Capítulo 4

Análise Sintáctica Descendente

Os automátos finitos apresentados no capítulo anterior são suficientes para tratar os elementos léxicos de uma linguagem de programação, o tratamento da estrutura sintáctica de uma linguagem de programação está, em geral, fora do alcance de tais métodos, sendo necessário construir reconhecedores para linguagens independentes do contexto mas não necessariamente regulares.

Neste capítulo vamos estudar uma classe de reconhecedores para linguagens independentes do contexto, os reconhecedores descendentes, sendo que no próximo capítulo tratar-se-á dos reconhecedores ascendentes.

4.1 Reconhecimento Descendente de Linguagens Não-Regulares

Um reconhecedor descendente efectua o reconhecimento de uma frase construindo uma árvore de derivação partindo da raiz e terminando nas folhas, criando os nós da árvore segundo uma travessia descendente da esquerda para a direita. Este processo pode também ser visto, para este tipo de linguagens, como a derivação pela esquerda de uma dada frase.

Vejamos um exemplo:

Exemplo 4.1 *Considere-se a seguinte gramática:*

$$G = (\{a, b, c\}, \{I, A, B\}, I, P)$$

com P o seguinte conjunto de produções:

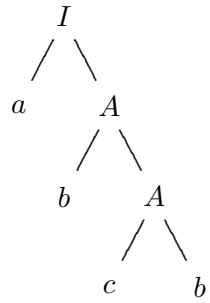
$$I \rightarrow aA \mid B$$

[1](Versão 1.8)

$$\begin{aligned} A &\rightarrow cb \mid bA \\ B &\rightarrow a \mid aB \end{aligned}$$

e a frase ≪abcb≫.

Esta frase é reconhecida com sucesso através da seguinte árvore de derivação:



A frase reconhecida (ou a reconhecer) é dada pelas folhas da árvore, sendo que os nós interiores vão ser ocupados por não-terminais.

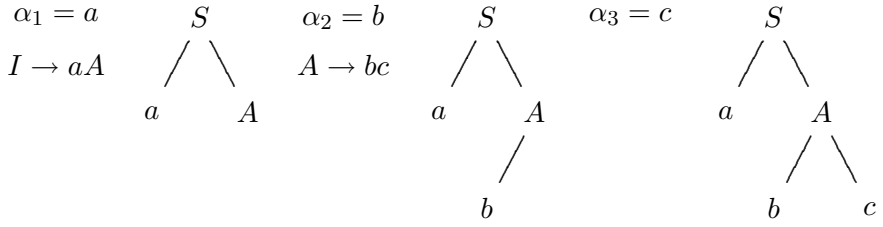
Definição 4.1 (Árvore de Derivação) *Seja $G = (T, N, I, P)$ uma gramática independente do contexto. Se $A \rightarrow \alpha_1 \ldots \alpha_n \in P$, então existe uma árvore de derivação cuja raiz é A e cujas sub-árvores são $S_1, \ldots, S_n$ (nesta ordem) em que:*

- *S_i consiste num único nó associado ao símbolo α_i, com $\alpha_i \in T$;*
- *S_i é uma árvore de derivação cuja raiz é α_i, com $\alpha_i \in N$.*

Exemplo 4.2 *Considere-se a seguinte gramática:*

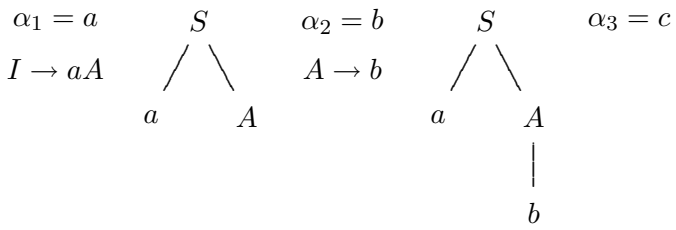
$$G = (\{a, b, c\}, \{I, A, B\}, I, \{I \rightarrow aA, I \rightarrow B, A \rightarrow bc, A \rightarrow b, B \rightarrow bc\})$$

e seja ≪abc≫ a frase a reconhecer. Temos que a aplicação das diferentes regras de produção vai-nos permitir construir a seguinte árvore de derivação:



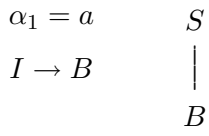
A fronteira coincide com a frase a reconhecer, temos então um reconhecimento feito com sucesso.

