MC910 - Implementação de Linguagens de Programação
Análise Sintática - Parte II

1 Conceitos

Antes de prosseguirmos, devemos lembrar algumas definições, e introduzir outras, novas:

$\psi(X)$ O conjunto de todos os símbolos terminais que podem estar *no início* de uma forma sentencial derivada de X é denotado "$\psi(X)$"

$\Delta(X)$ O conjunto de todos os símbolos terminais que podem *seguir* uma forma sentencial derivada de X é denotado "$\Delta(X)$"

Redutendo Se $A \Rightarrow \alpha$, então α é um redutendo (a redução é o contrário da derivação).

Frase Se $A ::= \alpha B \gamma$, e $B \stackrel{+}{\Rightarrow} \beta$, dizemos que β é uma frase da forma sentencial $\alpha\beta\gamma$ para o não-terminal B. Isto significa que, ao encontrarmos $\alpha\beta\gamma$, podemos reduzir até obter $\alpha B\beta$.

Frase Simples Se β é uma frase simples para B, como dito anteriormente, e se $B \Rightarrow \beta$ (i. e., se existe uma derivação direta), então dizemos que β é frase simples.

Item Uma produção com uma marca em algum lugar do lado direito é um *item*. Exemplos:
$A ::= x \bullet yz$
$E ::= EO \bullet E$

2 Funcionamento da Análise Ascendente LR(0)

Começaremos com a Análise Ascendente LR(0). O nome "LR(0)" significa "Left-to-right parsing" (lemos a entrada da esquerda para a direita), "Rightmost derivation" (derivação direita é conseguida), "0 lookahead" (não precisamos olhar à frente na entrada para saber o que está por vir antes de tomar uma decisão).

Um algoritmo de análise sintática ascendente lerá a entrada, da esquerda para a direita e aplicará uma redução quando isto for possível.

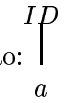
Assim, supondo que temos a seguinte gramática:

$E ::= EOE$
$E ::= ID$
$O ::= +|-$
$ID ::= a|b$

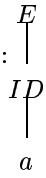
Como aconteceria a análise da sentença $a + b$?

Ao lermos a, nossa árvore tem um só nó: a

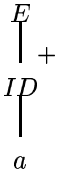
Podemos reduzir a para ID (usando a quinta regra da gramática), então:



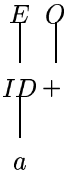
Agora reduzimos ID para E:



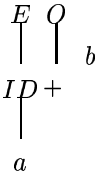
Agora, lemos "+". A nossa árvore parcial fica assim:



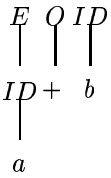
mos um "+", que pode ser reduzido para "O":



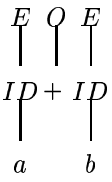
Agora, lemos b:



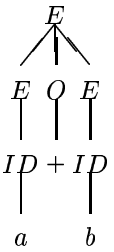
Novamente usamos a regra número cinco:



Usando a regra número dois, reduzimos ID para E:



E agora temos EOE, e sabemos que podemos reduzir isto para E:



E temos a árvore de derivação sintática da sentença.

Assim, se na nossa gramática tivermos uma regra $C ::= xyz$ e nossa cadeia de entrada for $xyz - abc$, após ter lido x, y, e z, teremos algo assim:

Parte lida: xyz
Parte restante: $-abc$
A cadeia é: $xyz - abc$

Mas como sabemos que $C ::= xyz$, aplicamos a redução e ficamos com:

Parte lida: C
Parte restante: $-abc$
A cadeia é: $C - abc$

E continuaremos assim até só restar o símbolo incial da gramática (ao contrário da análise descendente, onde começamos com o símbolo inicial e derivamos até que tivéssemos terminais em todas as folhas da árvore).

Note que, em uma forma sentencial direita, o redutendo será sempre a frase simples mais à esquerda.

Para saber onde já estamos, usaremos os itens: $C ::= xy \bullet z$ significa que lemos x, y, e poderíamos ler z em seguida.

