Compiladores

Análise sintática descendente

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

DETI, Universidade de Aveiro

Ano letivo de 2024-2025

Sumário

Análise sintática
Introdução

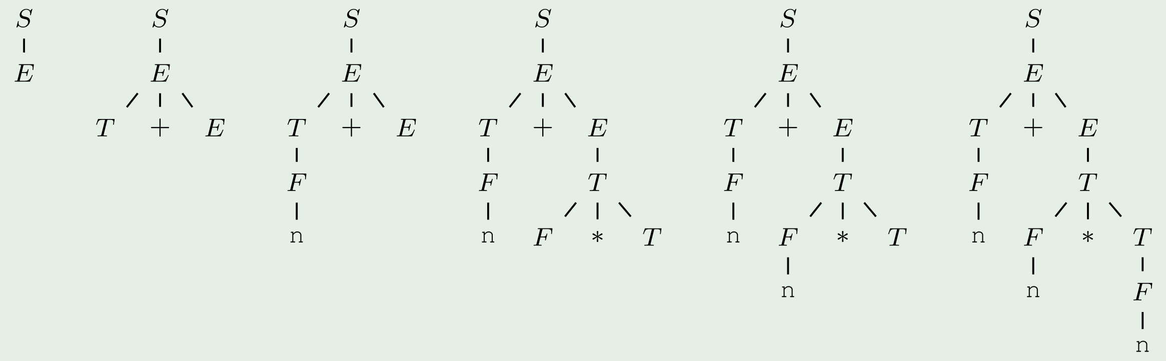
- Dada uma gramática $G = (T, N, P, S)$ e uma palavra $u \in T^*$, o papel da análise sintática é:
  - descobrir uma derivação que a partir de S produza u
  - gerar uma árvore de derivação (*parse tree*) que transforme S (a raiz) em u (as folhas)
- Se nenhuma derivação/árvore existir, então $u \notin L(G)$
- 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

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow T + E \mid T \\
T &\rightarrow F \star T \mid F \\
F &\rightarrow n \mid (E)
\end{aligned}
$$

- 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 baseada numa tabela de decisão
    - 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 de *tokens* usados na tomada de decisão
    - O primeiro L significa que a entrada é processada da esquerda para a direita (sem *backtracking*)
    - O segundo L significa que se produz uma derivação à esquerda
  - Pode ser implementado por uma estrutura recursivo ou por um autómato de pilha em que os símbolos não terminais se transformam no alfabeto da pilha
  - Assenta em 3 elementos de análise: os conjuntos **first**, **follow** e **predict**

