



Compiladores

Análise sintática descendente

Artur Pereira <artur@ua.pt>,
Miguel Oliveira e Silva <mos@ua.pt>

DETI, Universidade de Aveiro

Ano letivo de 2023-2024

Sumário

- ① Análise sintática descendente
- ② Analisador (*parser*) recursivo-descendente preditivo
- ③ Fatorização à esquerda
- ④ Remoção de recursividade à esquerda
- ⑤ Conjuntos *first*, *follow* e *predict*
- ⑥ Tabela de decisão de um reconhecedor descendente LL(1)

Análise sintática

Introdução

- Dada uma gramática $G = (T, N, P, S)$ e uma palavra $u \in T^*$, o papel da análise sintática é:
 - descobrir uma derivação que a partir de S produza u
 - gerar uma árvore de derivação (*parse tree*) que transforme S (a raiz) em u (as folhas)
- Se nenhuma derivação/árvore existir, então $u \notin L(G)$
- A análise sintática pode ser **descendente** ou **ascendente**
- Na análise sintática descendente:
 - a derivação pretendida é **à esquerda**
 - a árvore é gerada **a partir da raiz**, descendo para as folhas
- Na análise sintática ascendente:
 - a derivação pretendida é **à direita**
 - a árvore é gerada **a partir das folhas**, subindo para a raiz
- O objetivo final é a transformação da gramática num programa (reconhecedor sintático) que produza tais derivações/árvores
 - Para as gramáticas independentes do contexto, estes reconhecedores são os **autómatos de pilha**

Análise sintática descendente

Exemplo

- Considere a gramática

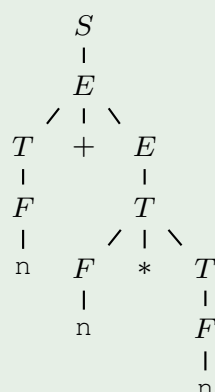
$$S \rightarrow E$$

$$E \rightarrow T + E \mid T$$

$$T \rightarrow F * T \mid F$$

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

- Desenhe-se a árvore de derivação da palavra $n+n*n$ a partir de S



Análise sintática descendente

Conceitos

- Existem diferentes abordagens à análise sintática descendente
- Análise sintática descendente **recursiva**
 - Os símbolos não terminais transformam-se em funções recursivas
 - Abordagem genérica
 - Pode requerer um algoritmo de *backtracking* (tentativa e erro) para descobrir a produção a aplicar a cada momento
- Análise sintática descendente **preditiva**
 - Abordagem recursiva ou através de uma tabela de decisão
 - No caso da tabela, os símbolos não terminais transformam-se no alfabeto da pilha do autómato de pilha
 - Não requer *backtracking*
 - A produção a aplicar a cada momento é escolhida com base no(s) primeiro(s) *token(s)* da entrada que ainda não foram consumidos (**lookahead**)
 - São designados $LL(k)$
 - k é o número (máximo) de *tokens* usados na tomada de decisão
 - O primeiro L significa que a entrada é analisada da esquerda para a direita
 - O segundo L significa que se faz uma derivação à esquerda
 - Assenta em 3 elementos de análise
 - os conjuntos **first**, **follow** e **predict**

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #1

- Sobre o alfabeto $\{a, b\}$, considere linguagem
$$L = \{a^n b^n : n \geq 0\}$$
descrita pela gramática
$$S \rightarrow a S b \mid \varepsilon$$
- Construa-se um programa com *lookahead* de 1, em que o símbolo não terminal S seja uma função recursiva, que reconheça a linguagem L .

```
int lookahead;

int main()
{
    while (1)
    {
        printf(">> ");
        adv();
        S();
        eat('\n');
        printf("\n");
    }
    return 0;
}

void S(void)
{
    switch(lookahead)
    {
        case 'a':
            eat('a'); S(); eat('b');
            break;
        default:
            epsilon();
            break;
    }
}

void adv()
{
    lookahead = getchar();
}

void eat(int c)
{
    if (lookahead != c) error();
    adv();
}

void epsilon()
{
}

void error()
{
    printf("Unexpected symbol\n");
    exit(1);
}
```

Analizador (*parser*) recursivo-descendente

Análise do exemplo #1

No programa anterior:

