Linguagens Formais e Autómatos / Compiladores

Análise sintática descendente

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

DETI, Universidade de Aveiro

Sumário

Análise sintática
Introdução

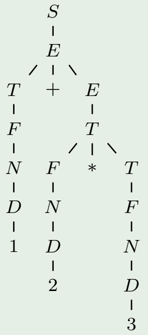
- Dada uma gramática $G = (T, N, P, S)$ e uma palavra $u \in T^*$, o papel da análise sintática é:
  - descobrir uma derivação que a partir de S produza u
  - gerar uma árvore sintática que transforme S (a raiz) em u (as folhas)
- Se nenhuma derivação/árvore existir, então $u \notin L(G)$
- A análise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

Análise sintática descendente
Exemplo

- Considere a gramática

$$
\begin{aligned}
S &\rightarrow E \\
E &\rightarrow T + E \mid T \\
T &\rightarrow F \star T \mid F
\end{aligned}
\qquad\qquad
\begin{aligned}
F &\rightarrow N \mid (\ E\ ) \\
N &\rightarrow D \mid N\ D \\
D &\rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3}
\end{aligned}
$$

- Desenhe-se a árvore de derivação da palavra `1+2*3`



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 análise
    - No caso da tabela, os símbolos não terminais transformam-se no alfabeto da pilha
  - Não requer *backtracking*
  - A produção a aplicar a cada momento baseia-se em *tokens* da entrada que ainda não foram consumidos (*lookahead*)
  - São designados $LL(k)$
    - k o número 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 conjuntos de análise
    - **first**, **follow** e **predict**

