



Compiladores

Análise sintática descendente

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

DETI, Universidade de Aveiro

Ano letivo de 2022-2023

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

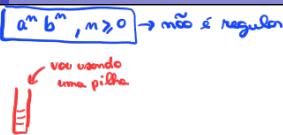
Introdução

↑ Não precisa de ser regular! „

- 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**

Modelo computacional

Reconhecem Gramáticas independentes de contexto



Análise sintática descendente

Exemplo

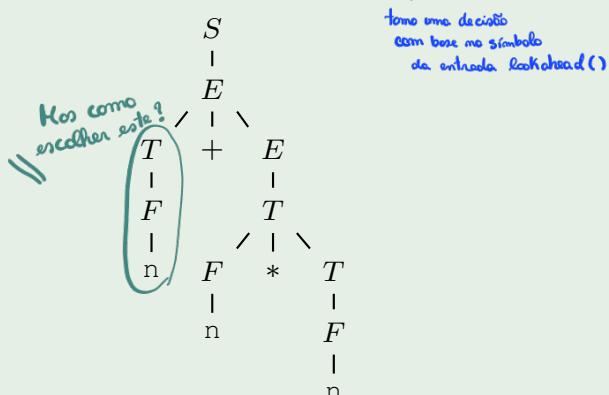
- Considere a gramática

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

Som operadores regulares!

=

- 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
 - Não requer *backtracking* !
 - A produção a aplicar a cada momento é escolhida com base no 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**

Analisador (parser) recursivo-descendente preditivo

Exemplo #1

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

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

descrita pela gramática

$$S \rightarrow a \underset{\sim}{S} b \mid \epsilon$$

↑ elemento

- 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(); Terminador
        eat('\n');
        printf("\n");
    }
    return 0;
}

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

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

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

void epsilon()
{
}

Tava a espera disto e apareceu isto...

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 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.

Analisador (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)

?

Analisador (parser) recursivo-descendente preditivo

Exemplo #2a

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

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

descrita pela gramática

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

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

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

- 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 $S \rightarrow \epsilon$

- Escolher $S \rightarrow \epsilon$ 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 $S \rightarrow \epsilon$

- É o que acontece com a entrada $bbbaa$ quando $E \dots$ (existem 2 produções quando lookahead = a)

$$S \Rightarrow b A \Rightarrow bb A A \Rightarrow bba S' A \Rightarrow \dots$$

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

Analisador (parser) recursivo-descendente preditivo

Exemplo #2b

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

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

descrita pela gramática

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

Ambiguidade!

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

- Tal como no caso anterior, escolher $S \rightarrow \epsilon$ quando lookahead for $\$$ pode não resolver

Analisador (parser) recursivo-descendente preditivo

Exemplo #3

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

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

descrita pela gramática

$$S \rightarrow \begin{cases} a & S b \\ | & a b \end{cases} \xrightarrow{\text{Como decidir?}} \begin{array}{l} \text{Por em evidência} \\ (\text{Poderia apenas aumentar}) \end{array} \xrightarrow{\text{Lookahead}} \begin{array}{l} S \rightarrow a X \\ X \rightarrow (s b | b) \end{array}$$

- 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 esqueda, transformando a gramática para

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

- Aumentar o número de símbolos de lookahead para 2

- se for aa , escolhe-se $S \rightarrow a S b$
- se for ab , escolhe-se $S \rightarrow a b$

Analisador (parser) recursivo-descendente preditivo

Exemplo #4

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

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

descrita pela gramática

$$S \rightarrow \begin{cases} S a b \\ | a b \end{cases}$$

Os parsers descendentes não aceitam recursividade à esquerda! Com lookahead de 1...

- 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
- Mas, 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** ($A \rightarrow \alpha$)

Resultado é um conjunto !!