- `lookahead` é uma variável global que representa o próximo símbolo à entrada
- `adv()` é uma função que avança na entrada, colocando em `lookahead` o próximo símbolo
- `eat(c)` é uma função que verifica se no `lookahead` está o símbolo `c`, gerando erro se não estiver, e avança para o próximo
- Há duas produções da gramática com cabeça S , sendo a decisão central do programa a escolha de qual usar face ao valor do `lookahead`.
 - deve escolher-se $S \rightarrow a S b$ se o `lookahead` for `a`
 - e $S \rightarrow \epsilon$ se o `lookahead` for `$` ou `b`

No programa anterior, o símbolo `$`, marcador de fim de entrada, corresponde ao `\n`

- Uma palavra é aceite pelo programa se e só se
`S(); eat($)`
não der erro.

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #2

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{\omega \in T^* : \#(a, \omega) = \#(b, \omega)\}$$

descrita pela gramática

$$S \rightarrow \epsilon \mid a B S \mid b A S$$

$$A \rightarrow a \mid b A A$$

$$B \rightarrow a B B \mid b$$

- Construa um programa em que os símbolos não terminais sejam funções recursivas que reconheça a linguagem L .
- O programa terá 3 funções recursivas, A , B e S , semelhantes à função S do exemplo anterior
- Em A , deve escolher-se $A \rightarrow a$ se `lookahead` for `a` e $A \rightarrow b A A$ se for `b`
- Em B , deve escolher-se $B \rightarrow b$ se `lookahead` for `b` e $B \rightarrow a B B$ se for `a`
- Em S , deve escolher-se $S \rightarrow a B S$ se `lookahead` for `a`, $S \rightarrow b A S$ se for `b` e $S \rightarrow \epsilon$ se for `$` (este último, mais tarde saber-se-á porquê)

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #2a

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{\omega \in T^* : \#(a, \omega) = \#(b, \omega)\}$$

descrita pela gramática

$$S \rightarrow \varepsilon \mid a B \mid b A$$

$$A \rightarrow a S \mid b A A$$

$$B \rightarrow a B B \mid b S$$

- Construa um programa em que os símbolos não terminais sejam funções recursivas que reconheça a linguagem L .
- O programa terá 3 funções recursivas, A , B e S , semelhantes à função S do exemplo anterior, exceto no critério de escolha da produção $S \rightarrow \varepsilon$
- Escolher $S \rightarrow \varepsilon$ quando lookahead for $\$$ pode não resolver
- Por exemplo, com o lookahead igual a a , há situações em que se tem de escolher $S \rightarrow a B$ e outras $S \rightarrow \varepsilon$
- É o que acontece com a entrada $bbaa$
 $S \Rightarrow b A \Rightarrow bb A A \Rightarrow bba S A \Rightarrow \dots$
momento em que o S tem de ser expandido para ε e o lookahead é a

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #2b

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{\omega \in T^* : \#(a, \omega) = \#(b, \omega)\}$$

descrita pela gramática

$$S \rightarrow \varepsilon$$

$$\mid a S b S$$

$$\mid b S a S$$

- Construa um programa em que os símbolos não terminais sejam funções recursivas que reconheça a linguagem L
- Tal como no caso anterior, escolher $S \rightarrow \varepsilon$ quando lookahead for $\$$ pode não resolver

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #3

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{a^n b^n : n \geq 1\}$$

descrita pela gramática

$$\begin{array}{l} S \rightarrow a S b \\ \quad | \quad a b \end{array}$$

- Construa um programa em que o símbolo não terminal S seja uma função recursiva que reconheça a linguagem L .
- Como escolher entre as duas produções se ambas começam pelo mesmo símbolo?
- Há duas abordagens:
 - Pôr em evidência o a à esquerda, transformando a gramática para
$$\begin{array}{l} S \rightarrow a X \\ X \rightarrow S b \mid b \end{array}$$
 - Aumentar o número de símbolos de *lookahead* para 2
 - se for aa , escolhe-se $S \rightarrow a S b$
 - se for ab , escolhe-se $S \rightarrow a b$
 - se for $a\$$, dá erro

Analizador (*parser*) recursivo-descendente preditivo

Exemplo #4

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{(ab)^n : n \geq 1\}$$

descrita pela gramática

$$\begin{array}{l} S \rightarrow S a b \\ \quad | \quad a b \end{array}$$

- Construa um programa em que o símbolo não terminal S seja uma função recursiva que reconheça a linguagem L .
- Escolher a primeira produção cria um ciclo infinito, por causa da recursividade à esquerda
 - O ANTLR consegue lidar com este tipo (simples) de recursividade à esquerda, mas falha com outros tipos
 - Em geral, os reconhecedores descendentes não lidam bem com recursividade à esquerda
- Solução geral: eliminar a recursividade à esquerda

Questões a resolver

Q Que fazer quando há prefixos comuns?