No entanto, e para este mesmo exemplo, podemos gerar as árvores.



No qual o reconhecimento não é feito com sucesso, a árvore já está «fechada» e ainda há símbolos na frase a reconhecer.

Ou então:



No qual o reconhecimento também não é feito com sucesso, a árvore não está fechada mas chegou-se a um ponto em que não é possível continuar a sua construção.

Temos então que um reconhecedor capaz de lidar com estas situações teria de ser capaz de recuar sobre decisões erradas para tentar outras soluções até esgotar todas as soluções, ou até ser bem sucedido numa delas. Na prática não se vão implementar reconhecedores com recuo («backtracking») dado não serem eficientes.

4.2 Gramáticas LL(1)

Para garantir que nunca surgem ambiguidades na escolha da produção a utilizar empregam-se gramáticas em que todas as produções da forma:

$$A \to \alpha_1 \mid \alpha_2 \mid \ \ldots \ \mid \alpha_n$$

derivam frases cujos símbolos iniciais são todos diferentes. Ou seja tem-se um conjunto de produções para as quais basta «olhar» para o primeiro símbolo para saber qual é a escolha correcta.

No entanto podem surgir situações em que não basta conhecer os primeiros símbolos de cada uma das produções do tipo referido acima para evitar situações ambíguas. É esse o caso quando se tem que um dado símbolo A (primeiro símbolo de uma dada produção) é um símbolo anulável, isto é um símbolo a partir do qual se deriva a frase nula, formalmente:

Definição 4.2 (Símbolo Anulável) *Um símbolo A não terminal diz-se anulável se existe*

$$A \stackrel{*}{\Rightarrow} \epsilon$$

Então, nestes casos é necessário saber qual o símbolo que vêm a seguir a A (símbolo anulável) para deste modo poder evitar situações ambíguas.

As gramáticas livres do contexto que evitam estes dois problemas designam-se por gramáticas $LL(1)$, de:

L	≪Left scan≫	leitura da esquerda para a direita
L	≪Leftmost derivation≫	derivação pela esquerda
(1)		Um símbolo antecipável

Para podermos definir as condições que uma gramática independente do contexto tem de verificar para ser uma gramática $LL(1)$ é necessário definir antes algumas funções auxiliares. São elas:

First: define o conjunto de símbolos que iniciam derivações a partir de uma sequência da símbolos terminais e não-terminais.

First$_N$ define o conjunto de símbolos que iniciam derivações a partir de um símbolo não-terminal.

Follow define o conjunto de símbolos que se podem seguir a derivar por um dado símbolo não terminal.

Definição 4.3 (First) *A função ≪First≫ define-se como:*

$$\begin{array}{rcccc} First & : & (N \cup T)^* & \longrightarrow & 2^T \\ & & \alpha & \longmapsto & First(\alpha) \end{array}$$

tal que, para todo o $a \in T$ *e* $\alpha' \in (N \cup T)^*$ *tem-se:*

$$First(\alpha) = \begin{cases} \emptyset, & \alpha = \epsilon \\ \{a\}, & \alpha = a\alpha' \\ \bigcup\limits_{1 \le i \le n} First(\beta_i), & \alpha = A\alpha' \text{ e } A \to \beta_1 \mid \ldots \mid \beta_n \text{ e } \beta_i \not\stackrel{*}{\Rightarrow} \epsilon \text{ para todo o } 1 \le i \le n \\ First(\alpha') \cup \bigcup\limits_{1 \le i \le n} First(\beta_i), & \alpha = A\alpha' \text{ e } A \to \beta_1 \mid \ldots \mid \beta_n \text{ e } \beta_i \stackrel{*}{\Rightarrow} \epsilon \text{ para um dado } 1 \le i \le n \end{cases}$$