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #1

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

$$L = \{\mathtt{a}^n\mathtt{b}^n \;:\; n \geq 0\}$$

descrita pela gramática

$$S \rightarrow \mathtt{a}\ S\ \mathtt{b} \ |\ \varepsilon$$

- 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);
}
```

Analisador (*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 `loookahead`.
  - deve escolher-se $S \to \texttt{a}\ S\ \texttt{b}$ se o `lookahead` for `a`
  - e $S \to \varepsilon$ 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.

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #2

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

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

  descrita pela gramática

  $$\begin{aligned} S &\to \varepsilon \ \mid \ \texttt{a}\ B\ S \ \mid \ \texttt{b}\ A\ S \\ A &\to \texttt{a} \ \mid \ \texttt{b}\ A\ A \\ B &\to \texttt{a}\ B\ B \ \mid \ \texttt{b} \end{aligned}$$

- Construa um programa em que os símbolos não terminais sejas 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 $A \to \texttt{a}$ se `lookahead` for `a` e $A \to \texttt{b}\ A\ A$ se for `b`
- Em B, deve escolher-se $B \to \texttt{b}$ se `lookahead` for `b` e $B \to \texttt{a}\ B\ B$ se for `a`
- Em S, deve escolher-se $S \to \texttt{a}\ B\ S$ se `lookahead` for `a`, $S \to \texttt{b}\ A\ S$ se for `b` e $S \to \varepsilon$ se for `$` (este último, mais tarde saber-se-á porquê)

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #2a

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

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

  descrita pela gramática

  $$\begin{aligned} S &\to \varepsilon \ | \ \mathtt{a} \ B \ | \ \mathtt{b} \ A \\ A &\to \mathtt{a} \ S \ | \ \mathtt{b} \ A \ A \\ B &\to \mathtt{a} \ B \ B \ | \ \mathtt{b} \ S \end{aligned}$$

- Construa um programa em que os símbolos não terminais sejas 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 $S \to \varepsilon$
- Escolher $S \to \varepsilon$ quando `lookahead` for `$` pode não resolver
- Por exemplo, com o `lookahead` igual a `a`, há situações em que se tem de escolher $S \to \mathtt{a}\ B$ e outras $S \to \varepsilon$
- É o que acontece com a entrada `bbaa`

  $S \Rightarrow \mathtt{b}\, A \Rightarrow \mathtt{bb}\, A\, A \Rightarrow \mathtt{bba}\, S\, A \Rightarrow \cdots$

  momento em que o S tem de ser expandido para ε e o `lookahead` é `a`

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #2b

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

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

  descrita pela gramática

  $$\begin{aligned} S \ &\to \ \varepsilon \\ &| \ \ \mathtt{a} \ S \ \mathtt{b} \ S \\ &| \ \ \mathtt{b} \ S \ \mathtt{a} \ S \end{aligned}$$

- Construa um programa em que os símbolos não terminais sejas funções recursivas que reconheça a linguagem L

- Tal como no caso anterior, escolher $S \to \varepsilon$ quando `lookahead` for `$` pode não resolver

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #3

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

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

  descrita pela gramática

  $S \to \mathtt{a}\ S\ \mathtt{b}$

  $\quad\mid\ \mathtt{a}\ \mathtt{b}$

- 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 esqueda, transformando a gramática para
    $S \to \mathtt{a}\ X$
    $X \to S\ \mathtt{b} \mid \mathtt{b}$
  - Aumentar o número de símbolos de *lookahead* para 2