R Pô-los em evidência (fatorização à esquerda)

Q Como lidar com a recursividade à esquerda?

R Transformá-la em recursividade à direita

Q Para que valores do *lookahead* usar uma regra $A \rightarrow \alpha$?

R **predict** ($A \rightarrow \alpha$)

Fatorização à esquerda

Exemplo de ilustração

- Sobre o alfabeto $\{a, b\}$, considere linguagem

$$L = \{a^n b^n : n \geq 1\}$$

descrita pela gramática

$$\begin{array}{l} S \rightarrow a S b \\ \quad | a b \end{array}$$

- Obtenha uma gramática equivalente, pondo em evidência o a
- Relaxando a definição *standard* de gramática que se tem usado, pode obter-se

$$S \rightarrow a (S b \mid b)$$

- e criando um símbolo não terminal que represente o que está entre parêntesis, obtém-se a gramática

$$\begin{array}{l} S \rightarrow a X \\ X \rightarrow b \\ \quad | S b \end{array}$$

- Esta gramática permite construir um *parser* preditivo com *lookahead* de 1

Eliminação de recursividade à esquerda

Recursividade direta simples

- A gramática seguinte, onde α e β representam seqüências de símbolos terminais e/ou não terminais, com β não começando por A , ilustra genericamente a recursividade direta simples à esquerda

$$\begin{array}{l} A \rightarrow A \alpha \\ | \quad \beta \end{array}$$

- Aplicando a primeira produção n vezes e a seguir a segunda, obtém-se

$$A \Rightarrow A \alpha \Rightarrow A \alpha \alpha \Rightarrow A \alpha \cdots \alpha \alpha \Rightarrow \beta \underbrace{\alpha \cdots \alpha \alpha}_{n \geq 0}$$

- Ou seja

$$A = \beta \alpha^n \quad n \geq 0$$

- Que corresponde ao β seguido do fecho de α , podendo ser representada pela gramática

$$\begin{array}{l} A \rightarrow \beta X \\ X \rightarrow \varepsilon \\ | \quad \alpha X \end{array}$$

- Em ANTLR seria possível fazer-se $A \rightarrow \beta (\alpha) ^*$

Eliminação de recursividade à esquerda

Exemplo com recursividade direta simples

- Para a gramática

$$\begin{array}{l} S \rightarrow S a b \\ | \quad c b a \end{array}$$

obtenha-se uma gramática equivalente sem recursividade à esquerda

- Aplicando a estratégia anterior, tem-se

$$S \Rightarrow S \underbrace{a b}_{\alpha} \Rightarrow S \underbrace{a b}_{\alpha} \cdots \underbrace{a b}_{\alpha} \Rightarrow \underbrace{c b a}_{\beta} \underbrace{a b}_{\alpha} \cdots \underbrace{a b}_{\alpha}$$

- Ou seja

$$S = \underbrace{(c b a)}_{\beta} \underbrace{(a b)}_{\alpha}^n, \quad n \geq 0$$

- Que corresponde à gramática

$$\begin{array}{l} S \rightarrow c b a X \\ X \rightarrow \varepsilon \\ | \quad a b X \end{array}$$

Eliminação de recursividade à esquerda

Recursividade direta múltipla

- A gramática seguinte, onde α_i e β_j representam sequências de símbolos terminais e/ou não terminais, com os β_j não começando por A , ilustra genericamente a recursividade direta múltipla à esquerda

$$\begin{aligned} A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \cdots \mid A \alpha_n \\ \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \end{aligned}$$

- Aplicando a estratégia anterior, tem-se

$$A = (\beta_1 \mid \beta_2 \mid \cdots \mid \beta_m)(\alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n)^k \quad k \geq 0$$

