	*****	**(**	**)**	**$**
id		$\bullet\!>$	$\bullet\!>$		$\bullet\!>$	$\bullet\!>$
+	$<\!\bullet$	$\bullet\!>$	$<\!\bullet$	$<\!\bullet$	$\bullet\!>$	$\bullet\!>$
*****	$<\!\bullet$	$\bullet\!>$	$\bullet\!>$	$<\!\bullet$	$\bullet\!>$	$\bullet\!>$
(	$<\!\bullet$	$<\!\bullet$	$<\!\bullet$	$<\!\bullet$	$\doteq$	
)		$\bullet\!>$	$\bullet\!>$		$\bullet\!>$	$\bullet\!>$
$	$<\!\bullet$	$<\!\bullet$	$<\!\bullet$	$<\!\bullet$		Aceita

Pelo algoritmo proposto temos o grafo indicado pela Figura 2. As funções f e g encontram-se indicadas abaixo.

	+	*	(	)	id	$
f	2	4	0	4	4	0
g	1	3	5	0	5	0

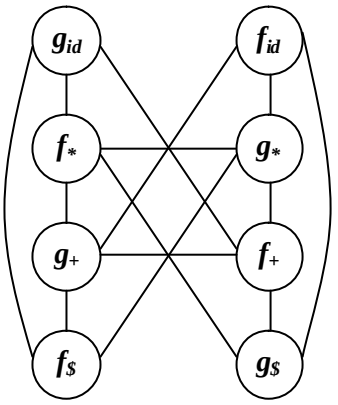

Figura 2

Análise LR(*k*)

Analisa-se a seguir uma técnica eficiente de análise ascendente, a qual pode ser utilizada para decompor uma ampla classe de gramáticas livres de contexto. A técnica é chamada de análise ***LR*(*k*)**, onde o ***L*** significa que a entrada é lida da esquerda para a direita (*left-to-right*), o ***R*** significa a construção da derivação mais à direita em reverso (*rightmost-derivation*), e o ***k*** é o número de símbolos de entrada que devem ser lidos (*lookahead*) para que o analisador possa tomar decisões. Dentre as vantagens destes analisadores destacam-se:

1. São capazes de reconhecer, praticamente, todas as estruturas sintáticas definidas por uma gramática livre de contexto;
2. É o método mais geral e eficiente que o de precedência de operadores e qualquer outro tipo de analisador ***shift-reduce***, podendo ser implementado com o mesmo grau de eficiência;
3. São capazes de descobrir erros sintáticos tão cedo quanto possível, numa varredura de entrada da esquerda para direita.

A principal desvantagem deste método está na dificuldade para se construir um analisador sintático ***LR*** manualmente, sendo normalmente, necessário a utilização de ferramentas especializadas, denominadas de geradores de analisadores (*parser-generator*), como por exemplo o YACC (***Y****et* ***A****nother* ***C****ompiler-****C****ompiler*) e o BISON.

Há basicamente três tipos de analisadores ***LR***:

1. ***SLR*** (*Simple LR*): de fácil implementação, porém aplicáveis a uma classe restrita de gramáticas;
2. ***LR Canônicos***: são os mais poderosos, podendo ser aplicados a um grande número de linguagens livres de contexto; e
3. ***LALR*** (*Look Ahead LR*): de nível intermediário e implementação eficiente, que funciona para a maioria das linguagens de programação – é o método utilizado pelo YACC.

Basicamente, um analisador ***LR*** é composto por uma *fita de entrada*, uma *pilha*, um *programa* e uma *tabela sintática*, a qual é composta por duas partes: ***ação*** (*action*) e ***desvio*** (*goto*). O *programa* é o mesmo para os três tipos de analisadores ***LR***, apenas a *tabela sintática* muda de um para outro. A Figura 3 ilustra a forma esquemática de um analisador ***LR***. A fita de entrada mostra a sentença a ser analisada, finalizada pelo marcador ***$***. A pilha armazena os símbolos X_i da gramática intercalados por estados s_m do analisador, formando o par $X_m s_m$. O símbolo na base da pilha é sempre S_0, ou seja, o estado inicial do analisador.