    - se for `aa`, escolhe-se $S \to \mathtt{a}\ S\ \mathtt{b}$
    - se for `ab`, escolhe-se $S \to \mathtt{a}\ \mathtt{b}$
    - se for `a$`, dá erro

Analisador (*parser*) recursivo-descendente preditivo
Exemplo #4

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

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

  descrita pela gramática

  $S \to S\ \mathtt{a}\ \mathtt{b}$

  $\quad\mid\ \mathtt{a}\ \mathtt{b}$

- 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

$\mathcal{Q}$ Que fazer quando há prefixos comuns?

$\mathcal{R}$ Pô-los em evidência (fatorização à esquerda)

$\mathcal{Q}$ Como lidar com a recursividade à esquerda?

$\mathcal{R}$ Transformá-la em recursividade à direita

$\mathcal{Q}$ Para que valores do *lookahead* usar uma regra $A \to \alpha$?

$\mathcal{R}$ **predict** $(A \to \alpha)$

Fatorização à esquerda
Exemplo de ilustração

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

  $$L = \{\texttt{a}^n\texttt{b}^n \; : \; n \ge 1\}$$

  descrita pela gramática

  $$\begin{aligned} S &\to \texttt{a}\ S\ \texttt{b} \\ &\mid \texttt{a}\ \texttt{b} \end{aligned}$$

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

  $$S \to \texttt{a}\ (\ S\ \texttt{b} \mid \texttt{b}\ )$$

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

  $$\begin{aligned} S &\to \texttt{a}\ X \\ X &\to \texttt{b} \\ &\mid S\ \texttt{b} \end{aligned}$$

- 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 sequências de símbolos terminais e/ou não terminais, com β não começando por A, ilustra genericamente a recursividade direta simples à esquerda
$$\begin{array}{rcl} A & \to & A\ \alpha \\ & | & \beta \end{array}$$
- 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
$$\begin{array}{rcl} A & \to & \beta\ X \\ X & \to & \varepsilon \\ & | & \alpha\ X \end{array}$$

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

Eliminação de recursividade à esquerda
Exemplo com recursividade direta simples

- Para a gramática
$$\begin{array}{rcl} S & \to & S\ \mathtt{a}\ \mathtt{b} \\ & | & \mathtt{c}\ \mathtt{b}\ \mathtt{a} \end{array}$$
obtenha-se uma gramática equivalente sem recursividade à esquerda
- Aplicando a estratégia anterior, tem-se
$$S \Rightarrow S\ \underbrace{\mathtt{a\,b}}_{\alpha} \Rightarrow S\ \underbrace{\mathtt{a\,b}}_{\alpha}\ \cdots\ \underbrace{\mathtt{a\,b}}_{\alpha} \Rightarrow \underbrace{\mathtt{c\,b\,a}}_{\beta}\ \underbrace{\mathtt{a\,b}}_{\alpha}\ \cdots\ \underbrace{\mathtt{a\,b}}_{\alpha}$$
- Ou seja
$$S = (\underbrace{\mathtt{c\,b\,a}}_{\beta})\ (\underbrace{\mathtt{a\,b}}_{\alpha})^n, \qquad n \geq 0$$
- Que corresponde à gramática
$$\begin{array}{rcl} S & \to & \mathtt{c}\ \mathtt{b}\ \mathtt{a}\ X \\ X & \to & \varepsilon \\ & | & \mathtt{a}\ \mathtt{b}\ X \end{array}$$

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

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

- 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

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

- 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

$$\begin{aligned} S \;&\rightarrow\; S\ \mathtt{a}\ \mathtt{b} \;|\; S\ \mathtt{c} \\ &\;|\; \mathtt{b}\ \mathtt{b} \;|\; \mathtt{c}\ \mathtt{c} \end{aligned}$$

- As palavras da linguagem são da forma

$$S = (\mathtt{b\,b} \,|\, \mathtt{c\,c})\,(\mathtt{a\,b} \,|\, \mathtt{c})^k, \qquad k \geq 0$$

- Obtendo-se a gramática