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

- Sobre o alfabeto $\{\mathtt{a}, \mathtt{b}\}$, considere linguagem
  $$L_1 \quad = \quad \{\mathtt{a}^n\mathtt{b}^n \;:\; n \geq 0\}$$
  descrita pela gramática
  $$S \rightarrow \mathtt{a}\ S\ \mathtt{b} \mid \varepsilon$$
  Construa um programa com *lookahead* de 1, em que o símbolo não terminal S seja uma função recursiva, que reconheça a linguagem L_1.

```
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()

  if (c != '\n') 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-se escolher $S \rightarrow$ `a` S `b` se o `lookahead` for `a`
  - e $S \rightarrow \varepsilon$ se o `lookahead` for `\$` ou `b`

  No programa, `$`, 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
Exemplo #2

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

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

descrita pela gramática

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

Construa um programa em que os símbolos não terminais sejas funções recursivas que reconheça a linguagem L_2.

- O programa terá 3 funções recursivas, A, B e S, semelhantes à função S do exemplo anterior
- Em A, deve escolher-se $A \to \texttt{a}$ se `lookahead` for `a` e $A \to \texttt{b}\; A\; A$ se for `b`
- Em B, deve escolher-se $B \to \texttt{b}$ se `lookahead` for `b` e $B \to \texttt{a}\; B\; B$ se for `a`
- Em S, deve escolher-se $S \to \texttt{a}\; B\; S$ se `lookahead` for `a`, $S \to \texttt{b}\; A\; S$ se for `b` e $S \to \varepsilon$ se for `$` (este último, mais tarde saber-se-á porquê)

Analisador (*parser*) recursivo-descendente

Exemplo #2a

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

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

descrita pela gramática

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

Construa um programa em que os símbolos não terminais sejas funções recursivas que reconheça a linguagem L_{2a}.

- O programa terá 3 funções recursivas, A, B e S, semelhantes à função S do exemplo anterior
- Escolher $S \to \varepsilon$ quando `lookahead` for `$` pode não resolver
- Por exemplo, com o `lookahead` igual a `a`, há situações em que se tem de escolher $S \to \mathtt{a}\, B$ e outras $S \to \varepsilon$
- É o que acontece com a entrada `bbaa`

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

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

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

  descrita pela gramática

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

  Construa um programa em que os símbolos não terminais sejas funções recursivas que reconheça a linguagem L_{2b}

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

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

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

$$L_3 = \{\mathtt{a}^n \mathtt{b}^n \; : \; n \geq 1\}$$

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_3.

- Como escolher entre as duas produções se ambas começam pelo mesmo símbolo?
- Há duas abordagens:
  - Pôr em evidência os fatores comuns à esqueda
  - Aumentar o número de símbolos de *lookahead*

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

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

  $$L_4 = \{(\texttt{ab})^n : n \geq 1\}$$

  que é descrita pela gramática

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

  Construa um programa em que o símbolo não terminal S seja uma função recursiva que reconheça a linguagem L_3.

- Escolher a primeira produção cria um ciclo infinito, por causa da recursividade à esquerda

- Solução: 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 $A \rightarrow \alpha$?

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

Fatorização à esquerda
Exemplo de ilustração

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

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

descrita pela gramática

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

Obtenha uma gramática equivalente, pondo em evidência o $\mathtt{a}$

- Relaxando a definição *standard* de gramática que se tem usado, pode obter-se

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

- Que correspone à gramatica

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

- O Antlr permite-o

Eliminação de recursividade à esquerda
Recursividade direta simples

- A gramática seguinte representa genericamente a recursividade direta simples à esquerda

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

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

$$A \Rightarrow A\,\alpha \Rightarrow A\,\alpha\,\alpha \Rightarrow A\,\alpha\,\cdots\,\alpha\,\alpha \Rightarrow \beta\,\alpha\,\cdots\,\alpha\,\alpha$$

- Ou seja

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

- Que corresponde ao β seguido do fecho de α, dando a gramática

$$\begin{array}{lcl} A & \rightarrow & \beta\ X \\ X & \rightarrow & \varepsilon \\ & \mid & \alpha\ X \end{array}$$

- α e β representam sequências de símbolos
- β não pode começar por A

Eliminação de recursividade à esquerda
Exemplo de recursividade direta simples

- Para a gramática

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

  obtenha-se uma gramática equivalente sem recursividade à esquerda
- Aplicando a estratégia anterior, tem-se

$$S \Rightarrow S\,\mathtt{a}\,\mathtt{b} \Rightarrow S\,\mathtt{a}\,\mathtt{b} \cdots \mathtt{a}\,\mathtt{b} \Rightarrow \mathtt{a}\,\mathtt{b}\,\mathtt{c}\,\mathtt{a}\,\mathtt{b} \cdots \mathtt{a}\,\mathtt{b}$$

- Ou seja

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

- Que corresponde à gramática

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

Eliminação de recursividade à esquerda
Recursividade direta múltipla

- A gramática seguinte representa genericamente a recursividade direta múltipla à esquerda

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

- Aplicando a estratégia anterior, tem-se

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

- Que corresponde à gramática

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

- α_i e β_i representam sequências de símbolos
- os β_i não podem começar por A

- Em Antlr é 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 de recursividade direta múltipla

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

$$
\begin{aligned}
S \;\rightarrow\;& S\ \texttt{a}\ \texttt{b} \mid S\ \texttt{c} \\
\mid\;& \texttt{a}\ \texttt{b} \mid \texttt{c}\,\texttt{c}
