Compiladores

Gramáticas livres de contexto

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

DETI, Universidade de Aveiro

Ano letivo de 2022-2023

Sumário

Gramáticas
Definição

Uma gramática é um quádruplo $G = (T, N, P, S)$, onde

- T é um conjunto finito não vazio de símbolos **terminais**;
- N, com $N \cap T = \emptyset$, é um conjunto finito não vazio de símbolos **não terminais**;
- P é um conjunto de **produções** (ou regras de rescrita), cada uma da forma $\alpha \rightarrow \beta$;
- $S \in N$ é o símbolo inicial.

- α e β são designados por **cabeça da produção** e **corpo da produção**, respetivamente.
- No caso geral $\alpha \in (N \cup T)^* \times N \times (N \cup T)^*$ e $\beta = (N \cup T)^*$.
- Em ANTLR:
  - os terminais são representados por ids começados por letra maíscula
  - os não terminais são representados por ids começados por letra minúscula

Gramáticas livres de contexto – GLC
Definição

$\mathcal{D}$ Uma gramática $G = (T, N, P, S)$ diz-se **livre de contexto** (ou **independente do contexto**) se, para qualquer produção $(\alpha \rightarrow \beta) \in P$, as duas condições seguintes são satisfeitas

$$\alpha \in N$$
$$\beta \in (T \cup N)^*$$

- A linguagem gerada por uma gramática livre de contexto diz-se livre de contexto
- As gramáticas regulares são livres de contexto
- As gramáticas livres de contexto são fechadas sob as operações de reunião, concatenação e fecho
  - **mas não o são** sob as operações de intersecção e complementação.

- Note que: se $\beta \in T^* \cup T^*N$, então $\beta \in (T \cup N)^*$

Derivação
Exemplo

$\mathcal{Q}$ Considere, sobre o alfabeto $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, a gramática

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

e transforme o símbolo inicial S na palavra $\mathtt{aabcbc}$ por aplicação sucessiva de produções da gramática

$\mathcal{R}$

$$S \Rightarrow \mathtt{a}B \Rightarrow \mathtt{aa}BB \Rightarrow \mathtt{aab}SB \Rightarrow \mathtt{aabc}SB \Rightarrow \mathtt{aabc}B \Rightarrow \mathtt{aabcb}S \Rightarrow \mathtt{aabcbc}S \Rightarrow \mathtt{aabcbc}$$

- Acabou de se obter uma **derivação à esquerda** da palavra $\mathtt{aabcbc}$
- Cada passo dessa derivação é uma **derivação direta à esquerda**

- Quando há dois ou mais símbolos não terminais, opta-se por expandir primeiro o mais à esquerda

Derivação
Definições

$\mathcal{D}$ Dada uma palavra $\alpha A \beta$, com $A \in N$ e $\alpha, \beta \in (N \cup T)^*$, e uma produção $(A \to \gamma) \in P$, com $\gamma \in (N \cup T)^*$, chama-se **derivação direta** à rescrita de $\alpha A \beta$ em $\alpha \gamma \beta$, denotando-se

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

$\mathcal{D}$ Dada uma palavra $\alpha A \beta$, com $A \in N$, $\alpha \in T^*$ e $\beta \in (N \cup T)^*$, e uma produção $(A \to \gamma) \in P$, com $\gamma \in (N \cup T)^*$, chama-se **derivação direta à esquerda** à rescrita de $\alpha A \beta$ em $\alpha \gamma \beta$, denotando-se

$$\alpha A \beta \overset{E}{\Rightarrow} \alpha \gamma \beta$$

$\mathcal{D}$ Dada uma palavra $\alpha A \beta$, com $A \in N$, $\alpha \in (N \cup T)^*$ e $\beta \in T^*$, e uma produção $(A \to \gamma) \in P$, com $\gamma \in (N \cup T)^*$, chama-se **derivação direta à direita** à rescrita de $\alpha A \beta$ em $\alpha \gamma \beta$, denotando-se

$$\alpha A \beta \overset{D}{\Rightarrow} \alpha \gamma \beta$$

Derivação
Definições

$\mathcal{D}$ Chama-se **derivação** a uma sucessão de zero ou mais derivações diretas, denotando-se

$$\alpha \Rightarrow^* \beta \qquad \equiv \qquad \alpha = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_n = \beta$$

onde n é o comprimento da derivação.

$\mathcal{D}$ Chama-se **derivação à esquerda** a uma sucessão de zero ou mais derivações diretas à esquerda, denotando-se

$$\alpha \overset{E}{\Rightarrow}{}^* \beta \qquad \equiv \qquad \alpha = \alpha_0 \overset{E}{\Rightarrow} \alpha_1 \overset{E}{\Rightarrow} \cdots \overset{E}{\Rightarrow} \alpha_n = \beta$$

onde n é o comprimento da derivação.

$\mathcal{D}$ Chama-se **derivação à direita** a uma sucessão de zero ou mais derivações diretas à direita, denotando-se

$$\alpha \overset{D}{\Rightarrow}{}^* \beta \qquad \equiv \qquad \alpha = \gamma_0 \overset{D}{\Rightarrow} \gamma_1 \overset{D}{\Rightarrow} \cdots \overset{D}{\Rightarrow} \gamma_n = \beta$$

onde n é o comprimento da derivação.

Derivação
Exemplo

$\mathcal{Q}$ Considere, sobre o alfabeto $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, a gramática seguinte

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

Determine as derivações à esquerda e à direita da palavra $\mathtt{aabcbc}$

$\mathcal{R}$

à esquerda

$$S \Rightarrow \mathtt{a}B \Rightarrow \mathtt{aa}BB \Rightarrow \mathtt{aab}SB \Rightarrow \mathtt{aabc}SB$$
$$\Rightarrow \mathtt{aabc}B \Rightarrow \mathtt{aabcb}S \Rightarrow \mathtt{aabcbc}S \Rightarrow \mathtt{aabcbc}$$

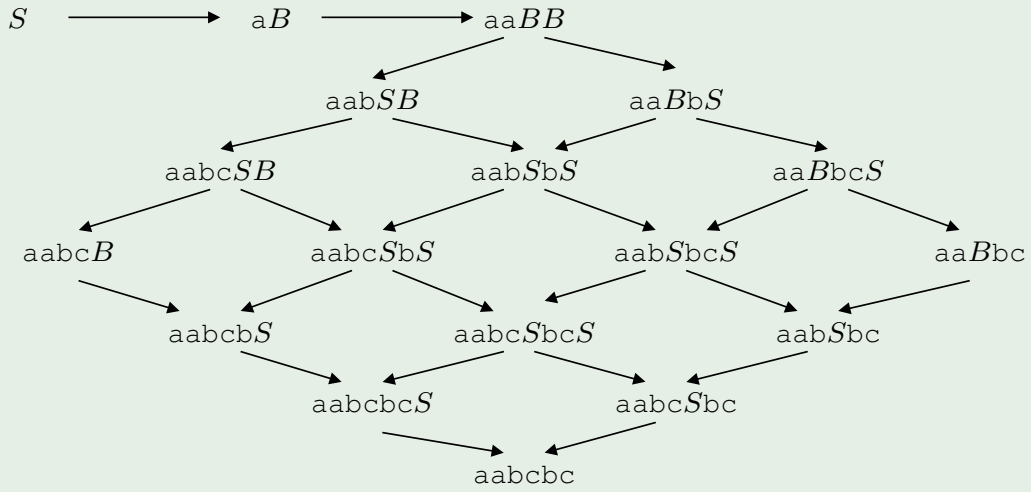
à direita

$$S \Rightarrow \mathtt{a}B \Rightarrow \mathtt{aa}BB \Rightarrow \mathtt{aa}B\mathtt{b}S \Rightarrow \mathtt{aa}B\mathtt{bc}S$$
$$\Rightarrow \mathtt{aa}B\mathtt{bc} \Rightarrow \mathtt{aab}S\mathtt{bc} \Rightarrow \mathtt{aabc}S\mathtt{bc} \Rightarrow \mathtt{aabcbc}$$

- Note que se usou $\Rightarrow$ em vez de $\overset{D}{\Rightarrow}$ e $\overset{E}{\Rightarrow}$

Derivação
Alternativas de derivação

- O grafo seguinte capta as alternativas de derivação. Considera-se novamente a palavra `aabcbc` e a gramática anterior

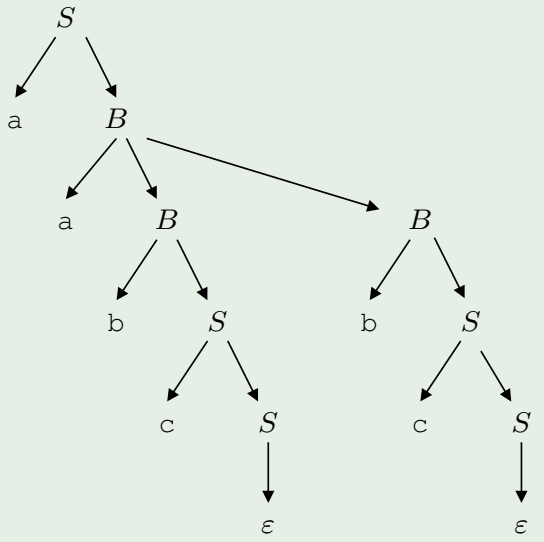


- Identifique os caminhos que correspondem às derivações à direita e à esquerda

Derivação
Árvore de derivação

$\mathcal{D}$ Uma **árvore de derivação** (*parse tree*) é uma representação de uma derivação onde os nós-ramos são símbolos não terminais e os nós-folhas são símbolos terminais

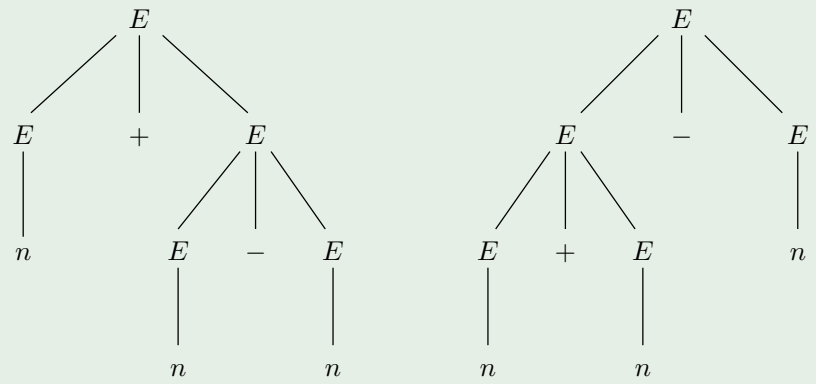
- A árvore de derivação da palavra `aabcbc` na gramática anterior é



Ambiguidade
Ilustração através de um exemplo

- Considere a gramática $S \rightarrow S \texttt{ + } S \mid S \texttt{ - } S \mid \texttt{ (} S \texttt{) } \mid \texttt{n}$
  e desenhe a árvore de derivação da palavra `n+n-n`

$\mathcal{R}$ Podem obter-se duas árvores de derivação diferentes



- Pode haver duas interpretações diferentes para a palavra; há **ambiguidade**

Ambiguidade
Definição

$\mathcal{D}$ Diz-se que uma palavra é derivada **ambiguamente** se possuir duas ou mais árvores de derivação distintas

$\mathcal{D}$ Diz-se que uma gramática é **ambígua** se possuir pelo menos uma palavra gerada ambiguamente

- Frequentemente é possível definir-se uma gramática não ambígua que gera a mesma linguagem que uma ambígua
- No entanto, há gramáticas **inerentemente ambíguas**

  Por exemplo, a linguagem

  $$L = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \vee j = k\}$$

  não possui uma gramática não ambígua que a represente.