Fatorização à esquerda

Exemplo de ilustração

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

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

descrita pela gramática

$$\begin{array}{rcl} S & \rightarrow & a \ S \ b \\ & | & a \ b \end{array} \Rightarrow \begin{array}{rcl} S & \rightarrow & a \ X \\ & & X \rightarrow S \ b \\ & & | \ b \end{array}$$

- Obtenha uma gramática equivalente, pondo em evidência o a
- Relaxando a definição standard de gramática que se tem usado, pode obter-se
Por em evidência \downarrow
 $S \rightarrow a (\underbrace{S \ b}_{\times} \mid b)$
- e criando um símbolo não terminal que represente o que está entre parêntesis, obtém-se a gramática

$$\begin{array}{rcl} S & \rightarrow & a \ X \\ X & \rightarrow & b \\ & | & S \ b \end{array} \quad \left. \begin{array}{l} \text{fazemos do lookahead } \underline{\text{?}} \end{array} \right.$$

- Esta gramática permite a construção de um programa preditivo com *lookahead* de

1

Eliminação de recursividade à esquerda

Recursividade direta simples

ANTLR

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

$$A \rightarrow A \alpha \\ | \beta$$

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

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

Pode ser 0!
Removemos a recursão

- Ou seja

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

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

$$A \rightarrow \beta \quad X \\ X \rightarrow \epsilon \\ | \alpha \quad X$$

ou $X \rightarrow \epsilon \mid X \alpha$ Fecho de α
 $X \rightarrow \epsilon \mid \alpha X$ Prefixo este!

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

Eliminação de recursividade à esquerda

Exemplo com recursividade direta simples

- Para a gramática

$$S \rightarrow S \ a \ b \\ | \ c \ b \ a$$

$$S \Rightarrow S \ a \ b \Rightarrow S \ a \ b \ a \ b \Rightarrow \dots \Rightarrow S \underbrace{a \ b \ \dots \ a \ b}_{m \geq 0}$$

ou seja, $S = c \ b \ a (a \ b)^m, m \geq 0 \Rightarrow X \rightarrow \epsilon \mid a \ b \ X$

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

$$S \rightarrow c \ b \ a \ X \\ X \rightarrow \epsilon \\ | \ a \ b \ X$$

Eliminação de recursividade à esquerda

Recursividade direta múltipla

Generalizada!

- 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 , representa genericamente a recursividade direta múltipla à esquerda

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_n$$

$$\mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

- Aplicando a estratégia anterior, tem-se

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

- Que corresponde à gramática

$$A \rightarrow \beta_1 X \mid \beta_2 X \mid \dots \mid \beta_m X$$

$$X \rightarrow \varepsilon$$

$$\mid \alpha_1 X \mid \alpha_2 X \mid \dots \mid \alpha_n X$$

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

Eliminação de recursividade à esquerda

Exemplo com recursividade direta múltipla

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

$$S \rightarrow S a b \mid S c$$
$$\mid b b \mid c c$$

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

- 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

$$S \rightarrow b b X \mid c c X$$
$$X \rightarrow (\varepsilon) \text{?}$$
$$\mid a b X \mid c X$$

↳ Recursividade à direita! //

Eliminação de recursividade à esquerda

Ilustração de recursividade indireta

Recursividade
Indireta

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

$$\begin{array}{l} S \rightarrow A \ a \mid b \\ A \rightarrow A \ c \mid S \ d \mid \varepsilon \end{array}$$

ANTER não resolve...

- O resultado seria

$$\begin{array}{l} S \rightarrow A \ a \mid b \\ A \rightarrow S \ d \ X \mid X \\ X \rightarrow \varepsilon \mid c \ X \end{array}$$

- A recursividade não foi eliminada

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

X Não funciona!

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

ordem

$$\begin{array}{l} A_1 \rightarrow \overbrace{\alpha_1 \mid \beta_1 \mid \dots \mid \Omega_1}^{\text{ordem}} \\ A_2 \rightarrow \alpha_2 \mid \beta_2 \mid \dots \mid \Omega_2 \\ \dots \\ A_n \rightarrow \alpha_n \mid \beta_n \mid \dots \mid \Omega_n \end{array}$$

- Algoritmo:

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

A_1, A_2, \dots, A_n , *Assume ordem*

- 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

$A_2 \rightarrow A_3 \mid \dots \mid A_m$ { Mas não $A_2 \in A_1$ }

Eliminação de recursividade à esquerda

Exemplo com recursividade indireta

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

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

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

problema!