- Que corresponde à gramática

$$\begin{aligned} A \rightarrow \beta_1 X \mid \beta_2 X \mid \cdots \mid \beta_m X \\ X \rightarrow \varepsilon \\ \mid \alpha_1 X \mid \alpha_2 X \mid \cdots \mid \alpha_n X \end{aligned}$$

- Em ANTLR seria possível fazer-se $(\beta_1 \mid \beta_2 \mid \cdots \mid \beta_m)(\alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n)^*$

Eliminação de recursividade à esquerda

Exemplo com recursividade direta múltipla

- Obtenha-se uma gramática equivalente à seguinte sem recursividade à esquerda

$$\begin{aligned} S \rightarrow S a b \mid S c \\ \mid b b \mid c c \end{aligned}$$

- As palavras da linguagem são da forma

$$S = (b b \mid c c)(a b \mid c)^k, \quad k \geq 0$$

- Obtendo-se a gramática

$$\begin{aligned} S \rightarrow b b X \mid c c X \\ X \rightarrow \varepsilon \\ \mid a b X \mid c X \end{aligned}$$

Eliminação de recursividade à esquerda

Ilustração de recursividade indireta

- Aplique-se o procedimento anterior à gramática seguinte, assumindo que a recursividade à esquerda está no A

$$S \rightarrow A a \mid b$$

$$A \rightarrow A c \mid S d \mid \varepsilon$$

- O resultado seria

$$S \rightarrow A a \mid b$$

$$A \rightarrow S d X \mid X$$

$$X \rightarrow \varepsilon \mid c X$$

- A recursividade não foi eliminada

$$S \Rightarrow A a \Rightarrow S d X a \Rightarrow A a d X a$$

- Porque a recursividade existe de forma indireta
- Como resolver a recursividade à esquerda (direta e indireta)?

-
- S pode transformar-se em algo começado por A que, por sua vez, se pode transformar em algo que começa por S

Eliminação de recursividade à esquerda

Recursividade indireta

- Considere a gramática (genérica) seguinte, em que alguns dos $\alpha_i, \beta_i, \dots, \Omega_i$ podem começar por A_j , com $i, j = 1, 2, \dots, n$

$$A_1 \rightarrow \alpha_1 \mid \beta_1 \mid \dots \mid \Omega_1$$

$$A_2 \rightarrow \alpha_2 \mid \beta_2 \mid \dots \mid \Omega_2$$

...

$$A_n \rightarrow \alpha_n \mid \beta_n \mid \dots \mid \Omega_n$$

- Algoritmo:

- Define-se uma ordem para os símbolos não terminais, por exemplo

$$A_1, A_2, \dots, A_n$$

- Para cada A_i :

- fazem-se transformações de equivalência de modo a garantir que nenhuma produção com cabeça A_i se expande em algo começado por A_j , com $j < i$
- elimina-se a recursividade à esquerda direta que as produções começadas por A_i possam ter

Eliminação de recursividade à esquerda

Exemplo com recursividade indireta

- Aplique-se este procedimento à gramática seguinte, estabelecendo-se a ordem S, A

$$S \rightarrow A a \mid b$$

$$A \rightarrow A c \mid S d \mid \varepsilon$$

- As produções começadas por S satisfazem a condição, pelo que não é necessária qualquer transformação
- A produção $A \rightarrow S d$ viola a regra definida, pelo que, nela, S é substituído por $(A a \mid b)$, obtendo-se

$$S \rightarrow A a \mid b$$

$$A \rightarrow A c \mid A a d \mid b d \mid \varepsilon$$

- Elimina-se a recursividade à esquerda direta das produções começadas por A , obtendo-se

$$S \rightarrow A a \mid b$$

$$A \rightarrow b d X \mid X$$

$$X \rightarrow \varepsilon \mid c X \mid a d X$$

Conjuntos **predict**, **first** e **follow**

Definições

- Considere uma gramática $G = (T, N, P, S)$ e uma produção $(A \rightarrow \alpha) \in P$
- O conjunto **predict** $(A \rightarrow \alpha)$ representa os valores de *lookahead* para os quais A deve ser expandido para α . Define-se por:

$$\mathbf{predict}(A \rightarrow \alpha) =$$

$$\begin{cases} \mathbf{first}(\alpha) & \varepsilon \notin \mathbf{first}(\alpha) \\ (\mathbf{first}(\alpha) - \{\varepsilon\}) \cup \mathbf{follow}(A) & \varepsilon \in \mathbf{first}(\alpha) \end{cases}$$