Ambiguidade
Remoção da ambiguidade

$\mathcal{R}$ Considere-se novamente a gramática

$$S \rightarrow S \texttt{ + } S \mid S \texttt{ - } S \mid \texttt{(} S \texttt{)} \mid \texttt{n}$$

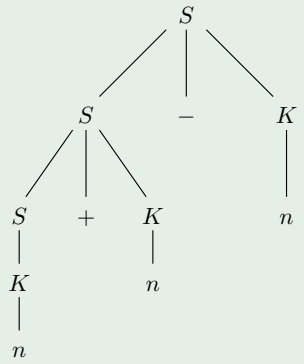
e obtenha-se uma gramática não ambígua equivalente

$\mathcal{R}$

$$S \rightarrow K \mid S \texttt{ + } K \mid S \texttt{ - } K$$

$$K \rightarrow \texttt{n} \mid \texttt{(} S \texttt{)}$$

$\mathcal{Q}$ Desenhe a árvore de derivação da palavra `n+n-n` na nova gramática



Projeto de gramáticas
Exemplo #1, solução #1

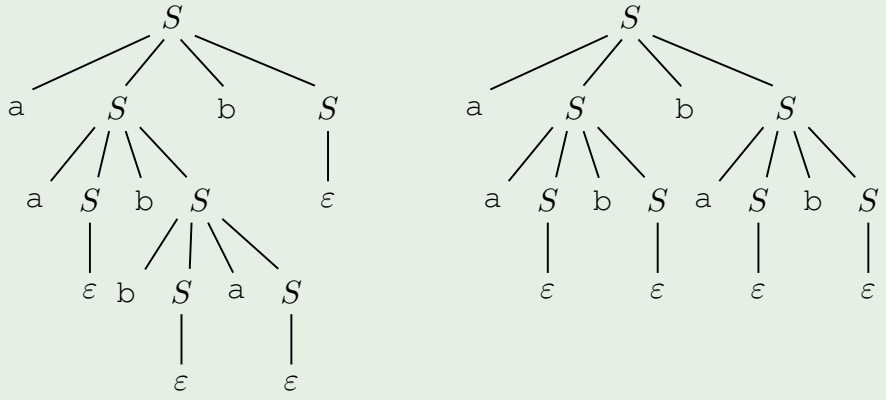
$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\texttt{a}, \texttt{b}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_1 = \{\omega \in T^* : \#(\texttt{a}, \omega) = \#(\texttt{b}, \omega)\}$$

$\mathcal{R}_1$

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

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra `aabbab`



Projeto de gramáticas
Exemplo #1, solução #2

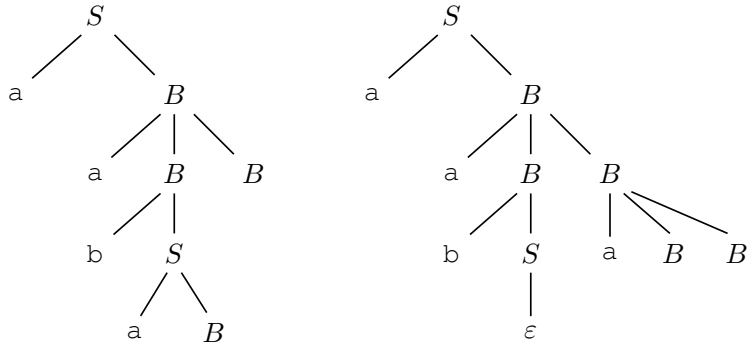
$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_1 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega)\}$$

$\mathcal{R}_2$

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

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra `aababb`.



- Falta expandir alguns nós

Projeto de gramáticas
Exemplo #1, solução #3

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_1 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega)\}$$

$\mathcal{R}_3$

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

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra `aababb`

Projeto de gramáticas
Exemplo #2

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_2 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega)\}$$

$\mathcal{R}$

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

$\mathcal{Q}$ A gramática é ambígua?

Projeto de gramáticas
Exemplo #3, solução #1

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_3 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) \wedge \forall_{i \leq |\omega|}\ \#(\mathtt{a}, \text{prefix}(i, \omega)) \geq \#(\mathtt{b}, \text{prefix}(i, \omega))\}$$

$\mathcal{R}_1$

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

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra $\mathtt{aababb}$

- O número de ocorrências das letras $\mathtt{a}$ e $\mathtt{b}$ é igual, mas em qualquer prefixo das palavras da linguagem não pode haver mais $\mathtt{b}$s que $\mathtt{a}$s, ou seja o $\mathtt{a}$ aparece antes
- Solução inspirada na do exemplo 1.1, removendo a produção $S \to \mathtt{b}\ S\ \mathtt{a}\ S$

Projeto de gramáticas
Exemplo #3: solução #2

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_3 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) \wedge \forall_{i \leq |\omega|} \#(\mathtt{a}, \text{prefix}(i, \omega)) \geq \#(\mathtt{b}, \text{prefix}(i, \omega))\}$$

$\mathcal{R}_2$

$$S \rightarrow \varepsilon \mid \mathtt{a}\ B \mid \mathtt{c}\ S$$

$$B \rightarrow \mathtt{a}\ B\ B \mid \mathtt{b}\ S \mid \mathtt{c}\ B$$

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra `aababb`

- Solução inspirada na do exemplo 1.2, removendo a produção $S \rightarrow \mathtt{b}\ A$ e as começadas por A

Projeto de gramáticas
Exemplo #3: solução #3

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L_3 = \{\omega \in T^* : \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) \wedge \forall_{i \leq |\omega|} \#(\mathtt{a}, \text{prefix}(i, \omega)) \geq \#(\mathtt{b}, \text{prefix}(i, \omega))\}$$

$\mathcal{R}_3$

$$S \rightarrow \varepsilon \mid \mathtt{a}\ B\ S \mid \mathtt{c}\ S$$

$$B \rightarrow \mathtt{a}\ B\ B \mid \mathtt{b} \mid \mathtt{c}\ B$$

$\mathcal{Q}$ A gramática é ambígua? Analise a palavra `aababb`

- Solução inspirada na do exemplo 1.3, removendo a produção $S \rightarrow \mathtt{b}\ A\ S$ e as começadas por A

Projeto de gramáticas
Exercício

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{(}, \texttt{)}, +, \star\}$, determine uma gramática independente do contexto que represente a linguagem

$$L = \{\, \omega \in T^* \;:\; \omega \text{ representa uma expressão regular sobre o alfabeto } \{\texttt{a}, \texttt{b}, \texttt{c}\}\}$$

$\mathcal{R}$ Em ANTLR, poder-se-ia fazer

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E\ \texttt{'*'} \\ &\mid E\ E \\ &\mid E\ \texttt{'+'}\ E \\ &\mid \texttt{'('}\ E\ \texttt{')'} \\ &\mid \texttt{'a'} \mid \texttt{'b'} \mid \texttt{'c'} \end{aligned}$$

mas em geral não, porque, em geral, as alternativas estão todas ao mesmo nível

- Como escrever a gramática de modo à precedência ser imposta por construção?

- Está a usar-se o operador + em vez do |

Projeto de gramáticas
Exercício (cont.)

$\mathcal{R}$ Em geral

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E\ \texttt{'+'}\ T \\ &\mid T \\ T &\rightarrow T\ F \\ &\mid F \\ F &\rightarrow F\ \texttt{'*'} \\ &\mid O \\ O &\rightarrow \texttt{'('}\ E\ \texttt{')'} \\ &\mid \texttt{'a'} \mid \texttt{'b'} \mid \texttt{'c'} \end{aligned}$$

- Uma expressão é vista como uma 'soma' de termos
- Um termo é visto como um 'produto' (concatenação) de fatores
- Um fator é visto como um 'fecho' de operandos
- Um operando ou é um elemento base ou uma expressão entre parêntesis

---

- Está a usar-se o operador + em vez do |

Reunião de GLC
Exemplo

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

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

$\mathcal{R}$

$L_1 = \{\, \omega \in T^* \,:\, \#(\mathtt{a},\omega) = \#(\mathtt{b},\omega) \,\}$	$S_1 \to \varepsilon \mid \mathtt{a}\ S_1\ \mathtt{b}\ S_1 \mid \mathtt{b}\ S_1\ \mathtt{a}\ S_1 \mid \mathtt{c}\ S_1$
$L_2 = \{\, \omega \in T^* \,:\, \#(\mathtt{a},\omega) = \#(\mathtt{c},\omega) \,\}$	$S_2 \to \varepsilon \mid \mathtt{a}\ S_2\ \mathtt{c}\ S_2 \mid \mathtt{b}\ S_2 \mid \mathtt{c}\ S_2\ \mathtt{a}\ S_2$
$L = L_1 \cup L_2$	$S \to S_1 \mid S_2$ $S_1 \to \varepsilon \mid \mathtt{a}\ S_1\ \mathtt{b}\ S_1 \mid \mathtt{b}\ S_1\ \mathtt{a}\ S_1 \mid \mathtt{c}\ S_1$ $S_2 \to \varepsilon \mid \mathtt{a}\ S_2\ \mathtt{c}\ S_2 \mid \mathtt{b}\ S_2 \mid \mathtt{c}\ S_2\ \mathtt{a}\ S_2$

- Para esta linguagem, mesmo que as gramáticas de L_1 e L_2 não sejam ambíguas, a de L será ambígua. Porquê?

Operações sobre GLCs
Reunião

$\mathcal{D}$ Sejam $G_1 = (T_1, N_1, P_1, S_1)$ e $G_2 = (T_2, N_2, P_2, S_2)$ duas gramáticas livres de contexto quaisquer, com $N_1 \cap N_2 = \emptyset$.

A gramática $G = (T, N, P, S)$ onde

$$\begin{aligned} T &= T_1 \cup T_2 \\ N &= N_1 \cup N_2 \cup \{S\} \quad \text{com} \quad S \notin (N_1 \cup N_2) \\ P &= \{S \to S_1, S \to S_2\} \cup P_1 \cup P_2 \end{aligned}$$

é livre de contexto e gera a linguagem $L = L(G_1) \cup L(G_2)$

- As novas produções $S \to S_i$, com $i = 1, 2$, permitem que G gere a linguagem $L(G_i)$
- Esta definição é idêntica à que foi dada para a operação de reunião nas gramáticas regulares

Concatenação de GLC
Exemplo

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

$$L = \{\, \omega_1\omega_2 \;:\; \omega_1, \omega_2 \in T^* \\ \wedge \#(\mathtt{a}, \omega_1) = \#(\mathtt{b}, \omega_1) \;\wedge\; \#(\mathtt{a}, \omega_2) = \#(\mathtt{c}, \omega_2) \,\}$$

$\mathcal{R}$

$L_1 = \{\, \omega \in T^* \;:\; \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) \,\}$	$S_1 \to \varepsilon \mid \mathtt{a}\ S_1\ \mathtt{b}\ S_1 \mid \mathtt{b}\ S_1\ \mathtt{a}\ S_1 \mid \mathtt{c}\ S_1$
$L_2 = \{\, \omega \in T^* \;:\; \#(\mathtt{a}, \omega) = \#(\mathtt{c}, \omega) \,\}$	$S_2 \to \varepsilon \mid \mathtt{a}\ S_2\ \mathtt{c}\ S_2 \mid \mathtt{b}\ S_2 \mid \mathtt{c}\ S_2\ \mathtt{a}\ S_2$
$L = L_1 \cdot L_2$	$S \to S_1\ S_2$ $S_1 \to \varepsilon \mid \mathtt{a}\ S_1\ \mathtt{b}\ S_1 \mid \mathtt{b}\ S_1\ \mathtt{a}\ S_1 \mid \mathtt{c}\ S_1$ $S_2 \to \varepsilon \mid \mathtt{a}\ S_2\ \mathtt{c}\ S_2 \mid \mathtt{b}\ S_2 \mid \mathtt{c}\ S_2\ \mathtt{a}\ S_2$

Operações sobre gramáticas:
Concatenação

$\mathcal{D}$ Sejam $G_1 = (T_1, N_1, P_1, S_1)$ e $G_2 = (T_2, N_2, P_2, S_2)$ duas gramáticas livres de contexto quaisquer, com $N_1 \cap N_2 = \emptyset$.

A gramática $G = (T, N, P, S)$ onde

$$T = T_1 \cup T_2$$
$$N = N_1 \cup N_2 \cup \{S\} \quad \text{com} \quad S \notin (N_1 \cup N_2)$$
$$P = \{S \to S_1S_2\} \cup P_1 \cup P_2$$

é livre de contexto e gera a linguagem $L = L(G_1) \cdot L(G_2)$

- A nova produção $S \to S_1S_2$ justapõe palavras de $L(G_2)$ às de $L(G_1)$
- Esta definição é **diferente** da que foi dada para a operação de concatenação nas gramáticas regulares

Fecho de Kleene de GLC
Exemplo

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, determine uma gramática livre de contexto que represente a linguagem

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

$\mathcal{R}$

$X = \{\, \omega \in T^* \,:\, \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) \}$	$X \to \varepsilon \mid \mathtt{a}\ B \mid \mathtt{b}\ A \mid \mathtt{c}\ X$ $A \to \mathtt{a}\ X \mid \mathtt{b}\ A\ A \mid \mathtt{c}\ A$ $B \to \mathtt{a}\ B\ B \mid \mathtt{b}\ X \mid \mathtt{c}\ B$
$A = \{\, \omega \in T^* \,:\, \#(\mathtt{a}, \omega) = \#(\mathtt{b}, \omega) + 1 \}$	Basta usar o A anterior como símbolo inicial
$L = X \cup A^*$	$S \to \varepsilon \mid A\ S \mid X$ $X \to \varepsilon \mid \mathtt{a}\ B \mid \mathtt{b}\ A \mid \mathtt{c}\ X$ $A \to \mathtt{a}\ X \mid \mathtt{b}\ A\ A \mid \mathtt{c}\ A$ $B \to \mathtt{a}\ B\ B \mid \mathtt{b}\ X \mid \mathtt{c}\ B$

- O fecho de A inclui a palavra vazia mas não as outras palavras com $\#_a = \#_b$