- S pode começar por A mas não por S
- A não pode começar por A nem S



- 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

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

$$A \rightarrow \underline{A} \ c \mid \overbrace{A \ a \ d}^{\text{Falta } \text{término } A \text{ mas é igual a recursividade direta!}} \mid b \ d \mid \epsilon$$

A ideia é: Substituir até ser possível aplicar ...

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

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

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

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



Conjuntos predict, first e follow

Definições

Para o teste!

lookahead = 1

- 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:

predict ($A \rightarrow \alpha$) =

$$\begin{cases} \text{first}(\alpha) \\ (\text{first}(\alpha) - \{\epsilon\}) \cup \text{follow}(A) \end{cases}$$

Como escolher?

Possíveis descendentes,

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

$$\text{first}(\alpha) = \{t \in T : \alpha \xrightarrow{*} t \omega \wedge \omega \in T^*\} \cup \{\epsilon : \alpha \xrightarrow{*} \epsilon\}$$

Quais são os first terminais de α , terminal

Se for possível

IMPORTANTÉ

- O conjunto **follow** (A) representa as letras (símbolos terminais) que podem aparecer imediatamente à frente de A numa derivação. Define-se por:

$$S \xrightarrow{*} A \alpha$$

é este, imediatamente à direita

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

Incluindo o \\$!
precisamos pois abab \\$
(\\$ abab) → então coloco a por \\$\\$

item adicional

Conjunto first

Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

```
first( $\alpha$ ) {
    if ( $\alpha = \varepsilon$ ) then
        return  $\{\varepsilon\}$ 
    else if ( $h \in T$ ) then
        return  $\{h\}$  → Se for terminal!
    else
        return  $\bigcup_{(h \rightarrow \beta_i) \in P} \text{first}(\beta_i \omega)$  → concatenação de  $\beta_i$  com  $\omega$ 
        } → união
        Se for não terminal!
```

primeiro símbolo terminal ou não
→ O resto da palavra!

- 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

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

$$\begin{aligned} \text{first}(BC) &= \text{first}(\varepsilon C) \cup \text{first}(bSC) \\ &= \text{first}(C) \cup \{b\} \\ \text{first}(C) &= \text{first}(c) \cup \text{first}(cS) \\ &= \{c\} \cup \text{first}(cS) \\ &= \{c\} \\ \therefore \text{first}(BC) &= \{c, b\} \end{aligned}$$

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

- Porque aS começa pelo símbolo terminal a

$$\text{first}(aS) = \{a\}$$

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

- Porque BC começa pelo símbolo não terminal B

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

- Porque C começa pelo símbolo não terminal C

$$\text{first}(C) = \text{first}(c) \cup \text{first}(cS)$$

$$\therefore \text{first}(BC) = \text{first}(c) \cup \text{first}(cS) \cup \text{first}(bSC) = \{b, c\}$$

dct **dct** **tbt**

- Note que, embora B se possa transformar em ε , $\varepsilon \notin \text{first}(BC)$

- Por essa razão, $\text{first}(BC) \neq \text{first}(B) \cup \text{first}(C)$

IMPORTANTE!

Conjunto **first**

Exemplo #2

- Considere a gramática

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

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

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

Agora é diferente! *Concatenação (Ambos podem \oplus)*

$$\begin{aligned} \text{first}(BC) &= \text{first}(c) \cup \text{first}(bSC) \\ &= \text{first}(c) \cup \{b\} \\ \text{first}(c) &= \text{first}(\epsilon) \cup \text{first}(cS) \\ &= \{\epsilon\} \cup \{c\} \\ &= \{\epsilon, c\} \\ \therefore \text{first}(BC) &= \{\epsilon, b, c\} \end{aligned}$$

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

- Porque BC começa pelo símbolo não terminal B

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

- Porque C começa pelo símbolo não terminal C

$$\text{first}(C) = \text{first}(\epsilon) \cup \text{first}(cS)$$

$$\text{first}(BC) = \text{first}(\epsilon) \cup \text{first}(cS) \cup \text{first}(bSC)$$

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



- Note que a gramática não é a mesma

Faz sentido...