- O conjunto **first** (α) representa as letras (símbolos terminais) pelas quais as palavras geradas por α podem começar mais o ε se for possível transformar todo o α em ε . Define-se por:

$$\mathbf{first}(\alpha) = \{t \in T : \alpha \Rightarrow^* t\omega \wedge \omega \in T^*\} \cup \{\varepsilon \text{ sse } \alpha \Rightarrow^* \varepsilon\}$$

- O conjunto **follow** (A) representa as letras (símbolos terminais, incluindo o $\$$) que podem aparecer imediatamente à direita de A numa derivação. Define-se por:

$$\mathbf{follow}(A) = \{t \in T_{\$} : S\$ \Rightarrow^* \gamma A t\omega\} \quad \text{com} \quad T_{\$} = \{T \cup \$\}$$

Conjunto **first**

Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

```
first ( $\alpha$ ) {  
  if ( $\alpha = \varepsilon$ ) then  
    return  $\{\varepsilon\}$   
   $h = \mathbf{head}(\alpha)$       # com  $|h| = 1$   
   $\omega = \mathbf{tail}(\alpha)$      # tal que  $\alpha = h \omega$   
  if ( $h \in T$ ) then  
    return  $\{h\}$   
  else  
    return  $\bigcup_{(h \rightarrow \beta_i) \in P} \mathbf{first}(\beta_i \omega)$     # concatenação de  $\beta_i$  com  $\omega$   
}
```

- Note que no último **return** o argumento do **first** é $\beta_i \omega$, concatenação dos β_i (que vêm dos corpos das produções começadas por h) com o ω (**tail** do α)
- Este algoritmo pode não convergir se a gramática tiver recursividade à esquerda

Conjunto **first**

Exemplo #1

- Considere a gramática

$$S \rightarrow a S \mid B C$$

$$B \rightarrow \varepsilon \mid b S$$

$$C \rightarrow c \mid c S$$

- Determine o conjunto **first** ($a S$)

- Porque $a S$ começa pelo símbolo terminal a

$$\mathbf{first}(a S) = \{a\}.$$

- Determine o conjunto **first** ($B C$)

- Porque $B C$ começa pelo símbolo não terminal B

$$\mathbf{first}(B C) = \mathbf{first}(C) \cup \mathbf{first}(b S C)$$

- Porque C começa pelo símbolo não terminal C

$$\mathbf{first}(C) = \mathbf{first}(c) \cup \mathbf{first}(c S)$$

$$\therefore \mathbf{first}(B C) = \mathbf{first}(c) \cup \mathbf{first}(c S) \cup \mathbf{first}(b S C) = \{b, c\}$$

- Note que, embora B se possa transformar em ε , $\varepsilon \notin \mathbf{first}(B C)$
- Por essa razão, $\mathbf{first}(B C) \neq \mathbf{first}(B) \cup \mathbf{first}(C)$

Conjunto **first**

Exemplo #2

- Considere a gramática

$$S \rightarrow a S \mid B C$$

$$B \rightarrow \varepsilon \mid b S$$

$$C \rightarrow \varepsilon \mid c S$$

- Determine o conjunto **first** (BC)

- Porque BC começa pelo símbolo não terminal B

$$\mathbf{first}(BC) = \mathbf{first}(C) \cup \mathbf{first}(bSC)$$

- Porque C começa pelo símbolo não terminal C

$$\mathbf{first}(C) = \mathbf{first}(\varepsilon) \cup \mathbf{first}(cS)$$

$$\mathbf{first}(BC) = \mathbf{first}(\varepsilon) \cup \mathbf{first}(cS) \cup \mathbf{first}(bSC)$$

$$= \{\varepsilon, b, c\}$$

- Note que a gramática não é a mesma
- Note que $\varepsilon \in \mathbf{first}(BC)$ apenas porque todo o BC se pode transformar em ε