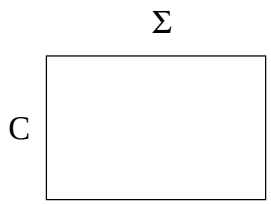
Mas se tivermos algo como $D ::= xy - ab$, onde estaríamos após ler x e y? Estaríamos tanto em $C ::= xy \bullet z$ como em $D ::= xy \bullet -ab$, e representaremos isto dizendo que existe um "estado" em que o algoritmo pode se encontrar, em que ele já leu xy, e que poderá ler tanto um $-$ como um z.

O estado seria:

$C ::= xy \bullet z$
$D ::= xy \bullet -ab$

O algoritmo poderá então se comportar como um autômato: usando uma tabela de estados $\times$ símbolos, verifica em que estado está (na coluna à esquer-

da), e qual o caracter lido da entrada (na linha superior), e baseado nisso tima uma decisão (que está na posição (i, j) da tabela, caso esteja no i-ésimo estado e o símbolo seja o j-ésimo). A tabela será da seguinte forma:



Onde C é o conjunto de estados, cada um representando em que pontos poderíamos estar, e quais as possíveis alternativas para o próximo símbololido – e para onde cada um nos levaria. Σ é o conjunto de símbolos da gramática.

As ações que o algoritmo poderá tomar (que serão as entradas na tabela) são:

- Ir para um outro estado – para isto simplesmente empilhamos o novo estado
- Aplicar uma redução usando uma determinada regra da gramática – paraisto, devemos desempihar n elementos (n é o tamanho do lado direito da regra); depois, verificamos quem ficou no fundo da pilha, e verificamos na tabela qual deve ser o novo estado; tabela(topo_da_pilha, X). X é o lado esquerdo da regra. Empilhamos este novo estado, então. Ao fazer a redução, modificamos a árvore sintática.
- Aceitar a cadeia
- Rejeitar a cadeia (um erro sintático ocorreu)

Veremos um exemplo a seguir. Não se preocupe por enquanto em saber como a tabela e os estados foram gerados; apenas tente entender como o algoritmo funciona, e mais à frente veremos como é a construção da tabela e estados.

Suponha que tenhamos a seguinte gramática:

1 $S' ::= S\#$
2 $S ::= EOE$
3 $E ::= a$
4 $O ::= *$

Os estados são:

e_0 $S' ::= \bullet S\#$
$S ::= \bullet EOE$
$E ::= \bullet a$

e_1 $S' ::= S \bullet \#$

e_2 $S ::= E \bullet OE$
$O ::= \bullet *$

e_3 $E ::= a\bullet$

e_4 $S ::= EO \bullet E$
$E ::= \bullet a$

e_5 $O ::= *\bullet$

e_6 $S ::= EOE\bullet$

E a seguinte tabela de análise:

	S	E	O	a	*	#
e_0	e_1	e_2		e_3		
e_1						a
e_2			e_4		e_5	
e_3				r_3	r_3	r_3
e_4		e_6		e_3		
e_5				r_4	r_4	r_4
e_6				r_2	r_2	r_2

A seguir faremos a análise da sentença "a*a" (a única aceita pela gramática). A cada passo, mostraremos a porção restante da cadeia de entrada, a pilha, o significado dos estados na pilha, e uma pilha "ilustrada". Esta pilha ilustrada mostrará, além de cada estado, quais foram os símbolos que causaram o empilhamento dos estados. Lendo a pilha do fundo para o topo (da esquerda para a direita, aqui), você poderá ver quanto da cadeia o algoritmo já leu, e de que forma ele "entendeu" a cadeia (que não-terminais ele escolheu para s reduções). por exemplo, a pilha e_0 E e_2 O e_4 a e_3 significa que depois do estado e_0, foi lida alguma cadeia que o algoritmo reduziu para E, e com isto foi para e_2. Dalí, o algoritmo leu algo que foi reduzido para O, e foi para o estado e_4. Em seguida, leu um terminal a, e foi para o estado e_3. Mostraremos também como a árvore de derivação vai sendo construída a cada passo do algoritmo.

Entrada (restante): $a * a\#$
Pilha: e_0
Significado: Estamos em e_0, o que portanto ainda não lemos nada.
Árvore: vazia

Símbolo lido: "a". Olhando na tabela, vemos que quando estamos no estado 0 e lemos um "a", devemos ir para o estado 3. Empilhamos o estado 3, então.