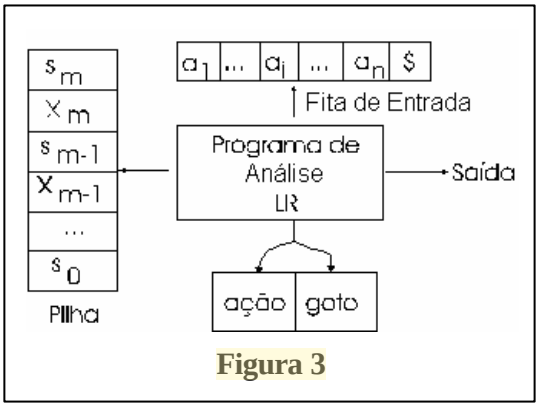

Figura 3

Na tabela sintática, a parte ***ação*** contém as seguintes ações, as quais estão associadas aos estados e aos símbolos terminais lidos da entrada:

1. Empilhar *s*, onde *s* é um estado;
2. Reduzir usando uma produção $A \rightarrow b$;
3. Aceitar; ou
4. Erro.

A parte ***desvio*** contém transições de estados em relação aos símbolos não terminais da gramática.

Basicamente, o analisador funciona como se segue. Seja s_m o estado no topo da pilha e a_i o terminal sob a cabeça de leitura. O analisador consulta a tabela ***Ação***$[s_m, a_i]$, a qual pode assumir um dos valores:

1. Empilha *s*: causa o empilhamento de $a_i s$;
2. Reduzir $A \rightarrow b$: causa o desempilhamento de $2r$ símbolos, onde $r = |b|$, e o empilhamento de $A s_j$, onde s_j resulta da consulta à tabela ***desvio***$[s_{m-r}, A]$;
3. Aceita: o analisador reconhece a sentença como válida; e
4. Erro: o analisador pára a execução, identificando um erro sintático.

O funcionamento de um analisador LR pode ser entendido considerando as transformações que ocorrem na pilha e na fita de entrada, denominada de configuração do analisador, a qual é representada pelo par (*pilha*, *fita*). Portanto, a configuração inicial de um analisador LR é dada por $(s_0, a_1 a_2 \ldots a_n \$)$.

Considerando a configuração ($s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m,\ a_i\ a_{i+1}\ \ldots\ a_n\ \$$), têm-se a seguinte configuração após a aplicação de cada ação:

1. Se Ação[s_m, a_i] = empilhar s, então tem-se a nova configuração: ($s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m\ a_i\ s,\ a_{i+1}\ \ldots\ a_n\ \$$). Observe que a cabeça de leitura é avançada para o próximo símbolo na entrada;
2. Se Ação[s_m, a_i] = reduzir $A \rightarrow b$, então tem-se a nova configuração: ($s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_{m-r}\ s_{m-r}\ A\ s,\ a_i\ a_{i+1}\ \ldots\ a_n\ \$$); onde s = ***desvio***[s_{m-r}, A] e r = |b|. Nesse caso, o analisador desempilha $2r$ símbolos da pilha, deixando no topo s_{m-r} e empilha A e s. Observe que a cabeça de leitura não é avançada, mantendo o mesmo símbolo à entrada;
3. Se ***ação***[s_m, a_i] = aceitar, então o analisador conclui com sucesso; e
4. Se ***ação***[s_m, a_i] = erro, então o analisador pára emitindo uma mensagem de erro, ou chama uma rotina de tratamento de erro.

Como exemplo, considere a gramática abaixo:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

A tabela sintática ***LR*** para esta gramática é indicada a seguir, onde os símbolos ***si*** e ***ri*** significam, respectivamente, empilhar ***i*** e reduzir ***i***. As entradas não definidas são de erro.