$$\begin{aligned} S \;&\rightarrow\; \mathtt{b}\ \mathtt{b}\ X \;|\; \mathtt{c}\ \mathtt{c}\ X \\ X \;&\rightarrow\; \varepsilon \\ &\;|\; \mathtt{a}\ \mathtt{b}\ X \;|\; \mathtt{c}\ X \end{aligned}$$

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

$$
\begin{aligned}
S &\rightarrow A\ \texttt{a} \mid \texttt{b} \\
A &\rightarrow A\ \texttt{c} \mid S\ \texttt{d} \mid \varepsilon
\end{aligned}
$$

- O resultado seria

$$
\begin{aligned}
S &\rightarrow A\ \texttt{a} \mid \texttt{b} \\
A &\rightarrow S\ \texttt{d}\ X \mid X \\
X &\rightarrow \varepsilon \mid \texttt{c}\ X
\end{aligned}
$$

- A recursividade não foi eliminada

$$S \Rightarrow A\,\texttt{a} \Rightarrow S\,\texttt{d}\,X\,\texttt{a} \Rightarrow A\,\texttt{a}\,\texttt{d}\,X\,\texttt{a}$$

  - 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 α_i, β_i, $\cdots$, Ω_i podem começar por A_j, com $i, j = 1, 2, \cdots, n$

$$
\begin{aligned}
A_1 &\rightarrow \alpha_1 \mid \beta_1 \mid \cdots \mid \Omega_1 \\
A_2 &\rightarrow \alpha_2 \mid \beta_2 \mid \cdots \mid \Omega_2 \\
&\cdots \\
A_n &\rightarrow \alpha_n \mid \beta_n \mid \cdots \mid \Omega_n
\end{aligned}
$$

- Algoritmo:
  - Define-se uma ordem para os símbolos não terminais, por exemplo $A_1, A_2, \cdots, A_n$
  - 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

$$S \to A\ \mathtt{a}\ |\ \mathtt{b}$$
$$A \to A\ \mathtt{c}\ |\ S\ \mathtt{d}\ |\ \varepsilon$$

- As produções começadas por S satisfazem a condição, pelo que não é necessária qualquer transformção
- A produção $A \to S\ \mathtt{d}$ viola a regra definida, pelo que, nela, S é substituído por ($A\ \mathtt{a}\,|\,\mathtt{b}$), obtendo-se

$$S \to A\ \mathtt{a}\ |\ \mathtt{b}$$
$$A \to A\ \mathtt{c}\ |\ A\ \mathtt{a}\ \mathtt{d}\ |\ \mathtt{b}\ \mathtt{d}\ |\ \varepsilon$$

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

$$S \to A\ \mathtt{a}\ |\ \mathtt{b}$$
$$A \to \mathtt{b}\ \mathtt{d}\ X\ |\ X$$
$$X \to \varepsilon\ |\ \mathtt{c}\ X\ |\ \mathtt{a}\ \mathtt{d}\ X$$

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

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

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

- 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:

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

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

$$\mathbf{follow}\,(A) = \{t \in (T \cup \{\$\})\ :\ S\,\$ \Rightarrow^* \gamma\, A\, t\, \beta \wedge \gamma, \beta \in (T \cup \{\$\} \cup N)^*\}$$

Conjunto **first**
Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

$$
\begin{array}{l}
\texttt{first}\,(\alpha)\,\{\\
\quad \texttt{if}\,(\alpha = \varepsilon)\ \texttt{then}\\
\quad\quad \texttt{return}\,\{\varepsilon\}\\
\quad h = \texttt{head}\ (\alpha) \qquad \textit{\# com } |h| = 1\\
\quad \omega = \texttt{tail}\ (\alpha) \qquad \textit{\# tal que } \alpha = h\,\omega\\
\quad \texttt{if}\,(\,h \in T\,)\ \texttt{then}\\
\quad\quad \texttt{return}\,\{h\}\\
\quad \texttt{else}\\
\quad\quad \texttt{return} \displaystyle\bigcup_{(h\to\beta_i)\in P} \texttt{first}\,(\beta_i\,\omega) \qquad \textit{\# concatenação de } \beta_i \textit{ com } \omega\\
\}
\end{array}
$$

- 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 gramatica

$$
\begin{array}{l}
S \to \mathrm{a}\ S \ \mid\ B\ C\\
B \to \varepsilon \ \mid\ \mathrm{b}\ S\\
C \to \mathrm{c} \mid \mathrm{c}\ S
\end{array}
$$

- Determine o conjunto **first** $(\mathrm{a}\,S)$
  - Porque $\mathrm{a}\,S$ começa pelo símbolo terminal a

    first $(\mathrm{a}\,S) = \{\mathrm{a}\}$.