Entrada (restante): $*a\#$
Pilha: e_0, e_3
Pilha Ilustrada: e_0 a e_3
Significado: e_3 significa que lemos um "a", e que devemos reduzir em seguida.
Árvore: a

Agora, como no estado 3 só temos ações de "redução" (veja a tabela), reduziremos, sem ler nada da entrada. Usaremos a produção número 3. Para isto, desempilhamos *um* estado da pilha (porque o lado direito da produção 3 tem um símbolo apenas). Olhamos agora para pilha, e ela nos diz "e_0". Olhamos para o lado *direito* da produção 3, e vemos o símbol o E. Assim, vemos na tabela que (e_0, E) nos leva para e_2. Empilhamos e_2, então:

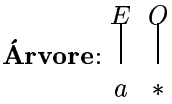
Entrada (restante): $*a\#$
Pilha: e_0, e_2
Pilha Ilustrada: e_0 E e_2
Significado: e_2 significa que um "E" foi lido.
Árvore:
E
|
a

Agora, lemos o símbolo "*", e vemos que, estando em e_2 e lendo este símbolo, devemos ir para e_5. Empilhamos e_5, então:

Entrada (restante): $a\#$
Pilha: e_0, e_2, e_5
Pilha Ilustrada: e_0 E e_2 * $_5$
Significado: a pilha indica que depois de começarmos, lemos um E (representado pela presença do e_2, e depois um $*$ (porque temos o e_5 ali).
Árvore:
E
| $*$
a

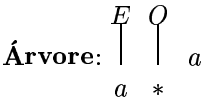
E no estado e_5, devemos reduzir! Usaremos a produção 4. Ao desempilhar um elemento (veja na produção 4 quantos símbolos estão à direita), sobra e_2 no topo da pilha. A produção 4 tem "O" à direita, assim verificamos (e_2, O) na tabela, e descobrimos que devemos ir para o estado e_4.

Entrada (restante): $a\#$
Pilha: e_0, e_2, e_4
Pilha Ilustrada: e_0 E e_2 O e_4
Significado: Depois de ler um E, lemos também um O (e_4).

Árvore: 

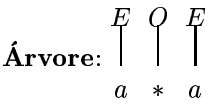
Agora, lemos um "a". Na tabela, $(e_4, a) = e_3$, então empilhamos o estado e_3:

Entrada (restante): #
Pilha: e_0, e_2, e_4, e_3
Pilha Ilustrada: e_0 E e_2 O e_4 a e_3
Significado: Depois de EO, lemos um a.

Árvore: 

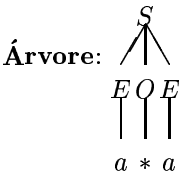
Agora devemos reduzir, usando a produção 3. Desempilhando e_3, veremos e_4 no topo da pilha, e então verificamos (e_4, E), que é e_6. Empilhamos e_6:

Entrada (restante): #
Pilha: e_0, e_2, e_4, e_6
Pilha Ilustrada: e_0 E e_2 O e_4 E e_6
Significado: Depois de EO, temos um E, mas não da mesma forma como no segundo passo, onde tínhamos o estado e_2. Agora sabemos que temos um O atrás deste E (veja a diferença entre os estados e_2 e e_6)!

Árvore: 

No estado 6, devemos reduzir usando a produção 2. Veja que a produção 2 tem *três* símbolos do lado direito. Assim, desempilhamos três elementos da pilha, e sobra e_0 no topo. Assim, vemos para onde (e_0, S) nos leva (porque S está na parte esquerda da produção 2). É para e_1.

Entrada (restante): #
Pilha: $e_0 e_1$
Pilha Ilustrada: e_0 S e_1
Significado: Acabamos de ler um S.

Árvore: 

E em e_1, como vemos agora o final do arquivo (veja $(e_1, \#)$ na tabela), aceitamos a cadeia.