Conjunto **follow**

Algoritmo de cálculo

- Os conjuntos **follow** podem ser calculados através de um algoritmo iterativo envolvendo todos os símbolos não terminais

- Aplicam-se as seguintes regras:

① $\$ \in \mathbf{follow}(S)$

② **if** ($A \rightarrow \alpha B \in P$) **then**
 $\mathbf{follow}(B) \supseteq \mathbf{follow}(A)$

③ **if** ($A \rightarrow \alpha B \beta \in P \wedge (\varepsilon \notin \mathbf{first}(\beta))$) **then**
 $\mathbf{follow}(B) \supseteq \mathbf{first}(\beta)$

④ **if** ($A \rightarrow \alpha B \beta \in P \wedge (\varepsilon \in \mathbf{first}(\beta))$) **then**
 $\mathbf{follow}(B) \supseteq ((\mathbf{first}(\beta) - \{\varepsilon\}) \cup \mathbf{follow}(A))$

- Partindo de conjuntos vazios, aplicam-se sucessivamente estas regras até que nada seja acrescentado

- Note que \supseteq significa **contém** e não está contido

Conjunto follow

Exemplo #1

- Considere a gramática

$$S \rightarrow a S \mid B C$$

$$B \rightarrow \varepsilon \mid b S$$

$$C \rightarrow c \mid c S$$

- Determine o conjunto **follow**(B)

- Procuram-se ocorrências de B no lado direito das produções. Há uma: na produção $S \rightarrow B C$
- A produção $S \rightarrow B C$ encaixa na regra 3 ou na regra 4 (mas apenas numa delas), dependendo de o ε pertencer ou não ao **first**(C)
- **first**(C) = $\{c\}$
- $\therefore \text{follow}(B) \supseteq \text{first}(C)$ [regra 3]
- Não havendo mais contribuições, tem-se
follow(B) = $\{c\}$

Conjunto follow

Exemplo #2

- Considere a gramática

$$S \rightarrow a S \mid B C$$

$$B \rightarrow \varepsilon \mid b S$$

$$C \rightarrow \varepsilon \mid c S$$

- Determine o conjunto **follow**(B)

- A produção $S \rightarrow B C$ encaixa na regra 3 ou na regra 4 (mas apenas numa delas), dependendo de o ε pertencer ou não ao **first**(C)
- **first**(C) = $\{\varepsilon, c\}$
- $\therefore \text{follow}(B) \supseteq ((\text{first}(C) - \{\varepsilon\}) \cup \text{follow}(S))$ [regra 4]
- Porque S é o símbolo inicial, $\$ \in \text{follow}(S)$ [regra 1]
- A produção $S \rightarrow a S$ é irrelevante, porque diz que **follow**(S) $\supseteq \text{follow}(S)$
- A produção $B \rightarrow b S$ diz que **follow**(S) $\supseteq \text{follow}(B)$
- A produção $C \rightarrow c S$ diz que **follow**(S) $\supseteq \text{follow}(C)$
- A produção $S \rightarrow B C$ diz que **follow**(C) $\supseteq \text{follow}(S)$
- Pelas contribuições tem-se que
follow(B) = $\{c, \$\}$
- Também se ficou a saber que **follow**(S) = **follow**(B) = **follow**(C)

Conjunto follow

Exemplo #3

- Considere a gramática

$$\begin{aligned} S &\rightarrow a S \mid B C \\ B &\rightarrow b \mid b S \\ C &\rightarrow \varepsilon \mid S c \end{aligned}$$

- Determine o conjunto **follow**(B)
 - A produção $S \rightarrow B C$ encaixa na regra 3 ou na regra 4 (mas apenas numa delas), dependendo de o ε pertencer ou não ao **first**(C)
 - **first**(C) = $\{\varepsilon, a, b\}$
 - $\therefore \text{follow}(B) \supseteq (\text{first}(C) - \{\varepsilon\}) \cup \text{follow}(S)$
 - Porque S é o símbolo inicial, $\$ \in \text{follow}(S)$
 - A produção $S \rightarrow a S$ é irrelevante, porque diz que $\text{follow}(S) \supseteq \text{follow}(S)$
 - A produção $B \rightarrow b S$ diz que $\text{follow}(S) \supseteq \text{follow}(B)$
 - A produção $C \rightarrow S c$ diz que $\text{follow}(S) \supseteq \{c\}$
 - Pelas contribuições tem-se que
 $\text{follow}(B) = \{a, b, c, \$\}$
 - Note que o ε **nunca pertence** a um **follow**