- Note que $\epsilon \in \text{first}(BC)$ apenas porque todo o BC se pode transformar em ϵ !

Conjunto **follow**

Algoritmo de cálculo

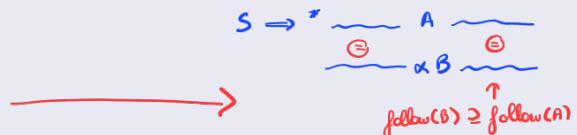
- Os conjuntos **follow** podem ser calculados através de um algoritmo iterativo envolvendo todos os símbolos não terminais
- Aplicam-se as seguintes regras:

$$1 \quad \$ \in \text{follow}(S)$$

metanegros até que numa delas nada acontece!

$$2 \quad \text{if } (A \rightarrow \alpha B \in P) \text{ then}$$

$$\text{follow}(B) \supseteq \text{follow}(A)$$



$$3 \quad \text{if } (A \rightarrow \alpha B \beta \in P) \wedge (\epsilon \notin \text{first}(\beta)) \text{ then}$$

$$\text{follow}(B) \supseteq \text{first}(\beta)$$

$$4 \quad \text{if } (A \rightarrow \alpha B \beta \in P) \wedge (\epsilon \in \text{first}(\beta)) \text{ then}$$

$$\text{follow}(B) \supseteq ((\text{first}(\beta) - \{\epsilon\}) \cup \text{follow}(A))$$

Retirando o teta

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

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

$A \supseteq B \rightarrow A \text{ contém } B$

Conjunto follow

Exemplo #1

- Considere a gramática

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

$$\begin{aligned} \text{first}(C) &= \text{first}(c) \cup \text{first}(cS) = \{c\} \quad t \notin \text{first}(c) \\ \text{Logo: } \text{follow}(B) &\supseteq \text{first}(c) \\ \text{Não havendo mais contribuições:} \\ \therefore \text{follow}(B) &= \{c\} \end{aligned}$$

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

- Procuram-se ocorrências de B no lado direito das produções. Há uma: BC
- A produção $S \rightarrow BC$ encaixa nas regras 3 ou 4, 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

$$\text{follow}(B) = \{c\}$$

Conjunto follow

Exemplo #2

- Considere a gramática

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

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

- A produção $S \rightarrow BC$ encaixa nas regras 3 ou 4, dependendo de o ϵ pertencer ou não ao **first**(C)
- **first**(C) = { ϵ , c}
- $\therefore \text{follow}(B) \supseteq ((\text{first}(C) - \{\epsilon\}) \cup \text{follow}(S))$ [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 $\text{follow}(S) \supseteq \text{follow}(S)$
- A produção $B \rightarrow b \ S$ diz que $\text{follow}(S) \supseteq \text{follow}(B)$
- A produção $C \rightarrow c \ S$ diz que $\text{follow}(S) \supseteq \text{follow}(C)$
- A produção $S \rightarrow BC$ diz que $\text{follow}(C) \supseteq \text{follow}(S)$ | $\text{follow}(S) = \text{follow}(C)$
- Pelas contribuições tem-se que

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

- Também se ficou a saber que $\text{follow}(S) = \text{follow}(B) = \text{follow}(C)$

Conjunto follow

Exemplo #3

- Considera a gramática

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

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

- A produção $S \rightarrow BC$ encaixa nas regras 3 ou 4, dependendo de o ϵ pertencer ou não ao $\text{first}(C)$
- $\text{first}(C) = \{\epsilon, a, b\}$
- $\therefore \text{follow}(B) \supseteq (\text{first}(C) - \{\epsilon\}) \cup \text{follow}(S)$
- Porque S é o símbolo inicial, $\$ \in \text{follow}(S)$
- A produção $S \rightarrow aS$ é irrelevante, porque diz que $\text{follow}(S) \supseteq \text{follow}(S)$
- A produção $B \rightarrow bS$ diz que $\text{follow}(S) \supseteq \text{follow}(B)$
- A produção $C \rightarrow Sc$ 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

$$\begin{aligned} \text{first}(C) &= \text{first}(\epsilon) \cup \text{first}(Sc) \\ \text{first}(Sc) &= \text{first}(aSc) \cup \text{first}(BSc) \\ &= \{a\} \cup \text{first}(BSc) \cup \text{first}(bSc) \\ &= \{a, b\} \end{aligned}$$

$$\therefore \text{follow}(C) = \{\epsilon, a, b\} \Rightarrow \epsilon \in \text{follow}(C)$$