2.1 Construindo Estados e Tabela de Análise

Para construir os estados e a tabela de análise, precisaremos de algumas funções:

Primeiro, o *fecho* de um conjunto de itens é o mesmo conjunto, aumentaddo com alguns itens. para cada item $A ::= X \bullet \beta$ no estado, queremos incluir também os itens $Y ::= \bullet\gamma$, para todo $Y \in \psi(\alpha)$. Isto porque, se queremos saber "onde estamos", devemos considerar tanto o não-terminal como tudo o que é derivado dele:

Se em uma gramática temos as regras $E ::= ID|(EOE)$ e $ID ::= a|b$, o item $E ::= (EO \bullet E)$ significa que lemos um abre parêntesis, um E (provavelmente uma expressão), um O, e agora estamos apra ler um E novamente. Mas um E pode derivar várias coisas. Se estamos para ler um E, devemos incluir no fecho todas as possibilidades para começarmos a ler um E, e o fecho ficaria:

$E ::= (EO \bullet E)$
$E ::= \bullet ID$
$E ::= \bullet(EOE)$
$ID ::= \bullet a$
$ID ::= \bullet b$

Algoritmo 1 Fecho (e_i)

for itens $A ::= \alpha \bullet X\beta \in e_i$ **do**
  for produções $X ::= \gamma$ **do**
    $e_i \leftarrow e_i \cup \{X ::= \bullet\gamma\}$
  end for
end for

Algoritmo 2 Transfere (e_i, k)

$J \leftarrow \emptyset$
for itens $A ::= \alpha \bullet X\beta \in e_i$ **do**
  $J \leftarrow J \cup \{A ::= \alpha X \bullet \beta\}$
end for
$J \leftarrow$ Fecho (J)

Agora, a função *transfere* nos diz, dado um estado e um símbolo lido, para que outro estado deveremos ir. Do estado anterior, ao ler um a, deveremos ir para o estado que só contémum item, "$ID ::= a\bullet$" (indicando que acabamos de ler um a, e que como terminamos de ler o lado direito de uma produção, provavelmente deveríamos reduzir.

Dissemos que o algoritmo de análise ascendente funciona como um autômato. Os estados (fechos de conjuntos de itens) serão os estados do autômato. A função *transfere* nos dará o conjunto de arestas do autômato (que usamos na forma de tabela no último exemplo de análise.

Agora veremos como, a partir de uma gramática, montar o conjunto de estados e a tabela de análise LR(0) pra a gramática.

Primeiro, se nossa gramática começa com $S ::= \gamma$ (o símbolo inicial é S, deveremos aumentá-la com o símbolo S':

$S' ::= S\#$
$S ::= \gamma$

Vamos supor então que nossa gramática seja:

1 $S' ::= S\#$
2 $S ::= A, B$
3 $A ::= x$
4 $A ::= y$
5 $B ::= (A)$
6 $B ::= a$
7 $B ::= b$

Agora, o primeiro item que vamos usar é $S' ::= \bullet S\#$ (não lemos nada ainda, e estamos para ler um "S", que é uma sentença da gramática). O fecho deste item será nosso primeiro estado:

$S' ::= \bullet S\#$
$S ::= \bullet A, B \quad A ::= \bullet x$
$A ::= \bullet y$

Este é o nosso estado e_0. Agora que já sabemos de onde começaremos, deveremos descobrir para que outros estados iremos se lermos algum dos símbolos da gramática. Para isso usaremos a função *transfere*. Do estado inicial, poderíamos ler A, x ou y. Assim, teremos três novos estados:

$e_1 = \text{transfere}(e_0, A)$
$e_2 = \text{transfere}(e_0, x)$
$e_3 = \text{transfere}(e_0, y)$

Aproveitaremos e incluiremos na nossa tabela de análise as entradas que nos dizem que, do estado e_0, lendo estes símbolos, iremos para estes estados:

$(e_0, A) = e_1$
$(e_0, x) = e_2$
$(e_0, y) = e_3$

Depois, olharemos para estes novos estados e verificaremos se deles ainda podemos ir para outros estados (se podemos mover o símbolo $\bullet$ para frente). Repetiremos isto até não houver mais estados de onde possamos ler símbolos.

Quando encontrarmos um estado cujos items possuem o símbolo $\bullet$ esteja no final da produção (itens *completos*), incluiremos na tabela, naquele estado, entradas r_k, onde k é o número da regra que aquele item representa. Isto significa que naquele momento, devemos aplicar uma redução. Preencheremos todas as entradas naquela linha, exceto as dos não-terminais.