Operações sobre gramáticas
Fecho de Kleene

Seja $G_1 = (T_1, N_1, P_1, S_1)$ uma gramática livre de contexto qualquer. A gramática $G = (T, N, P, S)$ onde

$$\begin{aligned} T &= T_1 \\ N &= N_1 \cup \{S\} \quad \text{com} \quad S \notin N_1 \\ P &= \{S \to \varepsilon, S \to S_1 S\} \cup P_1 \end{aligned}$$

é livre de contexto e gera a linguagem $L = (L(G_1))^*$

- A produção $S \to \varepsilon$, per si, garante que $L^0(G_1) \subseteq L(G)$
- As produções $S \to S_1 S$ e $S \to \varepsilon$ garantem que $L^i(G_1) \subseteq L(G)$, para qualquer $i > 0$
- Esta definição é **diferente** da que foi dada para a operação de fecho nas gramáticas regulares

Símbolos produtivos e improdutivos
Exemplo de ilustração

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, considere a gramática

$$
\begin{aligned}
S &\rightarrow \mathtt{a}\ A\ \mathtt{b} \mid \mathtt{b}\ B \\
A &\rightarrow \mathtt{c}\ C \mid \mathtt{b}\ B \mid \mathtt{d} \\
B &\rightarrow \mathtt{d}\ D \mid \mathtt{b} \\
C &\rightarrow A\ C \mid B\ D \mid S\ D \\
D &\rightarrow A\ D \mid B\ C \mid C\ S \\
E &\rightarrow \mathtt{a}\ A \mid \mathtt{b}\ B \mid \varepsilon
\end{aligned}
$$

- Tente expandir (através de uma derivação) o símbolo não terminal A para uma sequência apenas com símbolos terminais ($S \Rightarrow^* u$, com $u \in T^*$)
  - $A \Rightarrow \mathtt{d}$
- Faça o mesmo com o símbolo C
  - Não consegue
- A é um símbolo **produtivo**; C é um símbolo **improdutivo**

Símbolos produtivos e improdutivos
Definição de símbolo produtivo

- Seja $G = (T, N, P, S)$ uma gramática qualquer
- Um símbolo não terminal A diz-se **produtivo** se for possível expandi-lo para uma expressão contendo apenas símbolos terminais
- Ou seja, A é produtivo se

$$A \Rightarrow^+ u \quad \wedge \quad u \in T^*$$

- Caso contrário, diz-se que A é **improdutivo**
- Uma gramática é improdutiva se o seu símbolo inicial for improdutivo

- Na gramática

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

  - S é produtivo, porque $S \Rightarrow \mathtt{ab} \quad \wedge \quad \mathtt{ab} \in T^*$
  - X é improdutivo, porque $X \Rightarrow \mathtt{c}X \Rightarrow \mathtt{cc}X \Rightarrow^* \mathtt{c}\cdots\mathtt{c}X$

Símbolos produtivos
Algoritmo de cálculo

- O conjunto dos símbolos produtivos, N_p, pode ser obtido por aplicação sucessiva das seguintes regras construtivas

  if $(A \to \alpha) \in P$ **and** $\alpha \in T^*$ **then** $A \in N_p$
  if $(A \to \alpha) \in P$ **and** $\alpha \in (T \cup N_p)^*$ **then** $A \in N_P$

- Algoritmo de cálculo:

```
let N_p ← ∅,  P_p ← P          # N_p – símbolos produtivos
repeat
    nothingAdded ← true
    foreach (A → α) ∈ P_p do
        if α ∈ (T ∪ N_p)* then          # se todos são terminais ou produtivos, A é produtivo
            if A ∉ N_p then             # se ainda não pertence aos produtivos
                N_p ← N_p ∪ {A}         # é lá colocado
                nothingAdded ← false    # e é preciso repetir o processo
            P_p ← P_p − {A → α}         # a produção já não precisa de ser processada mais
until nothingAdded or N_p = N
```


Símbolos acessíveis e inacessíveis
Exemplo de ilustração

- Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, considere a gramática

$$
\begin{aligned}
S &\to \mathtt{a}\ A\ \mathtt{b} \mid \mathtt{b}\ B \\
A &\to \mathtt{c}\ C \mid \mathtt{b}\ B \mid \mathtt{d} \\
B &\to \mathtt{d}\ D \mid \mathtt{b} \\
C &\to A\ C \mid B\ D \mid S\ D \\
D &\to A\ D \mid B\ C \mid C\ S \\
E &\to \mathtt{a}\ A \mid \mathtt{b}\ B \mid \varepsilon
\end{aligned}
$$

- Tente alcançar (através de uma derivação) o símbolo não terminal C a partir do símbolo inicial (S) ($S \Rightarrow^* \alpha\, C\, \beta$, com $\alpha, \beta \in (T \cup N)^*$)
  - $S \Rightarrow \mathtt{b}\ B \Rightarrow \mathtt{b}\ \mathtt{d}\ D \Rightarrow \mathtt{b}\ \mathtt{d}\ B\ C$
- Faça o mesmo com o símbolo E
  - Não consegue
- C é um símbolo **acessível**; E é um símbolo **inacessível**

Símbolos acessíveis e inacessíveis
Definição de símbolo acessível

- Seja $G = (T, N, P, S)$ uma gramática qualquer
- Um símbolo terminal ou não terminal x diz-se **acessível** se for possível expandir S (o símbolo inicial) para uma expressão que contenha x
- Ou seja, x é acessível se

$$S \Rightarrow^* \alpha\, x\, \beta$$

- Caso contrário, diz-se que x é **inacessível**

- Na gramática

$$\begin{aligned} S &\to \varepsilon \mid \mathtt{a}\ S\ \mathtt{b} \mid \mathtt{c}\ C\ \mathtt{c} \\ C &\to \mathtt{c}\ S\ \mathtt{c} \\ D &\to \mathtt{d}\ X\ \mathtt{d} \\ X &\to C\ C \end{aligned}$$

  - D, $\mathtt{d}$, e X são inacessíveis
  - Os restantes são acessíveis

Símbolos acessíveis
Algoritmo de cálculo

- O conjunto dos seus símbolos acessíveis, V_A, pode ser obtido por aplicação das seguintes regras construtivas

$S \in V_A$
if $A \to \alpha B \beta \in P$ **and** $A \in V_A$ **then** $B \in V_A$

- Algoritmo de cálculo:

$V_A \leftarrow \{S\}$ # *no fim, ficará com todos os símbolos acessíveis*
$N_A \leftarrow \{S\}$ # *conjunto de símbolos não terminais acessíveis a processar*
repeat
  $X \leftarrow$ `elementOf`(N_A) # *retira um elemento qualquer de* N_A
  foreach $(X \to \alpha) \in P$ **do**
    foreach x **in** α **do**
      if $x \notin V_A$ **then** # *se ainda não está marcado como acessível*
        $V_A \leftarrow V_A \cup \{x\}$ # *passa a estar*
        if $x \in N$ **then** # *se adicinalmente é não terminal*
          $N_A \leftarrow N_A \cup \{x\}$ # *terá de ser processado*
until $N_A = \emptyset$

Gramáticas limpas
Algoritmo de limpeza

- Numa gramática, os símbolos inacessíveis e os símbolos improdutivos são **símbolos inúteis**
- Se tais símbolos forem removidos obtém-se uma gramática equivalente
- Diz-se que uma gramática é **limpa** se não possuir símbolos inúteis
- Para limpar uma gramática deve-se:
  - começar por a expurgar dos símbolos improdutivos
  - só depois remover os inacessíveis

Gramáticas limpas
Exemplo #1

$\mathcal{Q}$ Sobre o conjunto de terminais $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, determine uma gramática limpa equivalente à gramática seguinte

$$
\begin{aligned}
S &\to \mathtt{a}\ A\ \mathtt{b} \mid \mathtt{b}\ B \\
A &\to \mathtt{c}\ C \mid \mathtt{b}\ B \mid \mathtt{d} \\
B &\to \mathtt{d}\ D \mid \mathtt{b} \\
C &\to A\ C \mid B\ D \mid S\ D \\
D &\to A\ D \mid B\ C \mid C\ S \\
E &\to \mathtt{a}\ A \mid \mathtt{b}\ B \mid \varepsilon
\end{aligned}
$$

- Cálculo dos símbolos produtivos
  1. Inicialmente $N_p \leftarrow \emptyset$
  2. $A \to \mathtt{d} \ \wedge\ \mathtt{d} \in T^* \implies N_p \leftarrow N_p \cup \{A\}$
  3. $B \to \mathtt{b} \ \wedge\ \mathtt{b} \in T^* \implies N_p \leftarrow N_p \cup \{B\}$
  4. $E \to \varepsilon \ \wedge\ \varepsilon \in T^* \implies N_p \leftarrow N_p \cup \{E\}$
  5. $S \to \mathtt{a}\,A\,\mathtt{b} \ \wedge\ \mathtt{a}, A, \mathtt{b} \in (T \cup N_p)^* \implies N_p \leftarrow N_p \cup \{S\}$
  6. Nada mais se consegue acrescentar a $N_p \implies C$ e D são improdutivos

Gramáticas limpas
Exemplo #1, cont.

- Gramática após a remoção dos símbolos improdutivos

$$S \rightarrow \texttt{a}\ A\ \texttt{b} \mid \texttt{b}\ B$$
$$A \rightarrow \texttt{b}\ B \mid \texttt{d}$$
$$B \rightarrow \texttt{b}$$
$$E \rightarrow \texttt{a}\ A \mid \texttt{b}\ B \mid \varepsilon$$

- Cálculo dos símbolos não terminais acessíveis sobre a nova gramática
  1. S é acessível, porque é o inicial
  2. sendo S acessível, de $S \rightarrow \texttt{a}\ A\ \texttt{b}$, tem-se que A é acessível
  3. sendo S acessível, de $S \rightarrow \texttt{b}\ B$, tem-se que B é acessível
  4. de A só se chega a B, que já foi marcado como acessível
  5. de B não se chega a nenhum não terminal
  6. Logo E não é acessível, pelo que a gramática limpa é

$$S \rightarrow \texttt{a}\ A\ \texttt{b} \mid \texttt{b}\ B$$
$$A \rightarrow \texttt{b}\ B \mid \texttt{d}$$
$$B \rightarrow \texttt{b}$$

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

Análise sintática
Introdução

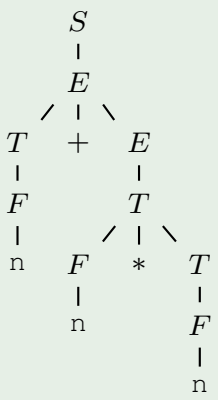
- Dada uma gramática $G = (T, N, P, S)$ e uma palavra $u \in T^*$, o papel da análise sintática é:
  - descobrir uma derivação que a partir de S produza u
  - gerar uma árvore de derivação (*parse tree*) que transforme S (a raiz) em u (as folhas)
- Se nenhuma derivação/árvore existir, então $u \notin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

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

- 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 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 $\{\mathtt{a}, \mathtt{b}\}$, 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 $S \rightarrow$ `a` S `b` se o `lookahead` for `a`
  - e $S \rightarrow \varepsilon$ 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 $\{\mathtt{a}, \mathtt{b}\}$, considere linguagem

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

  descrita pela gramática

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

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

- 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 $S \rightarrow \varepsilon$
- 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 $S \rightarrow \mathtt{a} \ B$ e outras $S \rightarrow \varepsilon$
- 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

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

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

- Tal como no caso anterior, escolher $S \rightarrow \varepsilon$ 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

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

- 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 $\mathtt{a}$ à esqueda, transformando a gramática para
    $$\begin{aligned} S &\rightarrow \mathtt{a}\ X \\ X &\rightarrow S\ \mathtt{b} \mid \mathtt{b} \end{aligned}$$
  - Aumentar o número de símbolos de *lookahead* para 2
    - se for $\mathtt{aa}$, escolhe-se $S \rightarrow \mathtt{a}\ S\ \mathtt{b}$
    - se for $\mathtt{ab}$, escolhe-se $S \rightarrow \mathtt{a}\ \mathtt{b}$

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

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

- 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

$\mathcal{Q}$ 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
  $$L = \{\texttt{a}^n\texttt{b}^n \ : \ n \geq 1\}$$
  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 a construção de um programa 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, representa genericamente a recursividade direta simples à esquerda

$$\begin{array}{rcl} A & \rightarrow & A\ \alpha \\ & | & \beta \end{array}$$

- Aplicando a primeira produção n vezes e a seguir a segunda, obtem-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

$$\begin{array}{rcl} A & \rightarrow & \beta\ X \\ X & \rightarrow & \varepsilon \\ & | & \alpha\ X \end{array}$$

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

Eliminação de recursividade à esquerda
Exemplo com recursividade direta simples

- Para a gramática

$$\begin{array}{rcl} S & \rightarrow & S\ \mathtt{a}\ \mathtt{b} \\ & | & \mathtt{c}\ \mathtt{b}\ \mathtt{a} \end{array}$$

obtenha-se uma gramática equivalente sem recursividade à esquerda

- Aplicando a estratégia anterior, tem-se

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

- Ou seja

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

- Que corresponde à gramática

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

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, representa genericamente a recursividade direta múltipla à esquerda

$$A \rightarrow A\ \alpha_1 \mid A\ \alpha_2 \mid \cdots \mid A\ \alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$