Exemplo 4.3 *Para a gramática,*

$$G = (\{a,b,c\}, \{I,A,B\}, I, \{I \to aBa \mid BAc \mid ABc;\ A \to aA \mid \epsilon;\ B \to ba \mid c\})$$

tem-se:

$$\begin{array}{rcll} First(aBa) & = & \{a\} & \\ First(BAc) & = & First(ba) \cup First(c) & (B \not\stackrel{*}{\Rightarrow} \epsilon) \\ & = & \{b\} \cup \{c\} & \\ & = & \{b,c\} & \\ First(ABc) & = & First(aA) \cup First(\epsilon) \cup First(Bc) & (A \stackrel{*}{\Rightarrow} \epsilon) \\ & = & \{a\} \cup \emptyset \cup First(ba) \cup First(c) & \\ & = & \{a\} \cup \{b\} \cup \{c\} & \\ & = & \{a,b,c\} & \end{array}$$

Definição 4.4 (First$_N$) *A função «$First_N$» define-se como:*

$$\begin{array}{rcccl} First_N & : & N & \longrightarrow & 2^T \\ & & X & \longmapsto & First_N(X) = \bigcup\limits_{(X \to \alpha_i)} First(\alpha_i) \end{array}$$

Exemplo 4.4 *Usando a gramática G definida no exemplo anterior temos:*

$$\begin{array}{rcl} First_N(I) & = & First(aBa) \cup First(BAc) \cup First(ABC) \\ & = & \{a\} \cup \{b,c\} \cup \{a,b,c\} \\ & = & \{a,b,c\} \\ First_N(A) & = & First(aA) \cup First(\epsilon) \\ & = & \{a\} \\ First_N(B) & = & First(ba) \cup First(c) \\ & = & \{b,c\} \end{array}$$

Definição 4.5 (Follow) *A função «$Follow$» define-se como:*

$$\begin{array}{rcccl} Follow & : & N & \longrightarrow & 2^T \\ & & X & \longmapsto & Follow(X) \end{array}$$

tal que, para todo o $Y \in N$ e α,$\beta \in (N \cup T)^*$ tem-se:

$$Follow(X) = \bigcup_{(Y \to \alpha X \beta)} \left(First(\beta) \cup \begin{cases} \emptyset, & \text{se } X \stackrel{*}{\not\Rightarrow} \epsilon \\ Follow(Y), & \text{se } X \stackrel{*}{\Rightarrow} \epsilon \end{cases} \right)$$

Exemplo 4.5 *Usando a gramática G definida no exemplo anterior temos:*

$$\begin{array}{rcl} Follow(I) & = & \emptyset \\ Follow(A) & = & First(c) \cup First(Bc) \cup (First(\epsilon) \cup Follow(A)) \\ & = & \{b,c\} \\ Follow(B) & = & First(a) \cup First(Ac) \cup First(c) \\ & = & \{a\} \cup \{a,c\} \cup \{c\} \\ & = & \{a,c\} \end{array}$$

Podemos agora definir a condição $LL(1)$.

Definição 4.6 (Condição LL(1)) *Uma gramática $G = (T, N, I, P)$ satisfaz a condição $LL(1)$ se, para toda a produção $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \in P$, se verificarem as seguintes condições:*

1. *Para $A \stackrel{*}{\not\Rightarrow} \epsilon$ então:*

$$First(\alpha_i) \cap First(\alpha_j) = \emptyset \qquad i \neq j, 1 \leq i,j \leq n$$

i.e. o conjunto de símbolos que iniciam frases derivadas de dois lados direitos de produções com o mesmo lado esquerdo devem ser disjuntos.

2. Para $A \stackrel{*}{\Rightarrow} \epsilon$ *então:*

$$\text{então} \quad First_N(A) \cap Follow(A) = \emptyset$$

i.e. no caso de símbolos anuláveis, o conjunto de símbolos que iniciam derivações a partir de A, tem de ser disjunto do conjunto de símbolos que se possam seguir a qualquer sequência gerada por A.