	Ação						Desvio		
Estado	***id***	+	*	(	)	***$***	*E*	*T*	*F*
0	***s5***			***s4***			1	2	3
1		***s6***				***aceita***			
2		***r2***	***s7***		***r2***	***r2***			
3		***r4***	***r4***		***r4***	***r4***			
4	***s5***			***s4***			8	2	3
5		***r6***	***r6***		***r6***	***r6***			
6	***s5***			***s4***				9	3
7	***s5***			***s4***					10
8		***s6***			***s11***				
9		***r1***	***s7***		***r1***	***r1***			
10		***r3***	***r3***		***r3***	***r3***			
11		***r5***	***r5***		***r5***	***r5***			

As configurações do analisador no reconhecimento da entrada: $id_1 * id_2 + id_3$; é indicado abaixo.

Passo	Pilha	Fita de Entrada	Ação
1.	0	$id_1 * id_2 + id_3 \$$	Empilhar
2.	0 id_1 5	$* id_2 + id_3 \$$	Reduzir $F \rightarrow id$
3.	0 F 3	$* id_2 + id_3 \$$	Reduzir $T \rightarrow F$
4.	0 T 2	$* id_2 + id_3 \$$	Empilhar
5.	0 T 2 * 7	$id_2 + id_3 \$$	Empilhar
6.	0 T 2 * 7 id_2 5	$+ id_3 \$$	Reduzir $F \rightarrow id$
7.	0 T 2 * 7 F 10	$+ id_3 \$$	Reduzir $T \rightarrow T * F$
8.	0 T 2	$+ id_3 \$$	Reduzir $E \rightarrow T$
9.	0 E 1	$+ id_3 \$$	Empilhar
10.	0 E 1 + 6	$id_3 \$$	Empilhar
11.	0 E 1 + 6 id_3 5	$\$$	Reduzir $F \rightarrow id$
12.	0 E 1 + 6 F 3	$\$$	Reduzir $T \rightarrow F$
13.	0 E 1 + 6 T 9	$\$$	Reduzir $E \rightarrow E + T$
14.	0 E 1	$\$$	Aceita

O algoritmo para um analisador LR é indicado a seguir, o qual recebe como entrada um tabela sintática *t* e a cadeia *w* a ser analisada.

```
Algoritmo AnaliseLR(t, w)
Inicio
   Seja w$ a cadeia de entrada
   Empilhar S, o símbolo de partida, na pilha
   Faça ip apontar para o primeiro símbolo de w$
   Repita sempre
       Seja s o estado ao topo da pilha
       Seja a o símbolo apontado por ip
       Se  ação[s, a] = empilhar x então
           Empilhar a
           Empilhar x
           Avançar ip para o próximo símbolo da entrada
       Senão
           Se  ação[s, a] = reduzir A → β então
               Desempilhar 2*|β| símbolos da pilha
               Seja x o estado ao topo da pilha
               Empilhar A
               Empilhar desvio[x, A]
               // Neste ponto executa-se qualquer ação
               // desejada, como por exemplo escrever
               // a redução A → β
           Senão
               Se  ação[s, a] = aceitar então
                   Retornar
               Senão
                   Chamar rotina de tratamento de erro
               Fim Se
           Fim Se
       Fim Se
   Fim Repita
Fim
```

Construção de Analisadores Sintáticos SLR

Conforme citado anteriormente, o que varia em um método de análise ***LR*** é a sua tabela sintática. Por sua facilidade de implementação, apresentaremos agora o algoritmo para construção da tabela sintática para o método ***SLR***.

A construção de analisadores SLR baseia-se no que se denomina ***conjunto canônico de itens LR(0)***. Um ***item LR(0)*** (ou simplesmente ***item***) de uma gramática ***G*** é uma produção de ***G*** com um ponto em alguma posição do lado direito da produção. Por exemplo, para a produção $A \rightarrow X\ Y\ Z$ tem-se quatro itens: $A \rightarrow \bullet X\ Y\ Z$, $A \rightarrow X \bullet Y\ Z$, $A \rightarrow X\ Y \bullet Z$, $A \rightarrow X\ Y\ Z \bullet$. A produção $A \rightarrow \varepsilon$ gera apenas um ítem: $A \rightarrow \bullet$.

Intuitivamente podemos dizer que o ***item*** indica o quanto de uma produção já foi visto até um determinado momento no processo de análise.