- 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

$$A \rightarrow \beta_1\ X \mid \beta_2\ X \mid \cdots \mid \beta_m\ X$$
$$X \rightarrow \varepsilon \mid \alpha_1\ X \mid \alpha_2\ X \mid \cdots \mid \alpha_n\ X$$

- 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

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

- As palavras da linguagem são da forma

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

- Obtendo-se a gramática

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

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

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

- O resultado seria

$$S \rightarrow A\ \mathtt{a} \mid \mathtt{b}$$
$$A \rightarrow S\ \mathtt{d}\ X \mid X$$
$$X \rightarrow \varepsilon \mid \mathtt{c}\ X$$

- A recursividade não foi eliminada

$$S \Rightarrow A\ \mathtt{a} \Rightarrow S\ \mathtt{d}\ X\ \mathtt{a} \Rightarrow A\ \mathtt{a}\ \mathtt{d}\ X\ \mathtt{a}$$

  - 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 $i, j = 1, 2, \cdots, n$

$$A_1 \rightarrow \alpha_1 \mid \beta_1 \mid \cdots \mid \Omega_1$$
$$A_2 \rightarrow \alpha_2 \mid \beta_2 \mid \cdots \mid \Omega_2$$
$$\cdots$$
$$A_n \rightarrow \alpha_n \mid \beta_n \mid \cdots \mid \Omega_n$$

- 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

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

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

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

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

$$S \rightarrow A\ \mathtt{a} \mid \mathtt{b}$$
$$A \rightarrow \mathtt{b}\ \mathtt{d}\ X \mid X$$
$$X \rightarrow \varepsilon \mid \mathtt{c}\ X \mid \mathtt{a}\ \mathtt{d}\ X$$

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:

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

- 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:

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

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

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

Conjunto **first**
Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

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

- 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

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

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

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

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

$$\textbf{first}\,(B\,C) = \textbf{first}\,(C) \cup \textbf{first}\,(\mathtt{b}\,S\,C)$$

  - Porque C começa pelo símbolo não terminal C

$$\textbf{first}\,(C) = \textbf{first}\,(\mathtt{c}) \cup \textbf{first}\,(c\,S)$$

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

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

Conjunto **first**
Exemplo #2

- Considere a gramatica

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

- Determine o conjunto $\mathbf{first}\,(B\,C)$
  - Porque $B\,C$ começa pelo símbolo não terminal B

    $\mathbf{first}\,(B\,C) = \mathbf{first}\,(C) \cup \mathbf{first}\,(\mathtt{b}\,S\,C)$
  - Porque C começa pelo símbolo não terminal C

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

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

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

- Note que a gramática não é a mesma
- Note que $\varepsilon \in \mathbf{first}\,(B\,C)$ 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. $\$ \in \mathbf{follow}\,(S)$
  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 gramatica

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

- Determine o conjunto $\mathbf{follow}\,(B)$
  - Procuram-se ocurrências de B no lado direito das produções. Há uma: $B\,C$
  - A produção $S \rightarrow B\,C$ encaixa nas regras 3 ou 4, dependendo de o ε pertencer ou não ao $\mathbf{first}\,(C)$
  - $\mathbf{first}\,(C) = \{\mathtt{c}\}$
  - $\therefore\ \mathbf{follow}\,(B) \supseteq \mathbf{first}\,(C)$ [*regra 3*]
  - Não havendo mais contribuições, tem-se

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

Conjunto **follow**
Exemplo #2

- Considere a gramatica

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

- Determine o conjunto $\mathbf{follow}\,(B)$
  - A produção $S \rightarrow B\,C$ encaixa nas regras 3 ou 4, dependendo de o ε pertencer ou não ao $\mathbf{first}\,(C)$
  - $\mathbf{first}\,(C) = \{\varepsilon, \mathtt{c}\}$
  - $\therefore\ \mathbf{follow}\,(B) \supseteq ((\mathbf{first}\,(C) - \{\varepsilon\}) \cup \mathbf{follow}\,(S))$ [*regra 4*]
  - Porque S é o símbolo inicial, $\$ \in \mathbf{follow}\,(S)$ [*regra 1*]
  - A produção $S \rightarrow \mathtt{a}\ S$ é irrelevante, porque diz que $\mathbf{follow}\,(S) \supseteq \mathbf{follow}\,(S)$
  - A produção $B \rightarrow \mathtt{b}\ S$ diz que $\mathbf{follow}\,(S) \supseteq \mathbf{follow}\,(B)$
  - A produção $C \rightarrow \mathtt{c}\ S$ diz que $\mathbf{follow}\,(S) \supseteq \mathbf{follow}\,(C)$
  - A produção $S \rightarrow B\,C$ diz que $\mathbf{follow}\,(C) \supseteq \mathbf{follow}\,(S)$
  - Pelas contribuições tem-se que

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

Conjunto **follow**
Exemplo #3

- Considere a gramatica

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

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

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

- 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 $t \in$ **predict** $(A \rightarrow \alpha)$
    add $(A \rightarrow \alpha)$ to $\tau(A, t)$

Tabela de decisão
Exemplo #1

- Considere a gramatica

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

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

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

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

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

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

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

  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

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

  $A \rightarrow \texttt{a} \mid \texttt{b}\ A\ A$

  $B \rightarrow \texttt{a}\ B\ B \mid \texttt{b}$

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

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

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

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

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

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

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

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

  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

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

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 $(S, \mathtt{a})$ 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

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

Tabela de decisão

	a	b	$
S	a A b S , ε	b S a S , ε	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 $\{\texttt{i}, \texttt{f}, \texttt{v}, \texttt{,}, \texttt{;}\}$, a linguagem L_4 descrita pela gramática

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

- 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.)

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

$$
\begin{array}{l}
\textbf{predict}\,(D \rightarrow T\ L\ \texttt{;}) = ? \\
\quad \textbf{first}\,(T\ L\ \texttt{;}) = ? \\
\quad \textbf{first}\,(T) = \textbf{first}\,(\texttt{i}) \cup \textbf{first}\,(\texttt{f}) = \{\texttt{i}\} \cup \{\texttt{f}\} \\
\quad \therefore\ \textbf{first}\,(T\ L\ \texttt{;}) = \{\texttt{i}, \texttt{f}\} \\
\quad \therefore\ \textbf{predict}\,(D \rightarrow T\ L\ \texttt{;}) == \{\texttt{i}, \texttt{f}\} \\
\textbf{predict}\,(T \rightarrow \texttt{i}) = ? \\
\quad \textbf{first}\,(\texttt{i}) = \{\texttt{i}\} \\
\quad \therefore\ \textbf{predict}\,(T \rightarrow \texttt{i}) = \{\texttt{i}\} \\
\textbf{predict}\,(T \rightarrow \texttt{f}) = \{\texttt{f}\} \\
\textbf{predict}\,(L \rightarrow \texttt{v}\ X) = ? \\
\quad \textbf{first}\,(\texttt{v}\ X) = \{\texttt{v}\} \\
\quad \therefore\ \textbf{predict}\,(L \rightarrow \texttt{v}\ X) = \{\texttt{v}\} \\
\textbf{predict}\,(X \rightarrow \varepsilon) = ? \\
\quad \textbf{first}\,(\varepsilon) = \{\varepsilon\} \\
\quad \therefore\ \textbf{predict}\,(X \rightarrow \varepsilon) = \textbf{follow}\,(X) \\
\quad \textbf{follow}\,(X) = \textbf{follow}\,(L) = \{\texttt{;}\} \\
\quad \therefore\ \textbf{predict}\,(X \rightarrow \varepsilon) = \{\texttt{;}\} \\
\textbf{predict}\,(X \rightarrow \texttt{,}\ L) = \{\texttt{,}\}
\end{array}
$$

Tabela de decisão
Exemplo #3 (cont.)

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

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

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

Compiladores

Análise sintática ascendente

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

DETI, Universidade de Aveiro

Ano letivo de 2022-2023

Sumário

Análise sintática ascendente
Ilustração por um exemplo

- Considere a gramática

$$\begin{array}{l} D \rightarrow T\ L\ \texttt{;} \\ T \rightarrow \texttt{i} \mid \texttt{r} \\ L \rightarrow \texttt{v} \mid L\ \texttt{,}\ \texttt{v} \end{array}$$

  que representa uma declaração de variáveis *a la* C
- Como reconhecer a palavra "$u = \texttt{i v , v ;}$" como pertencente à linguagem definida pela gramática dada?
- Se u pertence à linguagem definida pela gramática, então $D \Rightarrow^{+} u$
- Gerando uma derivação à direita, tem-se
  $D \Rightarrow T\,L\,\texttt{;} \Rightarrow T\,L\,\texttt{,}\,\texttt{v;} \Rightarrow T\,\texttt{v}\,\texttt{,}\,\texttt{v;} \Rightarrow \texttt{i v , v ;}$
- Tente-se agora fazer a derivação no sentido contrário, isto é, indo de u para D

Análise sintática ascendente
Ilustração por um exemplo (cont.)

- Considere a gramática

$$\begin{array}{l} D \rightarrow T\ L\ \texttt{;} \\ T \rightarrow \texttt{i} \mid \texttt{r} \\ L \rightarrow \texttt{v} \mid L\ \texttt{,}\ \texttt{v} \end{array}$$

  e *reduza-se* a palavra "$u = \texttt{i v , v ;}$" ao símbolo inicial D
- $$\begin{array}{lll} \texttt{i v , v ;} & & \\ \quad \Leftarrow T\ \texttt{v , v ;} & & (\text{por aplicação da produção } T \rightarrow \texttt{i}) \\ \quad \Leftarrow T\ L\ \texttt{, v ;} & & (\text{por aplicação da produção } L \rightarrow \texttt{v}) \\ \quad \Leftarrow T\ L\ \texttt{;} & & (\text{por aplicação da produção } L \rightarrow L\ \texttt{,}\ \texttt{v}) \\ \quad \Leftarrow D & & (\text{por aplicação da produção } D \rightarrow T\ L\ \texttt{;}) \end{array}$$
- Colocando ao contrário, tem-se

  $D \Rightarrow T\,L\,\texttt{;} \Rightarrow T\,L\,\texttt{,}\,\texttt{v;} \Rightarrow T\,\texttt{v}\,\texttt{,}\,\texttt{v;} \Rightarrow \texttt{i v , v ;}$

  que corresponde à derivação à direita da palavra "$u = \texttt{i v , v ;}$"

Análise sintática ascendente
Ilustração por um exemplo (cont.)

$$D \to T\ L\ \texttt{;}$$
$$T \to \texttt{i} \mid \texttt{r}$$
$$L \to \texttt{v} \mid L\ \texttt{,}\ \texttt{v}$$

- A tabela seguinte mostra como, na prática, se realiza esta (retro)derivação

pilha	entrada	próxima ação
	i v , v ; $	deslocamento
i	v , v ; $	redução por $T \to \texttt{i}$
T	v , v ; $	deslocamento
T v	, v ; $	redução por $L \to \texttt{v}$
$T\ L$	, v ; $	deslocamento
$T\ L$,	v ; $	deslocamento
$T\ L$, v	; $	redução por $L \to L\ \texttt{,}\ \texttt{v}$
$T\ L$	; $	deslocamento
$T\ L$;	$	redução por $D \to T\ L\ \texttt{;}$
D	$	deslocamento / aceitação
D $		aceitação

- A palavra à entrada foi reduzida ao símbolo inicial pelo que é aceite como pertencendo à linguagem

- A aceitação pode ser feita antes de consumir o $ ou depois

Análise sintática ascendente
Ilustração de um erro sintático

$$D \to T\ L\ \texttt{;}$$
$$T \to \texttt{i} \mid \texttt{r}$$
$$L \to \texttt{v} \mid L\ \texttt{,}\ \texttt{v}$$

- Veja-se a reação deste procedimento a uma entrada errada, por exemplo a palavra `i v v ;`.

pilha	entrada	próxima ação
	i v v ; $	deslocamento
i	v v ; $	redução por $T \to \texttt{i}$
T	v v ; $	deslocamento
T v	v ; $	redução por $L \to \texttt{v}$
$T\ L$	v ; $	deslocamento
$T\ L$ v	; $	rejeição

- Rejeita porque L v não corresponde ao prefixo de uma produção da gramática
- Na realidade, o erro poderia ter sido detetado dois passos antes, aquando da segunda redução, porque $\texttt{v} \notin \texttt{follow}(L)$
  - v corresponde ao símbolo à entrada
  - L é o símbolo que iria aparecer no topo da pilha se se fizesse a redução por $L \to \texttt{v}$

Análise sintática ascendente
Ilustração de conflito entre deslocamento e redução

- Considere a gramática

$$\begin{aligned} S &\to \texttt{i}\ \texttt{c}\ S \\ &\mid \texttt{i}\ \texttt{c}\ S\ \texttt{e}\ S \\ &\mid \texttt{a} \end{aligned}$$

e aplique-se o procedimento anterior à palavra `icicaea`

pilha	entrada	próxima ação
	icicaea$	deslocamento
i	cicaea$	deslocamento
ic	icaea$	deslocamento
ici	caea$	deslocamento
icic	aea$	deslocamento
icica	ea$	redução por $S \to \texttt{a}$
icic$S$	ea$	**conflito**:
		– redução por $S \to \texttt{i}\ \texttt{c}\ S$
		– deslocamento para tentar $S \to \texttt{i}\ \texttt{c}\ S\ \texttt{e}\ S$

- Esta gramática representa uma estrutura típica em linguagens de programação. Qual?