Exercício 4.1 *Dado a gramática G*

$$G = (\{a, b, c, d, e\}, \{I, A, B, C\}, I, P)$$

com:

$$P = \{I \to bAb \mid aBb, A \to aA \mid \epsilon, B \to cC \mid d, C \to e\}$$

verifique se G é $LL(1)$.

A condição $LL(1)$ pode-se reescrever através de uma outra função auxiliar que de alguma forma sintetiza as anteriores.

Definição 4.7 (Lookahead) *A função «Lookahead» define-se como:*

$$\begin{array}{rccl} Lookahead & : & P & \longrightarrow & 2^T \\ & & (A \to \alpha) & \longmapsto & First(\alpha) \cup \left\{ \begin{array}{ll} \emptyset, & \text{se } \alpha \stackrel{*}{\not\Rightarrow} \epsilon \\ Follow(A), & \text{se } \alpha \stackrel{*}{\Rightarrow} \epsilon \end{array} \right. \end{array}$$

Exemplo 4.6 *Para a gramática,*

$$G = (\{a, b, c\}, \{I, A, B\}, I, \{I \to aBa \mid BAc \mid ABc;\ A \to aA \mid \epsilon;\ B \to ba \mid c\})$$

tem-se:

$$\begin{array}{rcl} Lookahead(A \to aA) & = & First(aA) \cup \emptyset \\ & = & \{a\} \\ Lookahead(A \to \epsilon) & = & First(\epsilon) \cup Follow(A) \\ & = & \{a, b, c\} \end{array}$$

Usando a função «Lookahead» é então possível redefinir a condição $LL(1)$, sendo que ambas as variantes são equivalentes entre si.

Definição 4.8 (Condição LL(1)) *Uma gramática* $G = (T, N, I, P)$ *satisfaz a condição* $LL(1)$ *se para quaisquer produções* $A \to \alpha_1, A \to \alpha_2$ *se tenha:*

$$Lookahead(A \to \alpha_1) \cap Lookahead(A \to \alpha_2) = \emptyset$$

Exercício 4.2 *Dada a gramática* $G = (\{a, b, c, d, e\}, \{I, A, B, C\}, I, P)$ *com* P *o seguinte conjunto de produções:*

$$\begin{array}{lcl} I & \rightarrow & bAb \mid aBb \\ A & \rightarrow & aA \mid \epsilon \\ B & \rightarrow & cC \mid d \\ C & \rightarrow & e \end{array}$$

verifique (usando a função «Lookahead») que G *satisfaz a condição* $LL(1)$.

4.2.1 Transformações Essenciais à Condição LL(1)

Existem várias situações típicas aonde se verificam conflitos $LL(1)$, isto é, situações de não cumprimento da condição $LL(1)$.

1. Gramáticas ambíguas;
2. Gramáticas recursivas à esquerda.

Para cada uma destas duas situações existem transformações que podem ser feitas nas produções de forma a transforma uma dada gramática numa outra que verifique a condição $LL(1)$. Vejamos ambos os casos de seguida.

- Gramática ambíguas, gramáticas com produções com o mesmo lado esquerdo e cujos lados direitos têm o mesmo prefixo, por exemplo:

  Considere-se as seguintes produções:

  $$A \rightarrow \alpha\beta \mid \alpha\gamma$$

  existe um conflito $LL(1)$ dado que:

  $$Lookahead(A \rightarrow \alpha\beta) \cap Lookahead(A \rightarrow \alpha\gamma) \supseteq First(\alpha)$$

  a solução consiste em efectuar uma factorização à esquerda

  $$\begin{array}{lcl} A & \rightarrow & \alpha A' \\ A' & \rightarrow & \beta \mid \gamma \end{array}$$

  a correcção desta transformação demonstra-se através da álgebra de expressões regulares.

  $$\begin{array}{rcl} A \rightarrow \alpha\beta \mid \alpha\gamma & \Leftrightarrow & A = \alpha\beta + \alpha\gamma \\ & & A = \alpha(\beta + \gamma) \\ A \rightarrow \alpha A' & \Leftrightarrow & A = \alpha A' \\ A' \rightarrow \beta + \gamma & \Leftrightarrow & A' = \beta + \gamma \end{array}$$

- Gramática recursivas à esquerda.