- Determine o conjunto **first** $(B\,C)$
  - Porque $B\,C$ começa pelo símbolo não terminal B

    first $(B\,C) =$ **first** $((\varepsilon \mid \mathrm{b}\,S)\,C) =$ **first** $(C)\ \cup$ **first** $(\mathrm{b}\,S\,C)$
  - Porque C começa pelo símbolo não terminal C

    first $(C) =$ **first** $((\mathrm{c} \mid \mathrm{c}\,S)\,\varepsilon) =$ **first** $(\mathrm{c})\ \cup$ **first** $(\mathrm{c}\,S)$
  - $\therefore$ **first** $(B\,C) =$ **first** $(\mathrm{c})\ \cup$ **first** $(\mathrm{c}\,S)\ \cup$ **first** $(\mathrm{b}\,S\,C) = \{\mathrm{b}, \mathrm{c}\}$

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

Conjunto `first`
Exemplo #2

- Considere a gramatica

$$S \to \mathtt{a}\ S \mid B\ C$$
$$B \to \varepsilon \mid \mathtt{b}\ S$$
$$C \to \varepsilon \mid \mathtt{c}\ S$$

- Determine o conjunto **`first`** $(B\,C)$
  - Porque $B\,C$ começa pelo símbolo não terminal B

    $\texttt{first}\,(B\,C) = \texttt{first}\,((\varepsilon \,|\, \mathrm{b}\, S)\, C) = \texttt{first}\,(C) \cup \texttt{first}\,(\mathrm{b}\, S\, C)$
  - Porque C começa pelo símbolo não terminal C

    $\texttt{first}\,(C) = \texttt{first}\,((\varepsilon \,|\, \mathrm{c}\, S)\, \varepsilon) = \texttt{first}\,(\varepsilon) \cup \texttt{first}\,(\mathrm{c}\, S)$
  - $\therefore\quad \texttt{first}\,(B\,C) = \texttt{first}\,(\varepsilon) \cup \texttt{first}\,(c\,S) \cup \texttt{first}\,(\mathrm{b}\,S\,C)$

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

- Note que a gramática não é a mesma
- Note que $\varepsilon \in \texttt{first}\,(B\,C)$ apenas porque todo o $B\,C$ 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. $\$ \in \texttt{follow}\,(S)$
2. **if** $(A \to \alpha B \in P)$ **then**

   $\texttt{follow}\,(B) \supseteq \texttt{follow}\,(A)$
3. **if** $(A \to \alpha B \beta \in P) \wedge (\varepsilon \notin \texttt{first}\,(\beta))$ **then**

   $\texttt{follow}\,(B) \supseteq \texttt{first}\,(\beta)$
4. **if** $(A \to \alpha B \beta \in P) \wedge (\varepsilon \in \texttt{first}\,(\beta))$ **then**

   $\texttt{follow}\,(B) \supseteq ((\texttt{first}\,(\beta) - \{\varepsilon\}) \cup \texttt{follow}\,(A))$

- 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 gramatica

$$S \rightarrow \mathrm{a}\ S \mid B\ C$$
$$B \rightarrow \varepsilon \mid \mathrm{b}\ S$$
$$C \rightarrow \mathrm{c} \mid \mathrm{c}\ S$$

- Determine o conjunto $\texttt{follow}\,(B)$
  - Procuram-se ocurrências 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 $\texttt{first}\,(C)$
  - $\texttt{first}\,(C) = \{\mathrm{c}\}$
  - $\therefore\ \texttt{follow}\,(B) \supseteq \texttt{first}\,(C)$ [*regra 3*]
  - Não havendo mais contribuições, tem-se

    $\texttt{follow}\,(B) = \{c\}$

Conjunto follow
Exemplo #2

- Considere a gramatica

$$S \rightarrow \mathrm{a}\ S \mid B\ C$$
$$B \rightarrow \varepsilon \mid \mathrm{b}\ S$$
$$C \rightarrow \varepsilon \mid \mathrm{c}\ S$$

- Determine o conjunto $\texttt{follow}\,(B)$
  - 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 $\texttt{first}\,(C)$
  - $\texttt{first}\,(C) = \{\varepsilon, \mathrm{c}\}$
  - $\therefore\ \texttt{follow}\,(B) \supseteq ((\texttt{first}\,(C) - \{\varepsilon\}) \cup \texttt{follow}\,(S))$ [*regra 4*]
  - Porque S é o símbolo inicial, $\$ \in \texttt{follow}\,(S)$ [*regra 1*]
  - A produção $S \rightarrow \mathrm{a}\,S$ é irrelevante, porque diz que $\texttt{follow}\,(S) \supseteq \texttt{follow}\,(S)$
  - A produção $B \rightarrow \mathrm{b}\,S$ diz que $\texttt{follow}\,(S) \supseteq \texttt{follow}\,(B)$