Análise sintática ascendente
Ilustração de conflito entre reduções

- Considere a gramática

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

e aplique-se o procedimento anterior à palavra `c`

pilha	entrada	próxima ação
	c$	deslocamento
c	$	**conflito:**
		– redução usando $A \to \texttt{c}$
		– redução usando $B \to \texttt{c}$

Análise sintática ascendente
Ilustração de falso conflito

- Considere a gramática

$$\begin{aligned} S &\rightarrow \texttt{a} \\ &\mid \texttt{<}\ S\ \texttt{>} \\ &\mid \texttt{a}\ P \\ &\mid \texttt{<}\ S\ \texttt{>}\ S \\ P &\rightarrow \texttt{<}\ S\ \texttt{>} \\ &\mid \texttt{<}\ S\ \texttt{>}\ S \end{aligned}$$

  e aplique-se o procedimento de reconhecimento à palavra "a < a > a"

pilha	entrada	próxima ação
	a < a > a $	deslocamento
a	< a > a $	falso conflito:
		– redução usando $S \rightarrow \texttt{a}$
		– deslocamento para tentar $S \rightarrow \texttt{a}\ P$

- Deslocamento, porque se se optasse pela redução no topo da pilha ficaria um S e $\texttt{<} \notin \texttt{follow}(S)$

Análise sintática ascendente
Ilustração de falso conflito (cont.)

- Optando pelo deslocamento e continuando...

pilha	entrada	próxima ação
	a < a > a $	deslocamento
a	< a > a $	deslocamento, porque $\texttt{<} \notin \texttt{follow}(S)$
a <	a > a $	deslocamento
a < a	> a $	redução por $S \rightarrow \texttt{a}$
a < S	> a $	deslocamento
a < S >	a $	deslocamento, porque $\texttt{a} \notin \texttt{follow}(P)$
a < S > a	$	redução por $S \rightarrow \texttt{a}$
a < S > S	$	redução por $P \rightarrow \texttt{<}\ S\ \texttt{>}\ S$
a P	$	redução por $S \rightarrow \texttt{a}\ P$
S	$	deslocamento
S $		aceitação

Análise sintática ascendente
Eliminação de conflito

- Pode ser possível alterar uma gramática de modo a eliminar a fonte de conflito
- Considerando que se pretendia optar pelo deslocamento, a gramática da esquerda gera a mesma linguagem que a da direita e está isenta de conflitos.

$$
\begin{array}{rcl}
S & \to & \texttt{a} \\
 & \mid & \texttt{i c } S \\
 & \mid & \texttt{i c } S' \texttt{ e } S \\
S' & \to & \texttt{a} \\
 & \mid & \texttt{i c } S' \texttt{ e } S'
\end{array}
\qquad
\begin{array}{rcl}
S & \to & \texttt{a} \\
 & \mid & \texttt{i c } S \\
 & \mid & \texttt{i c } S \texttt{ e } S
\end{array}
$$

Análise sintática ascendente
`if..then..else` sem conflitos

- Considere a gramática seguinte e processe-se a palavra "`i c i c a e a`"

$$
\begin{array}{rcl}
S & \to & \texttt{a} \mid \texttt{i c } S \mid \texttt{i c } S' \texttt{ e } S \\
S' & \to & \texttt{a} \mid \texttt{i c } S' \texttt{ e } S'
\end{array}
$$

pilha	entrada	próxima ação
	i c i c a e a $	deslocamento
i	c i c a e a $	deslocamento
i c	i c a e a $	deslocamento
i c i	c a e a $	deslocamento
i c i c	a e a $	deslocamento
i c i c a	e a $	redução por $S' \to \texttt{a}$ //e $\in$ **follow** (S'), e $\notin$ **follow** (S)
i c i c S'	e a $	deslocamento
i c i c S' e	a $	deslocamento
i c i c S' e a	$	redução por $S \to \texttt{a}$ //$ $\in$ **follow** (S), $ $\notin$ **follow** (S')
i c i c S' e S	$	redução por $S \to \texttt{i c } S' \texttt{ e } S$
i c S	$	redução por $S \to \texttt{i c } S$
S	$	deslocamento e aceitação

Construção de um reconhecedor ascendente
Abordagem

- Como determinar de forma sistemática a ação a realizar (deslocamento, redução, aceitação, rejeição)?

pilha	entrada	próxima ação
	i v v ; $	deslocamento
i	v v ; $	redução por $T \rightarrow$ i
T	v v ; $	deslocamento
T v	v ; $	rejeição

- A ação a realizar em cada passo do procedimento de reconhecimento – deslocamento, redução, aceitação ou rejeição – depende da configuração em cada momento
- Uma **configuração** é formada pelo conteúdo da pilha mais a parte da entrada ainda não processada
- A pilha é conhecida – na realidade, é preenchida pelo procedimento de reconhecimento
- Da entrada, em cada momento, apenas se conhece o *lookahead*

Construção de um reconhecedor ascendente
Abordagem (cont.)

pilha	entrada	próxima ação
	i v v ; $	deslocamento
i	v v ; $	redução por $T \rightarrow$ i
T	v v ; $	deslocamento
T v	v ; $	rejeição

- Quantos símbolos da pilha usar?
- Poder-se-á usar apenas um?
- Se se quiser e puder construir um reconhecedor que apenas use o símbolo no topo, uma pilha onde se guardam os símbolos terminais e não terminais tem pouco interesse
- Mas pode definir-se um alfabeto adequado para a pilha
- Os símbolos a colocar na pilha devem representar estados no processo de deslocamento/redução/aceitação
- Por exemplo, um dado símbolo pode significar que, na produção "$D \rightarrow T\ L$;", já se processou algo que corresponde ao "$T\ L$", faltando o "**;**"

Construção de um reconhecedor ascendente
Itens de uma gramática

- O alfabeto da pilha representa assim o conjunto de possíveis estados nesse processo de reconhecimento
- Cada estado representa um conjunto de itens
- Cada item representa o quanto de uma produção já foi processado e o quanto ainda falta processar
  - Usa-se um ponto ($\cdot$) ao longo dos símbolos de uma produção para o representar
- A produção $A \to B_1 \ B_2 \ B_3$ produz 4 itens:

  $A \to \cdot \, B_1 \ B_2 \ B_3$
  $A \to B_1 \cdot B_2 \ B_3$
  $A \to B_1 \ B_2 \cdot B_3$
  $A \to B_1 \ B_2 \ B_3 \cdot$
- A produção $A \to \varepsilon$ produz um único item:

  $A \to \cdot$
- Um item com um ponto ($\cdot$) à direita representa uma **ação de redução**

Conjunto dos conjuntos de itens
Ilustração com um exemplo

- Considere a gramática

  $S \to E$
  $E \to \texttt{a} \mid \texttt{(} \ E \ \texttt{)}$
- Reconhecer a palavra $u = u_1 u_2 \cdots u_n$, significa reduzir $u\$$ a $S\$$, então, o estado inicial no processo de reconhecimento pode ser definido por

  $Z_0 = \{S \to \cdot \, E \, \$\}$
- O facto de o ponto ($\cdot$) se encontrar imediatamente à esquerda de um símbolo significa que para se avançar no processo de reconhecimento é preciso obter esse símbolo
  - Se o símbolo é terminal, isso corresponde a uma ação de deslocamento
  - Se o símbolo é não terminal, é preciso dar-se a redução de uma produção que o produza
  - Isso é considerado juntando ao conjunto os itens iniciais das produções cuja cabeça é o símbolo pretendido

    $Z_0 = \{ \, S \to \cdot \, E \, \$ \, \} \cup \{ \, E \to \cdot \, \texttt{a} \, , \, E \to \cdot \, \texttt{(} \ E \ \texttt{)} \, \}$
- Se aparecerem novos símbolos não terminais imediatamente à direita de um ponto ($\cdot$), repete-se o processo. Faz-se o **fecho** (**closure**)

Conjunto dos conjuntos de itens
Ilustração com um exemplo (cont.)

- Evolução de Z_0:

$$Z_0 = \{ S \to \cdot\, E\, \$ \} \cup \{ E \to \cdot\, \mathtt{a}\ ,\ E \to \cdot\, (\ E\ ) \}$$

- O estado Z_0 pode evoluir por ocorrência de um E, um `a` ou um `(`, que correspondem aos símbolos que aparecem imediatamente à direita do ponto ($\cdot$)

$$\delta(Z_0, E) = \{ S \to E \cdot \$ \} = Z_1 \qquad \textit{um estado novo}$$

$$\delta(Z_0, \mathtt{a}) = \{ E \to \mathtt{a} \cdot \} = Z_2 \qquad \textit{um estado novo}$$

$$\delta(Z_0, \mathtt{(}) = \{ E \to \mathtt{(} \cdot E\ \mathtt{)} \} = Z_3 \qquad \textit{um estado novo}$$

- Z_3 tem de ser estendido pela função de fecho, uma vez que o ponto ($\cdot$) ficou imediatamente à esquerda de um símbolo não terminal (E)

$$Z_3 = \delta(Z_0, \mathtt{(}) = \{ E \to \mathtt{(} \cdot E\ \mathtt{)} \} \cup \{ E \to \cdot\, \mathtt{a}\ ,\ E \to \cdot\, \mathtt{(}\ E\ \mathtt{)} \}$$

- Z_2, apenas possui um item terminal (com o ponto ($\cdot$) à direita), que representa uma situação passível de redução, neste caso pela produção $E \to \mathtt{a}$

Conjunto dos conjuntos de itens
Ilustração com um exemplo (cont.)

- Evolução de Z_1:

$$Z_1 = \{ S \to E \cdot \$ \}$$

- Apenas evolui por ocorrência de um $\$$

$$\delta(Z_1, \$) = \{ S \to E\, \$ \cdot \} \qquad \Longrightarrow \text{ACCEPT}$$

que corresponde à situação de aceitação

- Se o símbolo inicial da gramática não aparecer no corpo de qualquer produção (como acontece aqui), Pode-se considerar Z_1 como uma situação de aceitação se o *lookahead* for $\$$

- Evolução de Z_3:

$$Z_3 = \{ E \to \mathtt{(} \cdot E\ \mathtt{)} \} \cup \{ E \to \cdot\, \mathtt{a}\ ,\ E \to \cdot\, \mathtt{(}\ E\ \mathtt{)} \}$$

- Pode evoluir por ocorrência de um E, um `a` ou um `(`

$$\delta(Z_3, E) = \{ E \to \mathtt{(}\ E \cdot\ \mathtt{)} \} = Z_4 \qquad \textit{um estado novo}$$

$$\delta(Z_3, \mathtt{a}) = \{ E \to \mathtt{a} \cdot \} = Z_2 \qquad \textit{um estado repetido}$$

$$\delta(Z_3, \mathtt{(}) = \{ E \to \mathtt{(} \cdot E\ \mathtt{)} \} = Z_3 \qquad \textit{um estado repetido}$$

- A evolução de Z_3 com `a` e `(` dá origem a elementos já obtidos anteriormente

Conjunto dos conjuntos de itens
Ilustração com um exemplo (cont.)

- Evolução de Z_4

$$Z_4 = \{ E \to (\; E \; \cdot \;) \; \}$$

- Apenas evolui por ocorrência de)

$$\delta(Z_4,)) = \{ E \to (\; E \;) \; \cdot \; \} = Z_5 \qquad \textit{um estado novo}$$

- Z_5 apenas possui um item terminal, que representa uma situação passível de redução pela regra $E \to (\; E \;)$

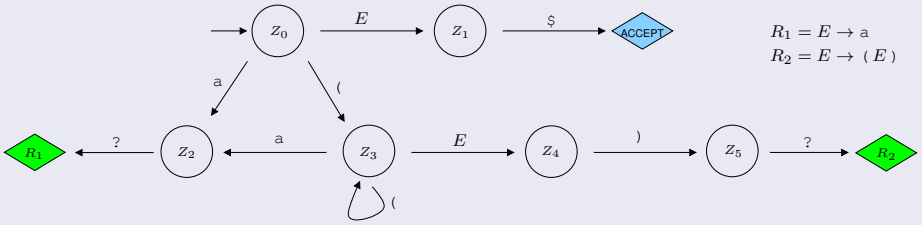
- Pode acontecer que um dado elemento (conjunto de itens) possua itens terminais (associados a reduções) e não terminais

Conjunto dos conjuntos de itens
Ilustração com um exemplo (cont.)

