Linguagens Formais e Autómatos / Compiladores

Análise sintática descendente

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

DETI, Universidade de Aveiro

Ano letivo de 2020-2021

Sumário

Análise sintática
Introdução

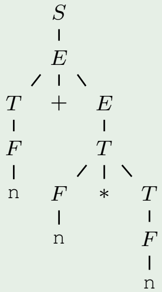
- Dada uma gramática $G = (T, N, P, S)$ e uma palavra $u \in T^*$, o papel da análise sintática é:
  - descobrir uma derivação que a partir de S produza u
  - gerar uma árvore de derivação (*parse tree*) que transforme S (a raiz) em u (as folhas)
- Se nenhuma derivação/árvore existir, então $u \notin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

$$
\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & T \ + \ E \ | \ T \\
T & \rightarrow & F \ \star \ T \ | \ F \\
F & \rightarrow & n \ | \ (\ E \ )
\end{array}
$$

- Desenhe-se a árvore de derivação da palavra `n+n*n`



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 elementos de análise
    - os conjuntos **first**, **follow** e **predict**

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

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

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

  descrita pela gramática

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

- Construa-se um programa com *lookahead* de 1, em que o símbolo não terminal S seja uma função recursiva, que reconheça a linguagem L.

```c
int lookahead;

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

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

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

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

void epsilon()
{
}

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

Analisador (*parser*) recursivo-descendente
Análise do exemplo #1

No programa anterior:

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

  No programa anterior, o símbolo `$`, marcador de fim de entrada, corresponde ao `\n`
- Uma palavra é aceite pelo programa se e só se
  ```
  S(); eat($)
  ```
  não der erro.

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

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

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

descrita pela gramática

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

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

- O programa terá 3 funções recursivas, A, B e S, semelhantes à função S do exemplo anterior
- Em A, deve escolher-se $A \to \mathtt{a}$ se `lookahead` for `a` e $A \to \mathtt{b}\,A\,A$ se for `b`
- Em B, deve escolher-se $B \to \mathtt{b}$ se `lookahead` for `b` e $B \to \mathtt{a}\,B\,B$ se for `a`
- Em S, deve escolher-se $S \to \mathtt{a}\,B\,S$ se `lookahead` for `a`, $S \to \mathtt{b}\,A\,S$ 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 $\{\texttt{a}, \texttt{b}\}$, considere linguagem

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

descrita pela gramática

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

- 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
- 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 $S \to \texttt{a}\ B$ 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

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

  descrita pela gramática

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

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

- 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

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

  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

    $S \rightarrow \mathtt{a}\ X$

    $X \rightarrow S\ \mathtt{b} \mid \mathtt{b}$
  - 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
Exemplo #4

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

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

que é 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

- 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 $A \to \alpha$?

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

Fatorização à esquerda
Exemplo de ilustração

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

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

  descrita pela gramática

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

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

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

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

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

- Esta gramática permite a contrução 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

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

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

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

- Ou seja

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

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

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

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

Eliminação de recursividade à esquerda
Exemplo de recursividade direta simples

- Para a gramática

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

  obtenha-se uma gramática equivalente sem recursividade à esquerda

- Aplicando a estratégia anterior, tem-se

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

- Ou seja

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

- Que corresponde à gramática

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

Eliminação de recursividade à esquerda
Recursividade direta múltipla

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

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

- Aplicando a estratégia anterior, tem-se

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

- Que corresponde à gramática

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

- 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

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

- 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

$$\begin{aligned} S &\to \texttt{a}\,\texttt{b}\,X \mid \texttt{c}\,\texttt{c}\,X \\ X &\to \varepsilon \\ &\quad \mid \texttt{a}\,\texttt{b}\,X \mid \texttt{c}\,X \end{aligned}$$

Eliminação de recursividade à esquerda
Recursividade indireta

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

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

- 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

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

- Existe de forma indireta
- Como resolver a recursividade à esquerda indireta?

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

Eliminação de recursividade à esquerda
Recursividade indireta

- Considere a gramática (genérica) seguinte, em que alguns dos α_i, β_i, $\cdots$, Ω_i podem começar por A_j, com $i, j = 1, 2, \cdots, n$

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

- 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 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
Recursividade indireta

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

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

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

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

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

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

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

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

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

- O conjunto $\texttt{first}(\alpha)$ 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:

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

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

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

Conjunto `first`
Algoritmo de cálculo

- Trata-se de um algoritmo recursivo