  Considere-se as seguintes produções:

  $$A \rightarrow A\alpha \mid \beta$$

  temos que

  $$First_N(A) = First(A\alpha) \cup First(\beta)$$

  existe um conflito $LL(1)$ na medida que:

  $$Lookahead(A \rightarrow A\alpha) \cap Lookahead(A \rightarrow \beta) \supseteq First(\beta)$$

  A solução passa por eliminar a recursividade à esquerda reescrevendo a gramática com recursividade à direita.

  $$\begin{array}{rcl} A & \rightarrow & \beta A' \\ A' & \rightarrow & \alpha A' \mid \epsilon \end{array}$$

  novamente é fácil de demonstrar a correcção desta transformação.

  $$\begin{array}{rcl} A \rightarrow A\alpha \mid \beta & \Leftrightarrow & A = A\alpha + \beta \\ & & A = \beta + A\alpha \\ & & A = \beta\alpha^* \\ & & A = (\beta\alpha^*)\epsilon \\ & & A = \beta(\alpha^*\epsilon) \\ A \rightarrow \beta A' & \Leftrightarrow & A = \beta A' \\ A' \rightarrow \alpha A' | \epsilon & \Leftrightarrow & A' = \epsilon + \alpha A' \end{array}$$

Exercício 4.3 *Faça as transformações necessárias para que a gramática*

$$G = (\{x, y, z\}, \{I, X, Y\}, I, P)$$

com P o seguinte conjunto de produções:

$$\begin{array}{rcl} I & \rightarrow & zX \mid zY \\ X & \rightarrow & Xx \mid x \\ Y & \rightarrow & yY \mid \epsilon \end{array}$$

verifique a condição $LL(1)$.

Pode-se dizer em relação aos reconhecedores descendentes que:

- a sua implementação é simples;
- a transformação de uma gramática de forma a que ela passe a verificar a condição $LL(1)$ faz que:
  - seja necessário introduzir novos símbolos;

  - o reconhecimento de uma frase seja feito através de uma sequência de derivação com um maior número de passos;
  - a árvore de derivação que se obtém durante o reconhecimento não representa fielmente a sua estrutura sintáctica.

A implementação de um reconhecedor descendente pode ser feita utilizando duas estratégias distintas:

- Reconhecedor Recursivo Descendente;
- Reconhecedores Dirigidos por uma Tabela.

4.3 Reconhecedor Recursivo Descendente

Um reconhecedor recursivo descendente (RRD) de uma linguagem L definida pela gramática $G = (N, T, I, P)$ constrói-se associando a cada símbolo $X \in N$ um procedimento de reconhecimento. O reconhecedor para L obtém-se por combinação de todos os procedimentos.

A definição dos procedimentos pode ser feita através de duas aproximações distintas:

- escrever os procedimentos tendo por base os grafos de sintaxe da linguagem;
- escrever os procedimentos tendo por base as produções da gramática.

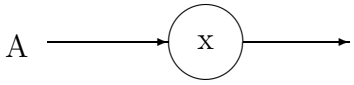
Vamos explorar somente a primeira destas duas aproximações possíveis.

4.3.1 Construção de um RRD a partir dos Grafos de Sintaxe

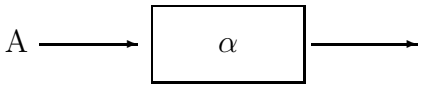
Como pré-condição é necessário verificar que a gramática verifica a condição $LL(1)$.