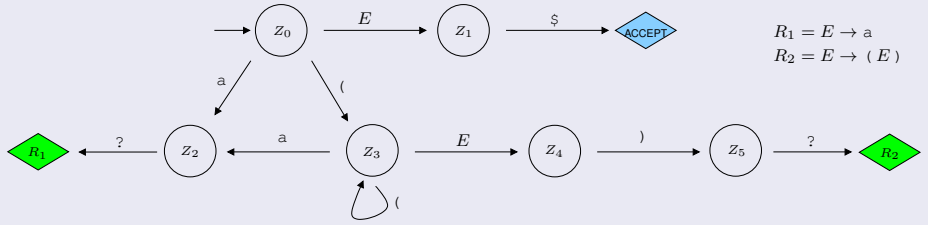
- Pondo tudo junto

$$\begin{aligned}
Z_0 &= \{ S \to \cdot \, E \, \$ \} \cup \{ E \to \cdot \, \mathtt{a} \, , \; E \to \cdot \, (\; E \;) \; \} \\
Z_1 &= \delta(Z_0, E) = \{ S \to E \cdot \$ \} \\
Z_2 &= \delta(Z_0, \mathtt{a}) = \{ E \to \mathtt{a} \cdot \} \\
Z_3 &= \delta(Z_0, () = \{ E \to (\cdot \, E \;) \; \} \cup \{ E \to \cdot \, \mathtt{a} \, , \; E \to \cdot \, (\; E \;) \; \} \\
Z_4 &= \delta(Z_3, E) = \{ E \to (\; E \cdot \;) \; \} \\
Z_5 &= \delta(Z_4,)) = \{ E \to (\; E \;) \cdot \}
\end{aligned}$$

- Representando na forma de um autómato, tem-se

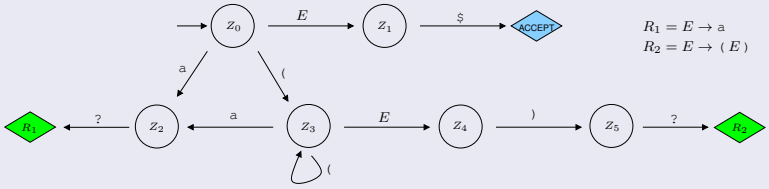


Conjunto dos conjuntos de itens
Ilustração com um exemplo (cont.)



- Neste autómato, os estados representam o alfabeto da pilha
- As transições representam operações de *push*
- As transições etiquetadas com símbolos terminais representam adicionalmente ações de deslocamento (*shift*)
- As ações de redução provocam operações de *pop*, em número igual ao número de elementos do corpo da produção
- As transições etiquetadas com símbolos não terminais ocorrem após as ações de redução
- Tudo isto representa o funcionamento de um autómato de pilha que permite fazer o reconhecimento da linguagem

Tabela de decisão de um reconhecedor ascendente
Introdução



- O autómato de pilha pode ser implementado usando uma tabela de decisão
- Esta tabela contém duas matrizes, ACTION e GOTO
  - as linhas de ambas são indexadas pelo alfabeto da pilha (conjunto de conjuntos de itens)
- A matriz ACTION representa ações
  - as colunas são indexadas pelos símbolos terminais da gramática, incluindo o marcador de fim de entrada ($)
  - As células contêm as ações *shift*, *reduce*, *accept* ou *error*
  - No caso de *shift*, também inclui o próximo símbolo a colocar na pilha
- A matriz GOTO representa a operação após uma redução
  - as colunas são indexadas pelos símbolos não terminais da gramática
  - As células indicam que valor colocar na *stack* após uma ação de redução

Tabela de decisão de um reconhecedor ascendente
Exemplo

- Considere-se o conjunto de conjunto de itens obtido anteriormente

$$
\begin{aligned}
Z_0 &= \{\, S \to \cdot\, E\,\$ \,\} \cup \{\, E \to \cdot\, \mathtt{a}\,,\; E \to \cdot\, (\; E\;) \,\} \\
Z_1 &= \delta(Z_0, E) = \{\, S \to E \cdot \$ \,\} \\
Z_2 &= \delta(Z_0, \mathtt{a}) = \{\, E \to \mathtt{a} \cdot \,\} \\
Z_3 &= \delta(Z_0, () = \{\, E \to (\; \cdot\, E\;) \,\} \cup \{\, E \to \cdot\, \mathtt{a}\,,\; E \to \cdot\, (\; E\;) \,\} \\
Z_4 &= \delta(Z_3, E) = \{\, E \to (\; E \cdot\;) \,\} \\
Z_5 &= \delta(Z_4,)) = \{\, E \to (\; E\;) \cdot \,\}
\end{aligned}
$$

- E o correspondente autómato de pilha

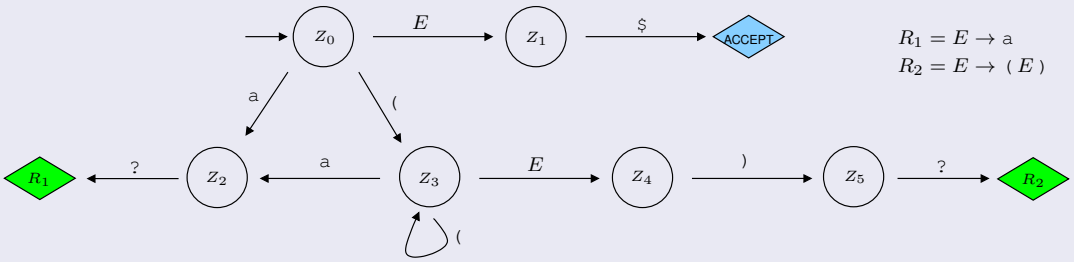
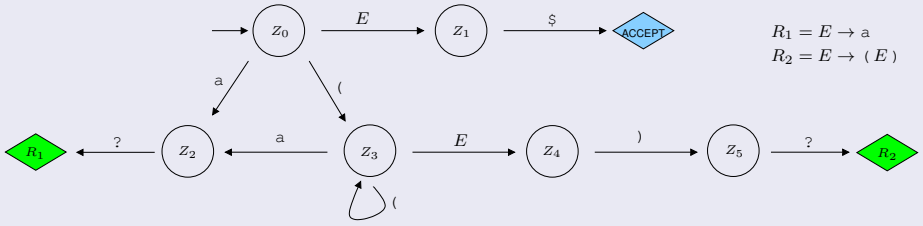


Tabela de decisão de um reconhecedor ascendente
Exemplo

- A este autómato de pilha



- Corresponde a tabela de decisão

	ACTION				GOTO
	a	(	)	$	E
Z_0	shift, Z_2	shift, Z_3			Z_1
Z_1				ACCEPT	
Z_2			reduce, $E \to \mathtt{a}$	reduce, $E \to \mathtt{a}$	
Z_3	shift, Z_2	shift, Z_3			Z_4
Z_4			shift, Z_5		
Z_5			reduce, $E \to (\; E\;)$	reduce, $E \to (\; E\;)$	

- Com *lookahead* de 1, as reduções apenas são colocadas nas colunas correspondentes aos **`follow`**.

Reconhecedor ascendente
Algoritmo de reconhecimento

	ACTION				GOTO
	a	(	)	$	E
Z_0	shift, Z_2	shift, Z_3			Z_1
Z_1				ACCEPT	
Z_2			reduce, $E \to$ a	reduce, $E \to$ a	
Z_3	shift, Z_2	shift, Z_3			Z_4
Z_4			shift, Z_5		
Z_5			reduce, $E \to$ (E)	reduce, $E \to$ (E)	

- Com base na tabela de decisão, o procedimento de reconhecimento pode ser implementado pelo seguinte algoritmo

```
push(Z0)
forever
    if top() == Z1 and lookahead == $
        ACCEPT
    action = ACTION[top,lookahead]
    if action is (shift, Zi)
        adv(); push(Zi);
    else if action is (reduce A → α)
        pop |α| símbolos; push(GOTO[top(),A]);
    else
        REJECT
```

- Note que após os *pops* o símbolo no *top* pode mudar e é o novo símbolo que é usado no `GOTO`

Reconhecedor ascendente
Ilustração com o exemplo anterior

	ACTION				GOTO
	a	(	)	$	E
Z_0	shift, Z_2	shift, Z_3			Z_1
Z_1				ACCEPT	
Z_2			reduce, $E \to$ a	reduce, $E \to$ a	
Z_3	shift, Z_2	shift, Z_3			Z_4
Z_4			shift, Z_5		
Z_5			reduce, $E \to$ (E)	reduce, $E \to$ (E)	

- Aplique-se este algoritmo à palavra `((a))`

pilha	entrada	próxima ação
Z_0	((a)) $	ACTION(Z_0, () = (shift, Z_3)
$Z_0\ Z_3$	(a)) $	ACTION(Z_3, () = (shift, Z_3)
$Z_0\ Z_3\ Z_3$	a)) $	ACTION(Z_3, a) = (shift, Z_2)
$Z_0\ Z_3\ Z_3\ Z_2$	)) $	ACTION(Z_2,)) = (reduce $E \to$ a) (1 *pop*)
$Z_0\ Z_3\ Z_3$	$[E]$	GOTO(Z_3,E) = Z_4 (*push* Z_4)
$Z_0\ Z_3\ Z_3\ Z_4$	)) $	ACTION(Z_4,)) = (shift, Z_5)
$Z_0\ Z_3\ Z_3\ Z_4\ Z_5$	) $	ACTION(Z_5,)) = (reduce $E \to$ (E)) (3 *pops*)
$Z_0\ Z_3$	$[E]$	GOTO(Z_3,E) = Z_4 (*push* Z_4)
$Z_0\ Z_3\ Z_4$	) $	ACTION(Z_4,)) = (shift, Z_5)
$Z_0\ Z_3\ Z_4\ Z_5$	$	ACTION(Z_5, $) = (reduce $E \to$ (E)) (3 *pops*)
Z_0	$[E]$	GOTO(Z_0,E) = Z_1 (*push* Z_1)
$Z_0\ Z_1$	$	ACTION(Z_1, $) = ACCEPT

Tabela de decisão de um reconhecedor ascendente
Exemplo #3

$\mathcal{Q}$ Determine-se a tabela de decisão para um reconhecedor ascendente com *lookahead* 1 da gramática seguinte

$$S \rightarrow \mathtt{a} \mid (\ S\ ) \mid \mathtt{a}\,P \mid (\ S\ )\ S$$
$$P \rightarrow (\ S\ ) \mid (\ S\ )\ S$$

- O primeiro passo corresponde a alterar a gramática de modo ao símbolo inicial não aparecer do lado direito

$$S_0 \rightarrow S$$
$$S \rightarrow \mathtt{a} \mid (\ S\ ) \mid \mathtt{a}\,P \mid (\ S\ )\ S$$
$$P \rightarrow (\ S\ ) \mid (\ S\ )\ S$$

Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- O passo seguinte corresponde a calcular o conjunto de conjunto de itens

$Z_0 = \{S_0 \rightarrow \cdot\, S\,\$\}$	
$\cup\ \{S \rightarrow \cdot\, \mathtt{a}, S \rightarrow \cdot\, (\ S\ ),\ S \rightarrow \cdot\, \mathtt{a}\,P, S \rightarrow \cdot\, (\ S\ )\ S\}$	*fecho*
$\delta(Z_0, S) = \{S_0 \rightarrow S \cdot \$\} = Z_1$	*um estado novo*
$\delta(Z_0, \mathtt{a}) = \{S \rightarrow \mathtt{a}\cdot, S \rightarrow \mathtt{a} \cdot P\}$	*um estado novo*
$\cup\ \{P \rightarrow \cdot\, (\ S\ ), P \rightarrow \cdot\, (\ S\ )\ S\} = Z_2$	*fecho*
$\delta(Z_0, (\,) = \{S \rightarrow (\ \cdot\ S\ ), S \rightarrow (\ \cdot\ S\ )\ S\}$	*um estado novo*
$\cup\ \{S \rightarrow \cdot\, \mathtt{a}, S \rightarrow \cdot\, (\ S\ ), S \rightarrow \cdot\, \mathtt{a}\,P, S \rightarrow \cdot\, (\ S\ )\ S\} = Z_3$	*fecho*
$\delta(Z_2, P) = \{S \rightarrow \mathtt{a}\,P \cdot\} = Z_4$	*um estado novo*
$\delta(Z_2, (\,) = \{P \rightarrow (\ \cdot\ S\ ), P \rightarrow (\ \cdot\ S\ )\ S\}$	*um estado novo*
$\cup\ \{S \rightarrow \cdot\, \mathtt{a}, S \rightarrow \cdot\, (\ S\ ), S \rightarrow \cdot\, \mathtt{a}\,P, S \rightarrow \cdot\, (\ S\ )\ S\} = Z_5$	*fecho*
$\delta(Z_3, S) = \{S \rightarrow (\ S \cdot\ ), S \rightarrow (\ S \cdot\ )\ S\} = Z_6$	*um estado novo*
$\delta(Z_3, \mathtt{a}) = \{S \rightarrow \mathtt{a}\cdot, S \rightarrow \mathtt{a} \cdot P\} = Z_2$	*um estado repetido*
$\delta(Z_3, (\,) = \{S \rightarrow (\ \cdot\ S\ ), S \rightarrow (\ \cdot\ S\ )\ S\} = Z_3$	*um estado repetido*

$$S_0 \rightarrow S$$
$$S \rightarrow \mathtt{a} \mid (\ S\ ) \mid \mathtt{a}\,P \mid (\ S\ )\ S$$
$$P \rightarrow (\ S\ ) \mid (\ S\ )\ S$$

Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- continuando, apenas mostrando os elementos envolvidos em processamento

Z_2	$= \{S \to \mathtt{a}\,\cdot, S \to \mathtt{a}\,\cdot\,P\} \cup \cdots$	
Z_3	$= \{S \to (\,\cdot\,S\,), S \to (\,\cdot\,S\,)\,S\} \cup \cdots$	
Z_5	$= \{P \to (\,\cdot\,S\,), P \to (\,\cdot\,S\,)\,S\}$ $\cup\ \{S \to \cdot\,\mathtt{a}, S \to \cdot\,(\,S\,), S \to \cdot\,\mathtt{a}\,P, S \to \cdot\,(\,S\,)\,S\}$	
Z_6	$= \{S \to (\,S\,\cdot\,), S \to (\,S\,\cdot\,)\,S\}$	
$\delta(Z_5, S)$	$= \{P \to (\,S\,\cdot\,), P \to (\,S\,\cdot\,)\,S\} = Z_7$	*um estado novo*
$\delta(Z_5, \mathtt{a})$	$= \{S \to \mathtt{a}\,\cdot, S \to \mathtt{a}\,\cdot\,P\} = Z_2$	*um estado repetido*
$\delta(Z_5, ($	$= \{S \to (\,\cdot\,S\,), S \to (\,\cdot\,S\,)\,S\} = Z_3$	*um estado repetido*
$\delta(Z_6,)$	$= \{S \to (\,S\,)\,\cdot, S \to (\,S\,)\,\cdot\,S\}$ $\cup\ \{S \to \cdot\,\mathtt{a}, S \to \cdot\,(\,S\,), S \to \cdot\,\mathtt{a}\,P, S \to \cdot\,(\,S\,)\,S\} = Z_8$	*um estado novo*
$\delta(Z_7,)$	$= \{P \to (\,S\,)\,\cdot, P \to (\,S\,)\,\cdot\,S\}$ $\cup\ \{S \to \cdot\,\mathtt{a}, S \to \cdot\,(\,S\,), S \to \cdot\,\mathtt{a}\,P, S \to \cdot\,(\,S\,)\,S\} = Z_9$	*um estado novo*

$S_0 \to S$
$S \to \mathtt{a} \mid (\,S\,) \mid \mathtt{a}\,P \mid (\,S\,)\,S$
$P \to (\,S\,) \mid (\,S\,)\,S$

Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- continuando...

Z_2	$= \{S \to \mathtt{a}\,\cdot, S \to \mathtt{a}\,\cdot\,P\} \cup \cdots$	
Z_3	$= \{S \to (\,\cdot\,S\,), S \to (\,\cdot\,S\,)\,S\} \cup \cdots$	
Z_8	$= \{S \to (\,S\,)\,\cdot, S \to (\,S\,)\,\cdot\,S\}$ $\cup\ \{S \to \cdot\,\mathtt{a}, S \to \cdot\,(\,S\,), S \to \cdot\,\mathtt{a}\,P, S \to \cdot\,(\,S\,)\,S\}$	
Z_9	$= \{P \to (\,S\,)\,\cdot, P \to (\,S\,)\,\cdot\,S\}$ $\cup\ \{S \to \cdot\,\mathtt{a}, S \to \cdot\,(\,S\,), S \to \cdot\,\mathtt{a}\,P, S \to \cdot\,(\,S\,)\,S\}$	
$\delta(Z_8, S)$	$= S \to (\,S\,)\,S\,\cdot\} = Z_{10}$	*um estado novo*
$\delta(Z_8, \mathtt{a})$	$= \{S \to \mathtt{a}\,\cdot, S \to \mathtt{a}\,\cdot\,P\} = Z_2$	*um estado repetido*
$\delta(Z_8, ($	$= \{S \to (\,\cdot\,S\,), S \to (\,\cdot\,S\,)\,S\} = Z_3$	*um estado repetido*
$\delta(Z_9, S)$	$= \{P \to (\,S\,)\,S\,\cdot\} = Z_{11}$	*um estado novo*
$\delta(Z_9, \mathtt{a})$	$= \{S \to \mathtt{a}\,\cdot, S \to \mathtt{a}\,\cdot\,P\} = Z_2$	*um estado repetido*
$\delta(Z_9, ($	$= \{S \to (\,\cdot\,S\,), S \to (\,\cdot\,S\,)\,S\} = Z_3$	*um estado repetido*

$S_0 \to S$
$S \to \mathtt{a} \mid (\,S\,) \mid \mathtt{a}\,P \mid (\,S\,)\,S$
$P \to (\,S\,) \mid (\,S\,)\,S$

Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- O que resulta em

$Z_0 = \{S_0 \to \cdot\, S\,\$\} \cup \cdots$	$\delta(Z_0, S) = Z_1$	$\delta(Z_0, \mathtt{a}) = Z_2$	$\delta(Z_0, \mathtt{(}) = Z_3$
$Z_1 = \{S_0 \to S \cdot \$\}$			
$Z_2 = \{S \to \mathtt{a}\cdot, S \to \mathtt{a} \cdot P\} \cup \cdots$		$\delta(Z_2, P) = Z_4$	$\delta(Z_2, \mathtt{(}) = Z_5$
$Z_3 = \{S \to \mathtt{(} \cdot S \mathtt{)}, S \to \mathtt{(} \cdot S \mathtt{)}\, S\} \cup \cdots$	$\delta(Z_3, S) = Z_6$	$\delta(Z_3, \mathtt{a}) = Z_2$	$\delta(Z_3, \mathtt{(}) = Z_3$
$Z_4 = \{S \to \mathtt{a}\, P \cdot\}$			
$Z_5 = \{P \to \mathtt{(} \cdot S \mathtt{)}, P \to \mathtt{(} \cdot S \mathtt{)}\, S\} \cup \cdots$	$\delta(Z_5, S) = Z_7$	$\delta(Z_5, \mathtt{a}) = Z_2$	$\delta(Z_5, \mathtt{(}) = Z_3$
$Z_6 = \{S \to \mathtt{(}\, S \cdot \mathtt{)}, S \to \mathtt{(}\, S \cdot \mathtt{)}\, S\}$			$\delta(Z_6, \mathtt{)}) = Z_8$
$Z_7 = \{P \to \mathtt{(}\, S \cdot \mathtt{)}, P \to \mathtt{(}\, S \cdot \mathtt{)}\, S\}$			$\delta(Z_7, \mathtt{)}) = Z_9$
$Z_8 = \{S \to \mathtt{(}\, S \,\mathtt{)}\cdot, S \to \mathtt{(}\, S \,\mathtt{)} \cdot S\} \cup \cdots$	$\delta(Z_8, S) = Z_{10}$	$\delta(Z_8, \mathtt{a}) = Z_2$	$\delta(Z_8, \mathtt{(}) = Z_3$
$Z_9 = \{P \to \mathtt{(}\, S \,\mathtt{)}\cdot, P \to \mathtt{(}\, S \,\mathtt{)} \cdot S\} \cup \cdots$	$\delta(Z_9, S) = Z_{11}$	$\delta(Z_9, \mathtt{a}) = Z_2$	$\delta(Z_9, \mathtt{(}) = Z_3$
$Z_{10} = \{S \to \mathtt{(}\, S \,\mathtt{)}\, S \cdot\}$			
$Z_{11} = \{P \to \mathtt{(}\, S \,\mathtt{)}\, S \cdot\}$			

$S_0 \to S$
$S \to \mathtt{a} \mid \mathtt{(}\, S \,\mathtt{)} \mid \mathtt{a}\, P \mid \mathtt{(}\, S \,\mathtt{)}\, S$
$P \to \mathtt{(}\, S \,\mathtt{)} \mid \mathtt{(}\, S \,\mathtt{)}\, S$

Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- O que resulta em

$Z_0 = \{S_0 \to \cdot\, S\,\$\} \cup \cdots$	$\delta(Z_0, S) = Z_1$	$\delta(Z_0, \mathtt{a}) = Z_2$	$\delta(Z_0, \mathtt{(}) = Z_3$
$Z_1 = \{S_0 \to S \cdot \$\}$			
$Z_2 = \{S \to \mathtt{a}\cdot, S \to \mathtt{a} \cdot P\} \cup \cdots$		$\delta(Z_2, P) = Z_4$	$\delta(Z_2, \mathtt{(}) = Z_5$
$Z_3 = \{S \to \mathtt{(} \cdot S \mathtt{)}, S \to \mathtt{(} \cdot S \mathtt{)}\, S\} \cup \cdots$	$\delta(Z_3, S) = Z_6$	$\delta(Z_3, \mathtt{a}) = Z_2$	$\delta(Z_3, \mathtt{(}) = Z_3$
$Z_4 = \{S \to \mathtt{a}\, P \cdot\}$			
$Z_5 = \{P \to \mathtt{(} \cdot S \mathtt{)}, P \to \mathtt{(} \cdot S \mathtt{)}\, S\} \cup \cdots$	$\delta(Z_5, S) = Z_7$	$\delta(Z_5, \mathtt{a}) = Z_2$	$\delta(Z_5, \mathtt{(}) = Z_3$
$Z_6 = \{S \to \mathtt{(}\, S \cdot \mathtt{)}, S \to \mathtt{(}\, S \cdot \mathtt{)}\, S\}$			$\delta(Z_6, \mathtt{)}) = Z_8$
$Z_7 = \{P \to \mathtt{(}\, S \cdot \mathtt{)}, P \to \mathtt{(}\, S \cdot \mathtt{)}\, S\}$			$\delta(Z_7, \mathtt{)}) = Z_9$
$Z_8 = \{S \to \mathtt{(}\, S \,\mathtt{)}\cdot, S \to \mathtt{(}\, S \,\mathtt{)} \cdot S\} \cup \cdots$	$\delta(Z_8, S) = Z_{10}$	$\delta(Z_8, \mathtt{a}) = Z_2$	$\delta(Z_8, \mathtt{(}) = Z_3$
$Z_9 = \{P \to \mathtt{(}\, S \,\mathtt{)}\cdot, P \to \mathtt{(}\, S \,\mathtt{)} \cdot S\} \cup \cdots$	$\delta(Z_9, S) = Z_{11}$	$\delta(Z_9, \mathtt{a}) = Z_2$	$\delta(Z_9, \mathtt{(}) = Z_3$
$Z_{10} = \{S \to \mathtt{(}\, S \,\mathtt{)}\, S \cdot\}$			
$Z_{11} = \{P \to \mathtt{(}\, S \,\mathtt{)}\, S \cdot\}$			

$R_1 = S \to \mathtt{a}$
$R_2 = S \to \mathtt{(}\, S \,\mathtt{)}$
$R_3 = S \to \mathtt{a}\, P$
$R_4 = S \to \mathtt{(}\, S \,\mathtt{)}\, S$
$R_5 = P \to \mathtt{(}\, S \,\mathtt{)}$
$R_6 = P \to \mathtt{(}\, S \,\mathtt{)}\, S$

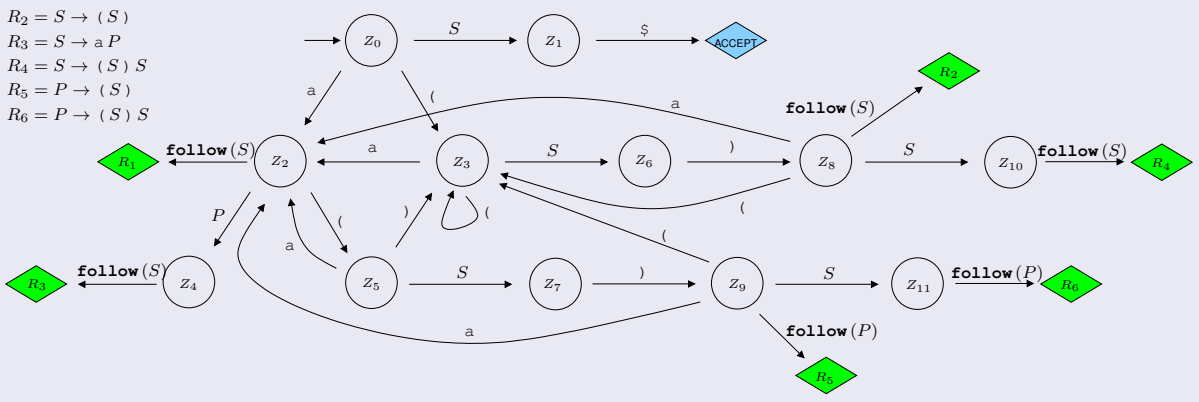


Tabela de decisão de um reconhecedor ascendente
Exemplo #3 (cont.)