Assim, pela regra 4:

$$\text{follow}(B) \supseteq (\text{follow}(C) - \{\epsilon\}) \cup \text{follow}(S)$$

Como S é o símbolo inicial: $\$ \in \text{follow}(S)$

$$\text{follow}(S) \supseteq \text{follow}(B)$$

$$\text{follow}(S) \supseteq \text{follow}(C) = \text{follow}(Sc) = \{c\}$$

$$\therefore \text{follow}(S) = \{a, b, c\} \Rightarrow \text{follow}(B) = \{a, b, c, \$\}$$

O slide tem diferenças
e Learning...

Reconhecedor descendente preditivo

Tabela de decisão (parsing table)

Sam backtracking!

- Para uma gramática $G = (T, N, P, S)$ e um lookahead de 1, o reconhecedor descendente pode basear-se numa tabela de decisão
- 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 subconjuntos 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  $\tau(A, t)$ 
  
```

• Para cada transição

Tabela de decisão

Exemplo #1

- Considera a gramática

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

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

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

$$\therefore \text{predict}(S \rightarrow a S b) = \{a\} \quad \text{E} \not\in \text{first}(a S b)$$

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

$$\text{follow}(S) = \{\$, b\} \quad \epsilon \in \text{first}(\epsilon)$$

$$\therefore \text{predict}(S \rightarrow \epsilon) = \{\$\}, b\} \Rightarrow \text{predict}(S \rightarrow \epsilon) = (\text{first}(\epsilon) - \{\epsilon\}) \cup \text{follow}(S)$$

Tabela de decisão

	a	b	\$
S	a S b	ϵ	ϵ

se possível construir
um reconhecedor
com lookahead = 1

- 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

- Considera a gramática

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

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

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

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

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

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

$$\text{predict}(S \rightarrow \epsilon) = \{\$\}$$

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

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

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

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

Tabela de decisão

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

"O Rosen diz logo
que a entrada é inválida!"

- As células vazias correspondem a situações de erro

Tabela de decisão

Exemplo #2a

- Considera a gramática

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

$$A \rightarrow a S \mid b A A \quad \leftarrow \text{é ambígua!}$$

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

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

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

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

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

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

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

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

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

Dependendo do que vem a seguir...

Passar preditivo
com lookahead = 1
não é possível
?

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

- Considera a gramática

$$S \rightarrow \epsilon$$

$$\mid a S b S$$

$$\mid b S a S$$

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

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

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

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

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

$$\begin{array}{l}
 D \rightarrow T \ L \ ; \\
 T \rightarrow i \\
 | \quad f \\
 L \rightarrow v \\
 | \quad v \ \textcolor{red}{O} \ L \ \text{vado a llamarlo!} \\
 \end{array}
 \quad \left\{ \begin{array}{l} \text{Noz aqui posso} \\ \text{fatorizar} \end{array} \right. \Rightarrow \begin{array}{l} \text{Tentou de alterar a gramática?} \\ \text{L} \rightarrow v(\underbrace{\varepsilon \mid}_{X}, L) \quad L \rightarrow \textcolor{blue}{v} \ X \xrightarrow{\text{predict}} \\ \quad X \rightarrow \varepsilon \mid \textcolor{red}{L} \end{array}$$

- Obtenha-se uma tabela de decisão de um reconhecedor descendente, com lookahead de 1, que reconheça a linguagem L_4 .
 - Pretende-se que, se necessário, se transforme a gramática numa equivalente que seja LL(1)
 - Neste caso, existem produções com prefixos comuns (os conjuntos **predict** não são disjuntos)
 - Antes de calcular os conjuntos **predict** é necessário começar por fatorizar à esquerda, por causa das produções com cabeça L

Tabela de decisão

Exemplo #3 (cont.)