A idéia central no método ***SLR*** é de construir, a partir de uma gramática, um autômato finito determinístico que reconheça ***prefixos viáveis***. ***Prefixos viáveis*** são formas sentenciais direitas que aparecem no topo da pilha de um analisador do tipo empilhar-reduzir. Os ***itens*** são agrupados em conjuntos que dão origem aos estados do analisador ***SLR***. São estes ***estados*** que irão reconhecer os prefixos viáveis.

A construção do ***conjunto canônico de ítens LR(0)***, ou simplesmente ***conjunto canônico LR(0)***, para uma gramática ***G***, requer duas operações e nas funções ***closure*** (***fechamento***) e ***goto*** (***desvio***):

1. Adicionar à gramática ***G*** a produção ***S'*** ® ***S***, onde ***S*** é o símbolo inicial de ***G*** gramática e ***S'*** é um novo símbolo não terminal, gerando a gramática ***G'***, denominada de ***gramática aumentada***; e
2. Computar a função ***closure*** e ***goto*** para ***G'***.

O objetivo de se acrescentar a produção ***S'*** ® ***S*** à ***G*** é indicar o momento em que o processo de análise acaba e aceita a sentença de entrada. Ou seja, isto ocorre quando o analisador está por reduzir ***S'*** ® ***S***.

Operação Closure

Se ***I*** é um ***conjunto de itens LR(0)*** de ***G***, então ***closure(I)*** é o ***conjunto de itens*** construído a partir de ***I*** pelas seguintes regras:

1. Todo ***item*** de ***I*** é adicionado em ***closure(I)***; e

2. Se $A \rightarrow \alpha \bullet B\ \beta$ está em ***closure(I)*** e $B \rightarrow \gamma$ pertence a ***G***, então adicione o ***item*** $B \rightarrow \bullet \gamma$ em ***I***, se ele já não estiver lá. Repita esta regra até que nenhum novo ***item*** possa ser adicionado.

Intuitivamente o item $A \rightarrow \alpha \bullet B\ \beta$ em ***closure(I)*** indica que, em algum ponto do processo de análise, esperamos ver uma cadeia derivável a partir de $B\beta$. Se $B \rightarrow \gamma$ é uma produção de ***G***, então também esperamos ver uma cadeia derivável de γ neste ponto. Por esta razão incluímos $B \rightarrow \bullet \gamma$ em ***closure(I)***.

Por exemplo, seja a gramática aumentada ***G***:

$E' \rightarrow E$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (\ E\ ) \mid id$

Se $I = \{E' \rightarrow \bullet E\}$, então ***closure(I)*** contém os seguintes itens:

$E' \rightarrow \bullet E$

$E \rightarrow \bullet E + T$

$E \rightarrow \bullet T$

$T \rightarrow \bullet T * F$

$E \rightarrow \bullet F$

$F \rightarrow \bullet (\ E\ )$

$F \rightarrow \bullet id$

Operação goto

Informalmente, ***goto(I, X)***, avanço de I através de X, consiste em coletar as produções com ponto no lado esquerdo de X, X terminal ou não terminal ($X \in V_N \cup V_T$), avançar o ponto de uma posição e obter a função ***closure*** deste conjunto.

Formalmente, para $X \in (V_N \cup V_T)$, a função ***goto(I, X)*** é a função ***closure*** do conjunto dos ***ítens*** $\{A \rightarrow \alpha\, X \bullet \beta\}$ tal que $\{A \rightarrow \alpha \bullet X\, \beta\} \in I$.

Por exemplo, seja a gramática aumentada G usada anteriormente. Se $I = \{E' \rightarrow E \bullet, E \rightarrow E \bullet + T\}$, então ***goto(I, +)*** é:

$E \rightarrow E + \bullet T$

$T \rightarrow \bullet T * F$

$T \rightarrow \bullet F$

$F \rightarrow \bullet (E)$

$F \rightarrow \bullet id$

Algoritmo para a Construção do Conjunto Canônico de Itens LR(0)