Reconhecedor descendente preditivo

Tabela de decisão (*parsing table*)

- Um reconhecedor descendente preditivo pode basear-se numa tabela de decisão
 - Apenas se irá abordar para o caso de *lookahead* de 1
- Corresponde a uma função $\tau : N \times T_{\$} \rightarrow \wp(P)$, onde $T_{\$} = T \cup \{\$\}$ e $\wp(P)$ representa o conjunto dos subconjuntos de P
- Pode ser representada por uma tabela, onde os elementos de N indexam as linhas, os elementos de $T_{\$}$ indexam as colunas, e as células são preenchidas com produções (subconjuntos elementos de P)
- Pode ser obtida (ou a tabela preenchida) usando o seguinte algoritmo:

Algoritmo:

```
foreach  $(n, t) \in (N \times T_{\$})$ 
     $\tau(n, t) = \emptyset$            # começa-se com as células vazias
foreach  $(A \rightarrow \alpha) \in P$ 
    foreach  $t \in \text{predict}(A \rightarrow \alpha)$ 
        add  $(A \rightarrow \alpha)$  to  $\tau(A, t)$ 
```

Tabela de decisão

Exemplo #1

- Considere a gramática

$$S \rightarrow a S b \mid \varepsilon$$

- Preencha a tabela de decisão de um reconhecedor descendente desta linguagem com *lookahead* de 1

$$\mathbf{first}(a S b) = \{a\}$$

$$\therefore \mathbf{predict}(S \rightarrow a S b) = \{a\}$$

$$\mathbf{first}(\varepsilon) = \{\varepsilon\}$$

$$\mathbf{follow}(S) = \{\$, b\}$$

$$\therefore \mathbf{predict}(S \rightarrow \varepsilon) = \{\$, b\}$$

Tabela de decisão

	a	b	\$
S	a S b	ε	ε

- Não havendo células com 2 ou mais produções, a gramática é LL(1)

- Para simplificação, optou-se por pôr nas células apenas o corpo da produção, uma vez que a cabeça é definida pela linha da tabela

Tabela de decisão

Exemplo #2

- Considere a gramática

$$S \rightarrow \varepsilon \mid a B S \mid b A S$$

$$A \rightarrow a \mid b A A$$

$$B \rightarrow a B B \mid b$$

- Preencha a tabela de decisão de um reconhecedor descendente desta linguagem com *lookahead* de 1

$$\mathbf{predict}(S \rightarrow a B S) = \{a\}$$

$$\mathbf{predict}(S \rightarrow b A S) = \{b\}$$

$$\mathbf{predict}(S \rightarrow \varepsilon) = \{\$\}$$

$$\mathbf{predict}(A \rightarrow a) = \{a\}$$

$$\mathbf{predict}(A \rightarrow b A A) = \{b\}$$

$$\mathbf{predict}(B \rightarrow a B B) = \{a\}$$

$$\mathbf{predict}(B \rightarrow b) = \{b\}$$

Tabela de decisão

	a	b	\$
S	a B S	b A S	ε
A	a	b A A	
B	a B B	b	

- As células vazias correspondem a situações de erro

Tabela de decisão

Exemplo #2a

- Considere a gramática

$$S \rightarrow \varepsilon \mid a B \mid b A$$

$$A \rightarrow a S \mid b A A$$

$$B \rightarrow a B B \mid b S$$

- Preencha a tabela de decisão de um reconhecedor descendente desta linguagem com *lookahead* de 1

$$\text{predict}(S \rightarrow a B) = \{a\}$$

$$\text{predict}(S \rightarrow b A) = \{b\}$$

$$\text{predict}(S \rightarrow \varepsilon) = \{a, b, \$\}$$

$$\text{predict}(A \rightarrow a S) = \{a\}$$

$$\text{predict}(A \rightarrow b A A) = \{b\}$$

$$\text{predict}(B \rightarrow a B B) = \{a\}$$

$$\text{predict}(B \rightarrow b S) = \{b\}$$

Tabela de decisão

	a	b	\$
S	a B, ε	b A, ε	ε
A	a S	b A A	
B	a B B	b S	

- As células (S, a) e (S, b) têm duas produções cada, o que torna o reconhecimento inviável para um *lookahead* de 1, pelo que a linguagem não é LL(1)