  - A produção $C \rightarrow \mathrm{c}\,S$ diz que $\texttt{follow}\,(S) \supseteq \texttt{follow}\,(C)$
  - A produção $S \rightarrow B\,C$ diz que $\texttt{follow}\,(C) \supseteq \texttt{follow}\,(S)$
  - Pelas contribuições tem-se que

    $\texttt{follow}\,(B) = \{\mathrm{c}, \$\}$
  - Também se ficou a saber que $\texttt{follow}\,(S) = \texttt{follow}\,(B) = \texttt{follow}\,(C)$

Conjunto $\texttt{follow}$
Exemplo #3

- Considere a gramatica

$$S \rightarrow \texttt{a}\ S \mid B\ C$$
$$B \rightarrow \texttt{b} \mid \texttt{b}\ S$$
$$C \rightarrow \varepsilon \mid S\ \texttt{c}$$

- Determine o conjunto $\texttt{follow}\,(B)$

produção	**contribuição para um follow**	**regra**
$S \rightarrow B\,C$	$\texttt{first}\,(C) = \{\varepsilon, \texttt{a}, \texttt{b}\}$	
$S \rightarrow B\,C$	$\texttt{follow}\,(B) \supseteq (\texttt{first}\,(C) - \{\varepsilon\}) \cup \texttt{follow}\,(S)$	4
	$\$ \in \texttt{follow}\,(S)$	1
$S \rightarrow \texttt{a}\,S$	$\texttt{follow}\,(S) \supseteq \texttt{follow}\,(S)$	2
$B \rightarrow \texttt{b}\,S$	$\texttt{follow}\,(S) \supseteq \texttt{follow}\,(B)$	2
$C \rightarrow S\,\texttt{c}$	$\texttt{follow}\,(S) \supseteq \{\texttt{c}\}$	3
$\therefore$	$\texttt{follow}\,(B) = \{\texttt{a}, \texttt{b}, \texttt{c}, \$\}$	

Reconhecedor descendente preditivo
Tabela de decisão (*parsing table*)

- Um reconhecedor descendente preditivo para uma gramática $G = (T, N, P, S)$ 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:

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

Tabela de decisão
Exemplo #1

- Considere a gramatica

  $S \rightarrow \mathtt{a}\ S\ \mathtt{b}\ |\ \varepsilon$

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

  $\textbf{first}\,(\mathtt{a}\,S\,\mathtt{b}) = \{\mathtt{a}\}$

  $\therefore\ \textbf{predict}\,(S \rightarrow \mathtt{a}\,S\,\mathtt{b}) = \{\mathtt{a}\}$

  $\textbf{first}\,(\varepsilon) = \{\varepsilon\}$

  $\textbf{follow}\,(S) = \{\$, \mathtt{b}\}$

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

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 gramatica

  $S \rightarrow \varepsilon\ |\ \mathtt{a}\ B\ S\ |\ \mathtt{b}\ A\ S$

  $A \rightarrow \mathtt{a}\ |\ \mathtt{b}\ A\ A$

  $B \rightarrow \mathtt{a}\ B\ B\ |\ \mathtt{b}$

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

  $\textbf{predict}\,(S \rightarrow \mathtt{a}\,B\,S) = \{\mathtt{a}\}$

  $\textbf{predict}\,(S \rightarrow \mathtt{b}\,A\,S) = \{\mathtt{b}\}$

  $\textbf{predict}\,(S \rightarrow \varepsilon) = \{\$\}$

  $\textbf{predict}\,(A \rightarrow \mathtt{a}) = \{\mathtt{a}\}$

  $\textbf{predict}\,(A \rightarrow \mathtt{b}\,A\,A) = \{\mathtt{b}\}$

  $\textbf{predict}\,(B \rightarrow \mathtt{a}\,B\,B) = \{\mathtt{a}\}$

  $\textbf{predict}\,(B \rightarrow \mathtt{b}) = \{\mathtt{b}\}$

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 gramatica

$$S \rightarrow \varepsilon \mid \mathtt{a}\ B \mid \mathtt{b}\ A$$
$$A \rightarrow \mathtt{a}\ S \mid \mathtt{b}\ A\ A$$
$$B \rightarrow \mathtt{a}\ B\ B \mid \mathtt{b}\ S$$

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

$\textbf{predict}\,(S \rightarrow \mathtt{a}\,B) = \{\mathtt{a}\}$
$\textbf{predict}\,(S \rightarrow \mathtt{b}\,A) = \{\mathtt{b}\}$
$\textbf{predict}\,(S \rightarrow \varepsilon) = \{\mathtt{a}, \mathtt{b}, \$\}$
$\textbf{predict}\,(A \rightarrow \mathtt{a}\,S) = \{\mathtt{a}\}$