- E finalmente a tabela de decisão

	a	(	)	$	S	P
Z_0	*shift*, Z_2	*shift*, Z_3			Z_1	
Z_1				*ACCEPT*		
Z_2		*shift*, Z_5	*reduce* $S \rightarrow$ a	*reduce* $S \rightarrow$ a		Z_4
Z_3	*shift*, Z_2	*shift*, Z_3			Z_6	
Z_4			*reduce* $S \rightarrow$ a P	*reduce* $S \rightarrow$ a P		
Z_5	*shift*, Z_2	*shift*, Z_3			Z_7	
Z_6			*shift*, Z_8			
Z_7			*shift*, Z_9			
Z_8	*shift*, Z_2	*shift*, Z_3	*reduce* $S \rightarrow$ (S)	*reduce* $S \rightarrow$ (S)	Z_{10}	
Z_9	*shift*, Z_2	*shift*, Z_3	*reduce* $P \rightarrow$ (S)	*reduce* $P \rightarrow$ (S)	Z_{11}	
Z_{10}			*reduce* $S \rightarrow$ (S) S	*reduce* $S \rightarrow$ (S) S		
Z_{11}			*reduce* $P \rightarrow$ (S) S	*reduce* $P \rightarrow$ (S) S		

Tabela de decisão de um reconhecedor ascendente
Exercício

Q Determine-se a tabela de decisão para um reconhecedor ascendente com *lookahead* 1 da gramática seguinte

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

$$A \rightarrow \texttt{a} \mid A\ A\ \texttt{b}$$

$$B \rightarrow B\ B\ \texttt{a} \mid \texttt{b}$$

Compiladores

Gramáticas de atributos

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

DETI, Universidade de Aveiro

Ano letivo de 2022-2023

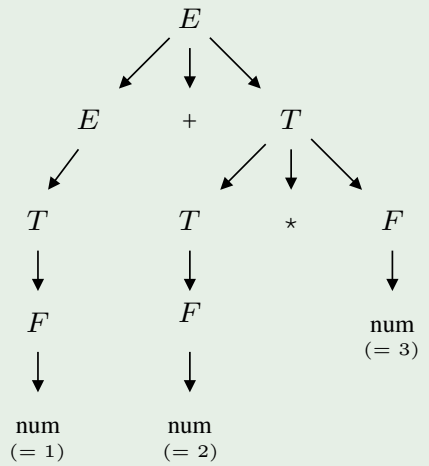
Sumário

Conteúdo semântico
Ilustração com uma expressão aritmética

- Considere a gramática seguinte, onde num é um *token* que representa um número

$$E \rightarrow E \ + \ T \mid T$$
$$T \rightarrow T \ * \ F \mid F$$
$$F \rightarrow \text{num} \mid (\ E \ )$$

- Desenhe-se a árvore de derivação da palavra `"1+2*3"`
- Como dar significado a esta árvore?
  1. Associando a cada símbolo um atributo que armazene o valor que a sub-árvore de que é raiz representa
  2. Relacionando os atributos associados aos símbolos de cada produção através de regras de cálculo

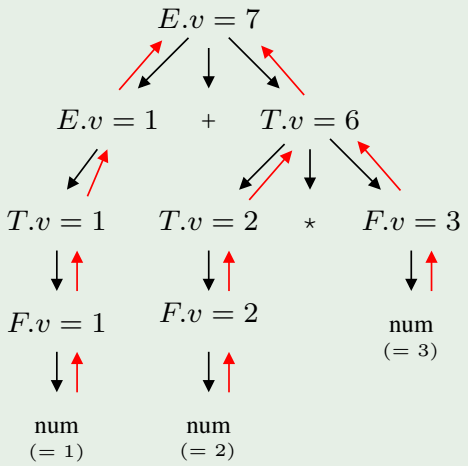


Conteúdo semântico
Ilustração com uma expressão aritmética

- Considere a gramática seguinte, onde num é um *token* que representa um número

$$E \rightarrow E \ + \ T \mid T$$
$$T \rightarrow T \ * \ F \mid F$$
$$F \rightarrow \text{num} \mid (\ E \ )$$

- Desenhe-se a árvore de derivação da palavra `"1+2*3"`
- Como dar significado a esta árvore?
  1. Associando a cada símbolo um atributo que armazene o valor que a sub-árvore de que é raiz representa
  2. Relacionando os atributos associados aos símbolos de cada produção através de regras de cálculo

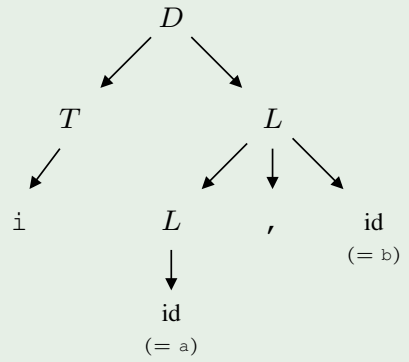


- As setas vermelhas representam dependência entre atributos
  - o sentido indica qual influencia qual

Conteúdo semântico
Ilustração com uma declaração de variáveis

- Considere a gramática seguinte, onde id é um *token* que representa o nome de uma variável

$$D \rightarrow T\ L$$
$$T \rightarrow \texttt{i} \mid \texttt{f}$$
$$L \rightarrow \text{id} \mid L\ \texttt{,}\ \text{id}$$

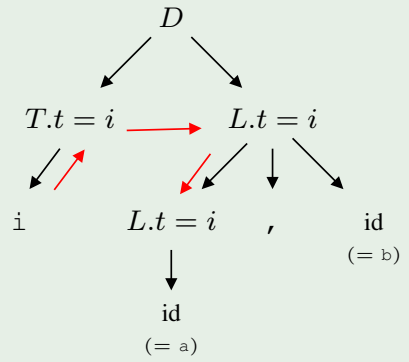


- desenhe-se a árvore de derivação da palavra `i a,b`
- Associe-se
  - a T e L um atributo t que armazene o tipo

Conteúdo semântico
Ilustração com uma declaração de variáveis

- Considere a gramática seguinte, onde id é um *token* que representa o nome de uma variável

$$D \rightarrow T\ L$$
$$T \rightarrow \texttt{i} \mid \texttt{f}$$
$$L \rightarrow \text{id} \mid L\ \texttt{,}\ \text{id}$$



- desenhe-se a árvore de derivação da palavra `i a,b`
- Associe-se
  - a T e L um atributo t que armazene o tipo
- As setas vermelhas representam dependência entre atributos
  - o sentido indica qual influencia qual

Gramática de atributos
Atributos, regras semânticas e definição semântica

- A análise sintática *per se* não atribui um significado às produções de uma gramática
  - É esse o papel da *gramática de atributos*
  - Isso é feito através de **atributos** e de **regras semânticas**
- Os atributos estão associados aos símbolos da gramática (terminais ou não terminais)
  - Cada símbolo terminal ou não terminal pode ter associado um conjunto de zero ou mais atributos
  - Um atributo pode ser uma palavra, um número, um tipo, uma posição de memória, ...
- As regras semânticas estão associadas às produções da gramática
  - Determinam os valores de atributos de símbolos não terminais em função de outros atributos
  - Podem ter efeitos laterais (alteração de uma estrutura de dados, ...)
- Uma **definição semântica** é composta por
  - uma gramática independente de contexto
  - um conjunto de atributos associados aos seus símbolos
  - um conjunto de regras semânticas associadas às suas produções
- Usar-se-á com o mesmo significado o termo **gramática de atributos**

Gramática de atributos
Regras semânticas

Seja $G = (T, N, S, P)$ uma gramática independente do contexto

- A cada produção $A \rightarrow B_1\, B_2\, \cdots\, B_n \in P$, com $B_i \in (T \cup N)^*$, podem associar-se regras semânticas para o cálculo dos valores dos atributos de símbolos não terminais

$$b = f(c_1, c_2, \cdots, c_n)$$

  onde
  - b é um atributo do símbolo A ou de um dos símbolos não terminais presentes em $B_1\, B_2\, \cdots\, B_n$
  - $c_1, c_2, \cdots, c_n$ são atributos dos símbolos que ocorrem na produção
- Podem ainda associar-se regras semânticas com efeitos colaterais

$$g(c_1, c_2, \cdots, c_n)$$

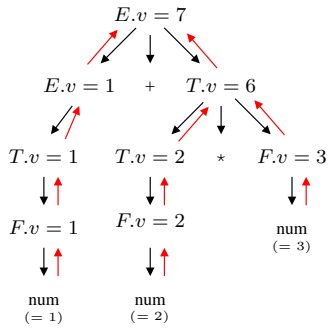
  - Embora este caso possa considerar-se o anterior atuando sobre um atributo fictício

Gramática de atributos
Tipos de atributos

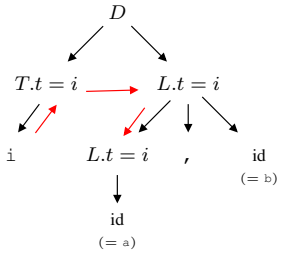
- Os atributos podem ser classificados como **sintetizados** ou **herdados**
- Considere-se uma produção $A \to B_1\, B_2\, \cdots\, B_n \in P$, com $B_i \in (T \cup N)^*$, e uma função de cálculo de um atributo associada a essa produção

$$b = f(c_1, c_2, \cdots, c_n)$$

  - O atributo b diz-se **sintetizado** se b está associado a A e todos os c_j, com $j = 1, 2, \cdots, n$, estão associados a símbolos do corpo da produção
  - O atributo b diz-se **herdado** se b está associado a um dos símbolos não terminais do corpo da produção



- Todos os atributos são sintetizados

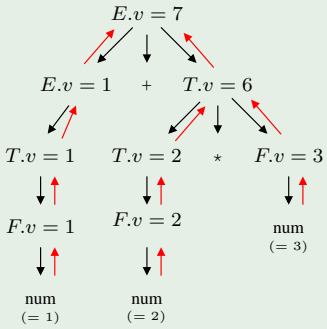


- $T.t$ é sintetizado
- $L.t$ é herdado

Gramática de atributos
Representação

- Uma gramática de atributos pode ser representada por uma tabela em que se associam as regras semânticas às produções da gramática

- Para o exemplo das expressões aritméticas, tem-se



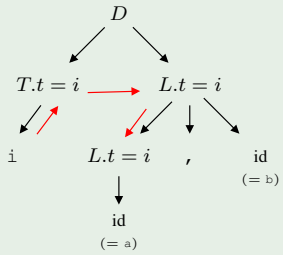
Produções	**Regras semânticas**
$F \to$ num	$F.v =$ num.v
$F \to$ (E)	$F.v = E.v$
$T \to F$	$T.v = F.v$
$T_1 \to T_2$ * F	$T_1.v = T_2.v * F.v$
$E \to T$	$E.v = T.v$
$E_1 \to E_2$ + T	$E_1.v = E_2.v + T.v$

- Note que se assume que o símbolo terminal num tem um atributo chamado v com o valor correspondente.
- O ANTLR não suporta atributos nos terminais (*tokens*)

Gramática de atributos
Representação

- Uma gramática de atributos pode ser representada por uma tabela em que se associam as regras semânticas às produções da gramática

- Para o exemplo da declaração de variáveis, tem-se



Produções	Regras semânticas
$T \rightarrow i$	$T.t$ = `int`
$T \rightarrow f$	$T.t$ = `float`
$D \rightarrow T\ L$	$L.t = T.t$
$L_1 \rightarrow L_2$, id	$L_2.t = L_1.t$ `addsym`(id.n, $L_1.t$)
$L \rightarrow$ id	`addsym`(id.n, $L.t$)

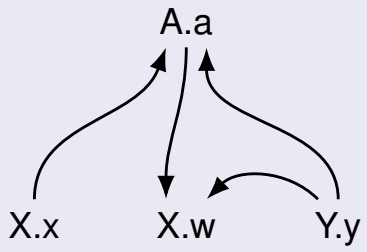
- Assume-se que o símbolo terminal id tem um atributo chamado n com o valor correspondente
- Neste caso, para além do cálculo de atributos, faz-se a inserção numa tabela de símbolos (`addsym`)

Avaliação dirigida pela sintaxe

- Numa **avaliação dirigida pela sintaxe** o cálculo dos atributos é feito à medida que é feita a análise sintática.
- Num analisador sintático ascendente (caso do bison) todos os atributos têm de ser sintetizados
- Num analisador sintático descendente (caso do Antlr) além de sintetizados os atributos podem ser herdados, desde que de símbolos à esquerda ou do símbolo pai
- para definir a ordem de cálculo dos atributos, usa-se o **grafo de dependências**

$$A \rightarrow X\ \ Y$$

$$A.a = f(X.x, Y.y)$$
$$X.w = g(A.a, Y.y)$$



- Aqui as setas apontam no sentido das dependências