Tabela de decisão

Exemplo #2b

- Considere a gramática

$$S \rightarrow \varepsilon$$

$$\mid a S b S$$

$$\mid b S a S$$

- Preencha a tabela de decisão de um reconhecedor descendente desta linguagem com *lookahead* de 1

$$\text{predict}(S \rightarrow a S b S) = \{a\}$$

$$\text{predict}(S \rightarrow b S a S) = \{b\}$$

$$\text{predict}(S \rightarrow \varepsilon) = \{a, b, \$\}$$

Tabela de decisão

	a	b	\$
S	a A b S, ε	b S a S, ε	ε

- As células (S, a) e (S, b) têm duas produções cada, o que torna o reconhecimento inviável para um *lookahead* de 1, pelo que a linguagem não é LL(1)

Tabela de decisão

Exemplo #3

- Considere, sobre o alfabeto $\{i, f, v, ,, ;\}$, a linguagem L_4 descrita pela gramática

$$D \rightarrow T L ;$$

$$T \rightarrow i$$

$$| \quad f$$

$$L \rightarrow v$$

$$| \quad v , L$$

- Obtenha-se uma tabela de decisão de um reconhecedor descendente, com *lookahead* de 1, que reconheça a linguagem L_4 .

- A gramática dada não permite fazê-lo, porque

$$\text{predict}(L \rightarrow v) = \{v\}$$

$$\text{predict}(L \rightarrow v , L) = \{v\}$$

pelo que as duas produções iriam ficar na mesma célula da tabela de decisão.

- É necessário transformar a gramática noutra equivalente, fatorizando à esquerda as produções com cabeça L .

Tabela de decisão

Exemplo #3 (cont.)

$$D \rightarrow T L ;$$

$$T \rightarrow i$$

$$| \quad f$$

$$L \rightarrow v X$$

$$X \rightarrow$$

$$| \quad , L$$

$$\text{predict}(D \rightarrow T L ;) = ?$$

$$\text{first}(T L ;) = ?$$

$$\text{first}(T) = \text{first}(i) \cup \text{first}(f) = \{i\} \cup \{f\}$$

$$\therefore \text{first}(T L ;) = \{i, f\}$$

$$\therefore \text{predict}(D \rightarrow T L ;) = \{i, f\}$$

$$\text{predict}(T \rightarrow i) = ?$$

$$\text{first}(i) = \{i\}$$

$$\therefore \text{predict}(T \rightarrow i) = \{i\}$$

$$\text{predict}(T \rightarrow f) = \{f\}$$

$$\text{predict}(L \rightarrow v X) = ?$$

$$\text{first}(v X) = \{v\}$$

$$\therefore \text{predict}(L \rightarrow v X) = \{v\}$$

$$\text{predict}(X \rightarrow \epsilon) = ?$$

$$\text{first}(\epsilon) = \{\epsilon\}$$

$$\therefore \text{predict}(X \rightarrow \epsilon) = \text{follow}(X)$$

$$\text{follow}(X) = \text{follow}(L) = \{;\}$$

$$\therefore \text{predict}(X \rightarrow \epsilon) = \{;\}$$

$$\text{predict}(X \rightarrow , L) = \{, \}$$

Tabela de decisão

Exemplo #3 (cont.)

$D \rightarrow T L ;$

$T \rightarrow i$

$\mid f$

$L \rightarrow v X$

$X \rightarrow$

\mid , L

$\text{predict}(D \rightarrow T L ;) = \{i, f\}$

$\text{predict}(T \rightarrow i) = \{i\}$

$\text{predict}(T \rightarrow f) = \{f\}$

$\text{predict}(L \rightarrow v X) = \{v\}$

$\text{predict}(X \rightarrow \epsilon) = \{;\}$

$\text{predict}(X \rightarrow , L) = \{, \}$

Tabela de decisão

	i	f	v	,	;	\$
D	$T L ;$	$T L ;$				
T	i	f				
L			$v X$			
X				$, L$	ϵ	

- As células vazias são situações de erro