$\textbf{predict}\,(A \rightarrow \mathtt{b}\,A\,A) = \{\mathtt{b}\}$
$\textbf{predict}\,(B \rightarrow \mathtt{a}\,B\,B) = \{\mathtt{a}\}$
$\textbf{predict}\,(B \rightarrow \mathtt{b}\,S) = \{\mathtt{b}\}$

Tabela de decisão

	a	b	$
S	$\mathtt{a}\,B\,,\varepsilon$	$\mathtt{b}\,A\,,\varepsilon$	ε
A	a S	$\mathtt{b}\,A\,A$	
B	$\mathtt{a}\,B\,B$	$\mathtt{b}\,S$	

- As células $(S, \mathtt{a})$ e $(S, \mathtt{b})$ 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 gramatica

$$S \rightarrow \varepsilon$$
$$\mid\ \mathtt{a}\ S\ \mathtt{b}\ S$$
$$\mid\ \mathtt{b}\ S\ \mathtt{a}\ S$$

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

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

Tabela de decisão

	a	b	$
S	$\mathtt{a}\,A\,\mathtt{b}\,S\,,\varepsilon$	$\mathtt{b}\,S\,\mathtt{a}\,S\,,\varepsilon$	ε

- As células $(S, \mathtt{a})$ e $(S, \mathtt{b})$ 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 $\{\mathtt{i}, \mathtt{f}, \mathtt{v}, \mathtt{,}, \mathtt{;}\}$, a linguagem L_4 descrita pela gramática

$$\begin{array}{lcl} D & \to & T\ L\ \mathtt{;} \\ T & \to & \mathtt{i} \\ & | & \mathtt{f} \\ L & \to & \mathtt{v} \\ & | & \mathtt{v}\ \mathtt{,}\ L \end{array}$$

- 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

    $$\mathbf{predict}\,(L \to \mathtt{v}) = \{\,\mathtt{v}\,\}$$
    $$\mathbf{predict}\,(L \to \mathtt{v}\ \mathtt{,}\ L) = \{\,\mathtt{v}\,\}$$

    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.)

$$\begin{array}{lcl} D & \to & T\ L\ \mathtt{;} \\ T & \to & \mathtt{i} \\ & | & \mathtt{f} \\ L & \to & \mathtt{v}\ X \\ X & \to & \\ & | & \mathtt{,}\ L \end{array}$$

$\mathbf{predict}\,(D \to T\,L\,\mathtt{;}) = ?$
- $\mathbf{first}\,(T\,L\,\mathtt{;}) = ?$
- $\mathbf{first}\,(T) = \mathbf{first}\,(\mathtt{i}) \cup \mathbf{first}\,(\mathtt{f}) = \{\mathtt{i}\} \cup \{\mathtt{f}\}$
- $\therefore\ \mathbf{first}\,(T\,L\,\mathtt{;}) = \{\mathtt{i}, \mathtt{f}\}$
- $\therefore\ \mathbf{predict}\,(D \to T\,L\,\mathtt{;}) == \{\mathtt{i}, \mathtt{f}\}$

$\mathbf{predict}\,(T \to \mathtt{i}) = ?$
- $\mathbf{first}\,(\mathtt{i}) = \{\mathtt{i}\}$
- $\therefore\ \mathbf{predict}\,(T \to \mathtt{i}) = \{\mathtt{i}\}$

$\mathbf{predict}\,(T \to \mathtt{f}) = \{\mathtt{f}\}$

$\mathbf{predict}\,(L \to \mathtt{v}\,X) = ?$
- $\mathbf{first}\,(\mathtt{v}\,X) = \{\mathtt{v}\}$
- $\therefore\ \mathbf{predict}\,(L \to \mathtt{v}\,X) = \{\mathtt{v}\}$

$\mathbf{predict}\,(X \to \varepsilon) = ?$
- $\mathbf{first}\,(\varepsilon) = \{\varepsilon\}$
- $\therefore\ \mathbf{predict}\,(X \to \varepsilon) = \mathbf{follow}\,(X)$
- $\mathbf{follow}\,(X) = \mathbf{follow}\,(L) = \{\mathtt{;}\}$
- $\therefore\ \mathbf{predict}\,(X \to \varepsilon) = \{\mathtt{;}\}$

$\mathbf{predict}\,(X \to \mathtt{,}\,L) = \{\mathtt{,}\}$

Tabela de decisão
Exemplo #3 (cont.)

$$
\begin{array}{rcl}
D & \rightarrow & T\ L\ \texttt{;} \\
T & \rightarrow & \texttt{i} \\
 & | & \texttt{f} \\
L & \rightarrow & \texttt{v}\ X \\
X & \rightarrow & \\
 & | & \texttt{,}\ L
\end{array}
$$

$\textbf{predict}\,(D \rightarrow T\ L\ \texttt{;}) = \{\texttt{i}, \texttt{f}\}$
$\textbf{predict}\,(T \rightarrow \texttt{i}) = \{\texttt{i}\}$
$\textbf{predict}\,(T \rightarrow \texttt{f}) = \{\texttt{f}\}$
$\textbf{predict}\,(L \rightarrow \texttt{v}\ X) = \{\texttt{v}\}$
$\textbf{predict}\,(X \rightarrow \varepsilon) = \{\texttt{;}\}$
$\textbf{predict}\,(X \rightarrow \texttt{,}\,L) = \{\texttt{,}\}$

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