O algoritmo 3 determina todos os estados e preenche a tabela de análise.

Algoritmo 3 Monta Tabela LR(0)

$e_0 \leftarrow fecho(S' ::= \bullet S\#)$
for estados e_i **do**
  if e_i é completo **then**
    $(e_i, j) \leftarrow$"r_k" (k = n^o da regra)
  else
    for itens $A ::= \alpha \bullet X\beta$ em e_i **do**
      $e_j \leftarrow transfere(e_i, X)$
      $(e_i, X) \leftarrow e_j$
    end for
  end if
end for

Prosseguindo assim, os estados e a tabela de transição para nossa gramática serão:

e_0
$S' ::= \bullet S\#$
$S ::= \bullet A, B$
$A ::= \bullet x$
$A ::= \bullet y$

$e_1 = transfere(e_0, A)$
$S ::= A\bullet, B$

$e_2 = transfere(e_0, x)$
$A ::= x \bullet$

$e_3 = transfere(e_0, y)$
$A ::= y \bullet$

$e_4 = transfere(e_1, \text{“,”})$
$S ::= A, \bullet B$
$B ::= \bullet(A)$
$B ::= \bullet a$
$B ::= \bullet b$

$e_5 = transfere(e_4, \text{“(”})$
$B ::= (\bullet A)$
$A ::= \bullet x$
$A ::= \bullet y$

$e_6 = transfere(e_5, A)$
$B ::= (A\bullet)$

$e_7 = transfere(e_6, \text{“)”})$
$B ::= (A)\bullet$

$e_8 = transfere(e_0, S)$
$S' ::= S \bullet \#$

$e_9 = transfere(e_4, B)$
$S ::= A, B \bullet$

$e_{10} = transfere(e_4, a)$
$B ::= a \bullet$

$e_{11} = transfere(e_4, b)$
$B ::= b \bullet$

Agora, a tabela:

	S	A	B	,	x	y	(	)	a	b	#
e_0	e_8	e_1			e_2	e_3					
e_1				e_4							
e_2				r_3	r_3	r_3	r_3	r_3	r_3	r_3	r_3
e_3				r_4	r_4	r_4	r_4	r_4	r_4	r_4	r_4
e_4			e_9				e_5		e_{10}	e_{11}	
e_5		e_6			e_2	e_3					
e_6								e_7			
e_7				r_5	r_5	r_5	r_5	r_5	r_5	r_5	r_5
e_8											a
e_9				r_2	r_2	r_2	r_2	r_2	r_2	r_2	r_2
e_{10}				r_6	r_6	r_6	r_6	r_6	r_6	r_6	r_6
e_{11}				r_7	r_7	r_7	r_7	r_7	r_7	r_7	r_7

Já vimos como a análise ascendente é feita; o algoritmo 4 é uma descrição formal.

Algoritmo 4 Análise LR(0)

empilha e_0
repeat
  le(prox)
  if tabela (topo_da_pilha, prox) = a **then**
    ACEITE
  else if tabela (topo_da_pilha, prox) = e_i **then**
    empilhe e_i
  else if tabela (topo_da_pilha, prox) = r_k **then**
    na árvore, reduza usando a regra k da gramática
    desempilhe n elementos {n é tamanho do lado direito de k}
    empilhe tabela (topo_da_pilha, X) {X é o lado esquerdo de k }
  else
    REJEITE
  end if
until cadeia aceita ou rejeitada

Verifique no exemplo de análise ascendente anterior os estados e a tabela. Você fixará melhor estas idéias se pegar uma pequena gramática, construir a tabela de análise, e depois fizer a análise de algumas sentenças.