A transformação de um grafo de sintaxe num reconhecedor vai-se processar através da associação a cada estrutura básica de um grafo de sintaxe um procedimento de reconhecimento, com a posterior combinação dos diferentes procedimentos. Vai-se assumir que:

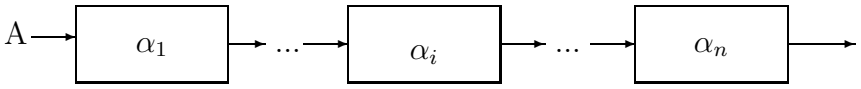
- os símbolos a reconhecer são caracteres, esta simplificação é plausível dado podermos ver o reconhecimento sintácticos como um segundo passo já após o reconhecimento léxico;
- existe uma variável global `simb` a qual contém o próximo símbolo a reconhecer;
- o avanço para o próximo símbolo é feito através da função `ler()`;
- o resultado do reconhecimento é dado por $T(\alpha)$.



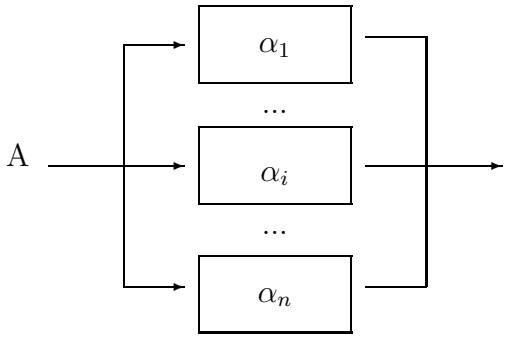

```
se simb==x então simb = ler()
senão erro
fimse
```



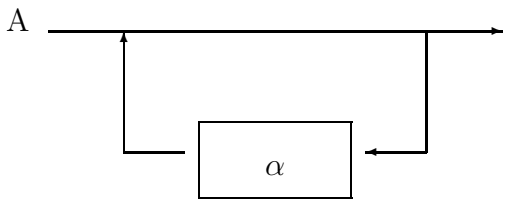
T(α);




```
T(α_1);
⋮
T(α_i);
⋮
T(α_n);
```

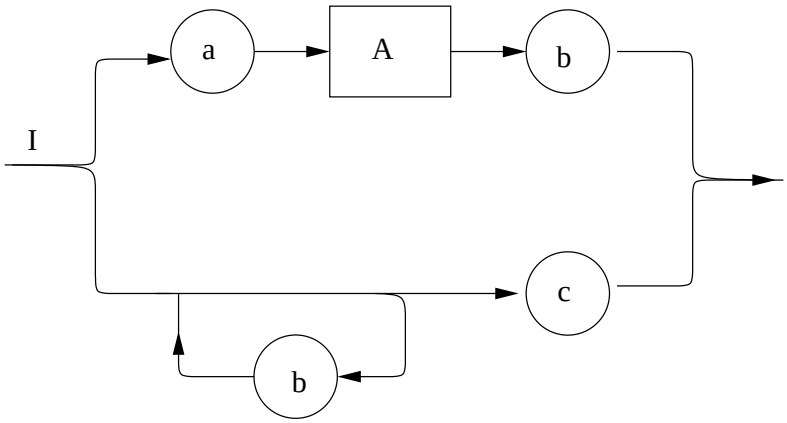



```
caso simb seja
   First(α_1) : T(α_1);
     ⋮
   First(α_i) : T(α_i);
     ⋮
   First(α_n) : T(α_n);
   noutros casos
     erro
fimcaso
```




```
enquanto simb ∈ First(α)
  T(α);
fimenquanto
```

Exercício 4.4 *Faça a construção de um reconhecedor para a linguagem L definida pelo seguinte grafo.*



Os reconhecedores RRD são bastante simples de desenvolver, no entanto a sua estrutura recursiva pode levantar problemas de eficiência. Para resolver esse problema existe uma outra estratégia de construção de um reconhecedor de gramáticas, designados por Reconhecedores Dirigidos por uma Tabela (RDT), estes reconhecedores substituem a recursão pela utilização de uma pilha. Por questões de tempo não se vai proceder aqui ao estudo desse tipo de reconhecedores.