```

predict ( $D \rightarrow TL ;$ ) = ?
  first ( $TL ;$ ) = ?
  first ( $T$ ) = first (i)  $\cup$  first (f) = {i}  $\cup$  {f}
 $\therefore$  first ( $TL ;$ ) = {i, f}
 $\therefore$  predict ( $D \rightarrow TL ;$ ) == {i, f}

```

$D \rightarrow T \ L ;$	$\text{predict}(T \rightarrow i) = ?$
$T \rightarrow i$	$\text{first}(i) = \{i\}$
f	$\therefore \text{predict}(T \rightarrow i) = \{i\}$
$L \rightarrow v \ X$	$\text{predict}(T \rightarrow f) = \{f\}$
$X \rightarrow$	$\text{predict}(L \rightarrow v \ X) = ?$
L	$\text{first}(v \ X) = \{v\}$
	$\therefore \text{predict}(L \rightarrow v \ X) = \{v\}$
	$\text{predict}(X \rightarrow _) = ?$

```

predict( $X \rightarrow \varepsilon$ ) = ?
  first( $\varepsilon$ ) = { $\varepsilon$ }
 $\therefore \text{predict}(X \rightarrow \varepsilon) = \text{follow}(X)$ 
follow( $X$ ) = follow( $L$ ) = {;}
 $\therefore \text{predict}(X \rightarrow \varepsilon) = \{;\}$ 
predict( $X \rightarrow , L$ ) = {,}

```

$$\begin{array}{l}
 D \rightarrow T L ; \\
 T \rightarrow i \\
 | f \\
 L \rightarrow v \\
 | v, L
 \end{array}
 \quad \left\{
 \begin{array}{l}
 \text{Gramática} \\
 \text{que seja LL(0)} \\
 \downarrow \\
 \begin{array}{l}
 = L \rightarrow v X \\
 X \rightarrow \epsilon \\
 |, L
 \end{array}
 \end{array}
 \right.$$

Gramática
que seja LL(0)

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

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

$$\begin{aligned}
 \text{first}(TL;) &= \text{first}(iL;) \cup \text{first}(fL;) \\
 &= \{i, f\}, \epsilon \notin \{i, f\}
 \end{aligned}$$

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

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

$$\begin{aligned}
 \text{first}(vX) &= \{v\}, \epsilon \notin \{v\} \\
 \therefore \text{predict}(L \rightarrow vX) &= \{v\}
 \end{aligned}$$

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

$$\begin{aligned}
 \text{first}(i) &= \{i\}, \epsilon \notin \{i\} \\
 \therefore \text{predict}(T \rightarrow i) &= \{i\}
 \end{aligned}$$

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

$$\begin{aligned}
 \text{first}(f) &= \{f\}, \epsilon \notin \{f\} \\
 \therefore \text{predict}(T \rightarrow f) &= \{f\}
 \end{aligned}$$

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

$$\begin{aligned}
 \text{first}(\epsilon) &= \{\epsilon\}, \epsilon \in \{\epsilon\} \\
 \text{follow}(X) &\supseteq \text{follow}(L) = \{;\} \\
 \text{única} \Rightarrow \text{follow}(X) &= \{;\} \\
 \therefore \text{predict}(X \rightarrow \epsilon) &= \{;\}
 \end{aligned}$$

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

$$\begin{aligned}
 \text{first}(,L) &= \{,\} \\
 \therefore \text{predict}(X \rightarrow , L) &= \{,\}
 \end{aligned}$$

Tabela de decisão

Exemplo #3 (cont.)

$D \rightarrow T \ L \ ;$

$T \rightarrow i$

| f

$L \rightarrow v \ X$

$X \rightarrow$

| , L

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

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

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

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

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

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

Tabela de decisão

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

→ Gramática do tipo
LL(1)

Para o teste!

} • Preencher tabelas
 } • Predicts

Não esquecer!

- As células vazias são situações de erro