```
first(α) {
    if (α = ε) then
        return {ε}
    h = head (α)      # com |h| = 1
    ω = tail (α)      # tal que α = h ω
    if ( h ∈ T ) then
        return {h}
    else
        return   ⋃              first(β_i ω)
             (h→β_i)∈P
}
```

$$\texttt{first}(\alpha) = \begin{cases} \{\varepsilon\} & \alpha = \varepsilon \\ \{h\} & h \in T \\ \bigcup_{(h\to\beta_i)\in P} \texttt{first}(\beta_i\,\omega) & \text{else} \end{cases}$$

- 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:

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

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

---

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

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 $\wp(P)$ 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 decisão
Exemplo #1

- Considere a gramatica

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

- Preencha a tabela de decisão 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 decisão

	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 decisão
Exemplo #2

- Considere a gramatica

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

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

$\mathbf{predict}(S \to \mathtt{a}\ B\ S) = \{\mathtt{a}\}$
$\mathbf{predict}(S \to \mathtt{b}\ A\ S) = \{\mathtt{b}\}$
$\mathbf{predict}(S \to \varepsilon) = \{\$\}$
$\mathbf{predict}(A \to \mathtt{a}) = \{\mathtt{a}\}$
$\mathbf{predict}(A \to \mathtt{b}\ A\ A) = \{\mathtt{b}\}$
$\mathbf{predict}(B \to \mathtt{b}) = \{\mathtt{b}\}$
$\mathbf{predict}(B \to \mathtt{a}\ B\ B) = \{\mathtt{a}\}$

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

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

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

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

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

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

$\mathbf{predict}(A \to \mathtt{a}\ S) = \{\mathtt{a}\}$

$\mathbf{predict}(A \to \mathtt{b}\ A\ A) = \{\mathtt{b}\}$

$\mathbf{predict}(B \to \mathtt{b}\ S) = \{\mathtt{b}\}$

$\mathbf{predict}(B \to \mathtt{a}\ B\ B) = \{\mathtt{a}\}$

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

Tabela de decisão
Exemplo #2b

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

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

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

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

Tabela de decisão

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

Tabela de decisão
Exemplo #3

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

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

- Preencha a tabela de decisão 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 decisão
Exemplo #3 (cont.)

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

$\textbf{predict}(D \rightarrow T\,L\,\texttt{;}) = ?$

$\quad \textbf{first}(T\,L\,\texttt{;}) = ?$

$\quad \textbf{first}(T) = \textbf{first}(\texttt{i}) \cup \textbf{first}(\texttt{f}) = \{\texttt{i}\} \cup \{\texttt{f}\}$

$\quad \therefore\ \textbf{first}(T\,L\,\texttt{;}) = \{\texttt{i},\texttt{f}\}$

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

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

$\quad \textbf{first}(\texttt{i}) = \{\texttt{i}\}$

$\quad \therefore\ \textbf{predict}(T \rightarrow \texttt{i}) = \{\texttt{i}\}$

$\textbf{predict}(T \rightarrow \texttt{f}) = \{\texttt{f}\}$

$\textbf{predict}(L \rightarrow \texttt{v}\,X) = ?$

$\quad \textbf{first}(\texttt{v}\,X) = \{\texttt{v}\}$

$\quad \therefore\ \textbf{predict}(L \rightarrow \texttt{v}\,X) = \{\texttt{v}\}$

$\textbf{predict}(X \rightarrow \varepsilon) = ?$

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

$\quad \therefore\ \textbf{predict}(X \rightarrow \varepsilon) = \textbf{follow}(X)$

$\quad \textbf{follow}(X) = \textbf{follow}(L) = \{\texttt{;}\}$

$\quad \therefore\ \textbf{predict}(X \rightarrow \varepsilon) = \{\texttt{;}\}$

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

Tabela de decisão
Exemplo #3 (cont.)

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

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

Tabela de decisão

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

- As células vazias são situações de erro