3 Análise SLR

Considere a seguinte gramática:

0 $E' ::= E\#$
1 $E ::= E + T$
2 $E :; = T$
3 $T ::= T * F$
4 $T ::= F$
5 $F ::= (E)$
6 $F ::= a$

Tentaremos montar uma tabela LR(0). Estes são os estados:

e_0
  $E' ::= \bullet E\#$
  $E ::= \bullet E + T | \bullet T$
  $T ::= \bullet T * F | \bullet F$
  $F ::= \bullet(E) | \bullet a$

$e_1 = transfere(e_0, E)$
  $E' ::= E \bullet \#$

$$E ::= E \bullet + T$$

$$\begin{aligned} e_2 &= transfere(e_0, T) \\ E &::= T \bullet \\ T &::= T \bullet * F \end{aligned}$$

$$\begin{aligned} e_3 &= transfere(e_0, F) \\ T &::= F \bullet \end{aligned}$$

$$\begin{aligned} e_4 &= transfere(e_0, \text{“(”}) \\ E &::= (\bullet E) \\ E &::= \bullet E + T | \bullet T \\ T &::= \bullet T * F | \bullet F \\ F &::= \bullet (E) | \bullet a \end{aligned}$$

$$\begin{aligned} e_5 &= transfere(e_0, a) \\ F &::= a \bullet \end{aligned}$$

$$\begin{aligned} e_6 &= transfere(e_1, +) \\ E &::= E + \bullet T \\ T &::= \bullet T * F | \bullet F \\ F &::= \bullet (E) | \bullet a \end{aligned}$$

$$\begin{aligned} e_7 &= transfere(e_2, *) \\ T &::= T * \bullet F \\ F &::= \bullet (E) | \bullet a \end{aligned}$$

$$\begin{aligned} e_8 &= transfere(e_4, E) \\ F &::= (E \bullet) \\ E &::= E \bullet + T \end{aligned}$$

$$\begin{aligned} e_9 &= transfere(e_6, T) \\ E &::= E + T \bullet \\ T &::= T \bullet * F \end{aligned}$$

$$\begin{aligned} e_{10} &= transfere(e_7, F) \\ T &::= T * F \bullet \end{aligned}$$

$e_{11} = transfere(e_8, ")")$
$F ::= (E)\bullet$

Note que nos estados e_2 e e_9 existem itens completos e incompletos. Nestes estados, devemos reduzir ou deslocar? Por exemplo, no estado e_9, devemos reduzir $E + T$ para T, ou devemos deslocar T para a pilha e esperar um $*$ em seguida? No estado e_2, reduzimos T para E ou deslocamos T para a pilha e esperamos um $*$?

Ao tentarmos montar a tabela, haveria entradas onde teríamos que colocar duas ações diferentes a serem tomadas. Isto é uma característica desta gramática: ela não é uma gramática LR(0).

Há um a maneira simples de resolver o problema: veja que no estado e_2. Este estado significa que foi lida uma cadeia, que foi reduzida a T. Não sabemos se este T é a primeira parte de um redutendo $T * F$ ou se deve ser reduzido sozinho pra E. Mas nesta gramática, veja que $\Delta(E) = \{+,), \#\}$ e portanto, um símbolo $*$ nunca deveria aparecer na frente de um E. Assim, ao preenchermos a tabela, faremos o seguinte: se o símbolo lido a seguir for $*$, deslocamos T para a pilha (porque ele é parte de um redutendo que ainda não foi completamente lido). Caso contrário, reduzimos este T para um E.

Podemos fazer isto preenchendo apenas as colunas de $\Delta(E)$ com r_2 (em outros casos não devemos reduzir mesmo, porque o E resultante não poderia ser seguido dos outros símbolos).

Não há mais conflito, porque $*$ não pertence a $\Delta(E)$. Fazemos o mesmo com o estado e_9; na verdade, podemos construir a tabela preenchendo apenas as colunas em $\Delta(X)$ com reduções, sempre que tivermos um item completo $X ::= \gamma\bullet$.

Veja como fica a tabela, construída desta maneira:

	E	T	F	a	+	*	(	)	#
e_0	e_1	e_2	e_3	e_5			e_4		
e_1					e_6				a
e_2					r_2	e_7		r_2	r_2
e_3					r_4	r_4		r_4	r_4
e_4	e_8	e_2	e_3	e_5			e_4		
e_5		e_6			r_6	r_6		r_6	r_6
e_6		e_9	e_3	e_5			e_4		
e_7			e_10	e_5			e_4		
e_8					e_6			e_11	
e_9					r_1	e_7		r_1	r_1
e_{10}					r_3	r_3		r_3	r_3
e_{11}					r_5	r_5		r_5	r_5

Esta é uma tabela de análise SLR(1). Uma gramática que permite a construção de tabela de análise desta forma é chamada gramática SLR(1).