\end{aligned}
$$

- As palavras da linguagem são da forma

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

- Obtendo-se a gramática

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

Eliminação de recursividade à esquerda
Recursividade indireta

- Aplique-se o procedimento anterior à gramática

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

- O resultado seria

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

- A recursividade não foi eliminada

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

- Como resolver o caso de a recursividade à esquerda ser indireta?

- Aplicado às produções começadas por A
- com $\alpha = \mathtt{c}$ e $\beta_1 = S\ \mathtt{d}$ e $\beta_2 = \varepsilon$

- Embora S não comece por A, pode transformar-se em algo que começa por A

Eliminação de recursividade à esquerda
Recursividade indireta

- Considere a gramática (genérica) seguinte

$$\begin{array}{rcllcl} A_1 & \to & A_i & \alpha_1 & \mid & \beta_1 \\ A_2 & \to & A_j & \alpha_2 & \mid & \beta_2 \\ & \cdots & & & & \\ A_n & \to & A_k & \alpha_n & \mid & \beta_n \end{array}$$

- 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 começada por 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
Recursividade indireta

- Apliquemos este novo procedimento à gramática seguinte, estabelecendo a ordem S, A

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

- A produção $A \to S\ \mathtt{d}$ viola a regra definida, pelo que nela S é substituído por $(A\ \mathtt{a} \,|\, \mathtt{b}$, obtendo-se

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

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

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

- Nas produções começadas por S não é preciso atuar

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

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

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

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

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

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

$$\mathtt{follow}\,(A) = \{t \in T_\$ \;:\; S\,\$ \;\Rightarrow^*\; \gamma\, A\, t\, \omega\}$$

Conjunto first
Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

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

- Este algoritmo pode não convergir se a gramática tiver recursividade à esquerda

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:

$$\$ \in \texttt{follow}(S)$$

$$\begin{array}{l}
\textbf{if } (A \rightarrow \alpha B \in P) \textbf{ then} \\
\quad \texttt{follow}(B) \supseteq \texttt{follow}(A) \\
\textbf{else if } (A \rightarrow \alpha B \beta \in P) \textbf{ then} \\
\quad \textbf{if } (\varepsilon \notin \texttt{first}(\beta)) \textbf{ then} \\
\quad\quad \texttt{follow}(B) \supseteq \texttt{first}(\beta) \\
\quad \textbf{else} \\
\quad\quad \texttt{follow}(B) \supseteq (\texttt{first}(\beta) - \{\varepsilon\}) \cup \texttt{follow}(A)
\end{array}$$

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

---

- Note que $\supseteq$ significa **contém**

Reconhecedor descendente preditivo
Tabela de análise (*parsing*)

- Para uma gramática $G = (T, N, P, S)$ e um *lookahead* de 1, o reconhecedor descendente pode basear-se numa tabela de análise
- Corresponde a uma função $\tau : N \times T_\$ \rightarrow \wp(P)$, onde $\wp(P)$ 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 com as células vazias*
foreach $(A \rightarrow \alpha) \in P$
  foreach $t \in$ **predict** $(A \rightarrow \alpha)$
    add $(A \rightarrow \alpha)$ to $\tau(A, t)$

- $T_\$ = T \cup \{\$\}$

Tabela de análise
Exemplo #1

- Considere a gramatica

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

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

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

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

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

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

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

Tabela de análise

	a	b	$
S	a S b	ε	ε

- 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 análise
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 análise de um reconhecedor descendente desta linguagem com *lookahead* de 1

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

Tabela de análise

	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 análise
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 análise de um reconhecedor descendente desta linguagem com *lookahead* de 1

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

Tabela de análise

	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 reconhecedor inviável para um *lookahead* de 1

Tabela de análise
Exemplo #2

- Considere a gramatica

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

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

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

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

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

Tabela de análise

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

- As células $(S, \mathtt{a})$ e $(S, \mathtt{b})$ têm duas produções cada, o que torna o reconhecedor inviável para um *lookahead* de 1

Tabela de análise
Exemplo #3

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

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

- Preencha a tabela de análise de um reconhecedor descendente, com *lookahead* de 1, que reconheça a linguagem L_4.
- Antes de calcular os conjuntos **`predict`** é necessário começar por fatorizar à esquerda, por causa das produções começadas por L

Tabela de análise
Exemplo #3 (cont.)

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

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

Tabela de decisão

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

Tabela de análise
Processo de reconhecimento

- Processo de reconhecimento da paravra `iv,v;`

consumido	na entrada	pilha	próxima ação
	`iv,v;`$	$D\,\$$	$D \to T\,L$ `;`
	`iv,v;`$	$T\,L$ `;` $	$T \to$ `i`
	`iv,v;`$	`i` L `;` $	match `i`
`i`	`v,v;`$	L `;` $	$L \to$ `v` X
`i`	`v,v;`$	`v` X `;` $	match `v`
`iv`	`,v;`$	X `;` $	$X \to$ `,` L
`iv`	`,v;`$	`,` L `;` $	match `,`
`iv,`	`v;`$	L `;` $	$L \to$ `v` X
`iv,`	`v;`$	`v` X `;` $	match `v`
`iv,v`	`;`$	X `;` $	$X \to \varepsilon$
`iv,v`	`;`$	`;` $	match `;`
`iv,v;`	$	$	match $
`iv,v;`$			accept