Para uma gramática G, o ***conjunto canônico de itens LR(0)***, referido por C, é obtido pelo algoritmo abaixo.

```
Algoritmo Closure(G)
Inicio
   C <- I0 = closure({S' → •S})
   repita
      Para cada conjunto de itens I em C e
      Para cada símbolo X de G, tal que
      goto(I,X) ≠ ∅ e goto(I,X) ∉ C Faça
          Adicione goto(I,X) em C
      Até que nenhum conjunto de itens possa ser adicionado à C
Fim
```

Por exemplo, seja a gramática aumentada ***G***:

$E' \rightarrow E$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

O ***conjunto canônico de itens LR(0)*** para ***G*** é dado por:

1. Inicialização: $C = \{I_0 = closure(\{E' \rightarrow \bullet E\})\}$

   $I_0 = \{E' \rightarrow \bullet E, E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id\}$

2. Repita: *Para todo* $I \in C$ *e* $X \in G$, *calcular* $goto(I, X)$ *e adicionar a* C

   $I_1 = goto(I_0, E) = closure(\{E' \rightarrow E \bullet, E \rightarrow E \bullet + T\}) = \{E' \rightarrow E \bullet, E \rightarrow E \bullet + T\}$;

   $I_2 = goto(I_0, T) = closure(\{E \rightarrow T \bullet, T \rightarrow T \bullet * F\}) = \{E \rightarrow T \bullet, T \rightarrow T \bullet * F\}$;

   $I_3 = goto(I_0, F) = closure(\{T \rightarrow F \bullet\}) = \{T \rightarrow F \bullet\}$;

   $I_4 = goto(I_0, \text{'('}) = closure(\{F \rightarrow (\bullet E)\}) = \{F \rightarrow (\bullet E), E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id\}$

   $I_5 = goto(I_0, id) = closure(\{F \rightarrow id \bullet\}) = \{F \rightarrow id \bullet\}$

   $I_6 = goto(I_1, \text{'+'}) = closure(\{E \rightarrow E + \bullet T\}) = \{E \rightarrow E + \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id\}$

   $I_7 = goto(I_2, \text{'*'}) = closure(\{T \rightarrow T * \bullet F\}) = \{T \rightarrow T * \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id\}$

   $I_8 = goto(I_4, E) = closure(\{F \rightarrow (E \bullet), E \rightarrow E \bullet + T\}) = \{F \rightarrow (E \bullet), E \rightarrow E \bullet + T\}$

   $goto(I_4, T) = closure(\{E \rightarrow T \bullet, T \rightarrow T \bullet * F\}) = I_2$, o qual já foi incluído

   $goto(I_4, F) = closure(\{T \rightarrow F \bullet\}) = I_3$, o qual já foi incluído

   $goto(I_4, \text{'('}) = closure(\{F \rightarrow (\bullet E)\}) = I_4$, o qual já foi incluído

   $goto(I_4, id) = closure(\{F \rightarrow id \bullet\}) = I_5$, o qual já foi incluído

   $I_9 = goto(I_6, T) = closure(\{E \rightarrow E + T \bullet, T \rightarrow T \bullet * F\}) = \{E \rightarrow E + T \bullet, T \rightarrow T \bullet * F\}$

   $goto(I_6, F) = closure(\{T \rightarrow F \bullet\}) = I_3$, o qual já foi incluído

$goto(I_6, \text{'('}) = closure(\{F \rightarrow (\bullet E)\}) = I_4$, o qual já foi incluído

$goto(I_6, id) = closure(\{F \rightarrow id \bullet\}) = I_5$, o qual já foi incluído

$I_{10} = goto(I_7, F) = closure(\{T \rightarrow T * F \bullet\}) = \{T \rightarrow T * F \bullet\}$

$goto(I_7, \text{'('}) = closure(\{F \rightarrow (\bullet E)\}) = I_4$, o qual já foi incluído

$goto(I_7, id) = closure(\{F \rightarrow id \bullet\}) = I_5$, o qual já foi incluído

$I_{11} = goto(I_8, \text{')'}) = closure(\{F \rightarrow (E) \bullet\}) = \{F \rightarrow (E) \bullet\}$

$goto(I_8, \text{'+'}) = closure(\{E \rightarrow E + \bullet T\}) = I_6$, o qual já foi incluído

$goto(I_9, \text{'*'}) = closure(\{T \rightarrow T * \bullet F\}) = I_7$, o qual já foi incluído

Como resultado temos a seguinte tabela:

Conjunto Canônicos de Itens LR(0)	
Conjunto	Itens
I_0	$E' \rightarrow \bullet E, E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id$
$I_1 = goto(I_0, E)$	$E' \rightarrow E \bullet, E \rightarrow E \bullet + T$
$I_2 = goto(I_0, T)$	$E \rightarrow T \bullet, T \rightarrow T \bullet * F$
$I_3 = goto(I_0, F)$	$T \rightarrow F \bullet$
$I_4 = goto(I_0, \text{'('})$	$F \rightarrow (\bullet E), E \rightarrow \bullet E + T, E \rightarrow \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id$
$I_5 = goto(I_0, id)$	$F \rightarrow id \bullet$
$I_6 = goto(I_1, \text{'+'})$	$E \rightarrow E + \bullet T, T \rightarrow \bullet T * F, T \rightarrow \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id$
$I_8 = goto(I_4, E)$ $I_9 = goto(I_6, T)$	$T \rightarrow T * \bullet F, F \rightarrow \bullet (E), F \rightarrow \bullet id$ $F \rightarrow (E \bullet), E \rightarrow E \bullet + T$
I_9	$E \rightarrow E + T \bullet, T \rightarrow T \bullet * F$
$I_{10} = goto(I_7, F)$	$T \rightarrow T * F \bullet$
$I_{11} = goto(I_8, \text{')'})$	$F \rightarrow (E) \bullet$

Construção da Tabela de Parsing SLR

Dada uma gramática G, obtém-se G', aumentando G com a produção $S' \rightarrow S$, onde S é o símbolo de partida de G. A partir de G', determina-se o ***conjunto canônico de itens LR(0)***. Finalmente, constrói-se as tabelas ***ação*** e ***desvio***, conforme indicado pelo algoritmo abaixo.

1. Construa o conjunto canônico $C = \{I_0, I_1, \ldots, I_n\}$;
2. Cada linha da tabela corresponde a um estado i do analisador, o qual é construído a partir de I_i, $0 \leq i \leq n$;
3. A linha para da tabela ***ação*** para o estado i é determinada pelas regras abaixo. Entretanto, se houver algum conflito na aplicação destas regras, então a gramática não é ***SLR(1)***, e o algoritmo falha:
   a) Se $A \rightarrow \alpha \bullet a\beta$ está em I_i e $goto(I_i, a) = I_j$, $a \in V_T$, então $ação[i, a] = empilhar\ j$;
   b) Se $A \rightarrow \alpha \bullet$ está em I_i, $A \neq S'$, então $ação[i, a] = reduzir\ A \rightarrow \alpha$, para todo $a \in Seguinte(A)$;
   c) Se $S' \rightarrow S \bullet$ está em I_i, então defina $ação[i, \$] = aceita$.
4. A linha da tabela ***desvio*** para o estado i é determinada do seguinte modo:
   a) Para todos os não terminais A, se $goto(I_i, A) = I_j$, então $desvio[i, A] = j$.
5. As entradas não definidas são erros;
6. O estado de partida é o estado 0.

Pelo algoritmo, a tabela sintática para a gramática de expressões aumentada G, definida anteriormente, é:

	Ação						Desvio		
Estado	***id***	**+**	*****	**(**	**)**	**$**	*E*	*T*	*F*
0	***s5***			***s4***			1	2	3
1		***s6***				***aceita***			
2		***r2***	***s7***		***r2***	***r2***			
3		***r4***	***r4***		***r4***	***r4***			
4	***s5***			***s4***			8	2	3
5		***r6***	***r6***		***r6***	***r6***			
6	***s5***			***s4***				9	3
7	***s5***			***s4***					10
8		***s6***			***s11***				
9		***r1***	***s7***		***r1***	***r1***			
10		***r3***	***r3***		***r3***	***r3***			
11		***r5***	***r5***		***r5***	***r5***			