Compiladores (CC3001)
Aula 7: Análise sintática *top-down*

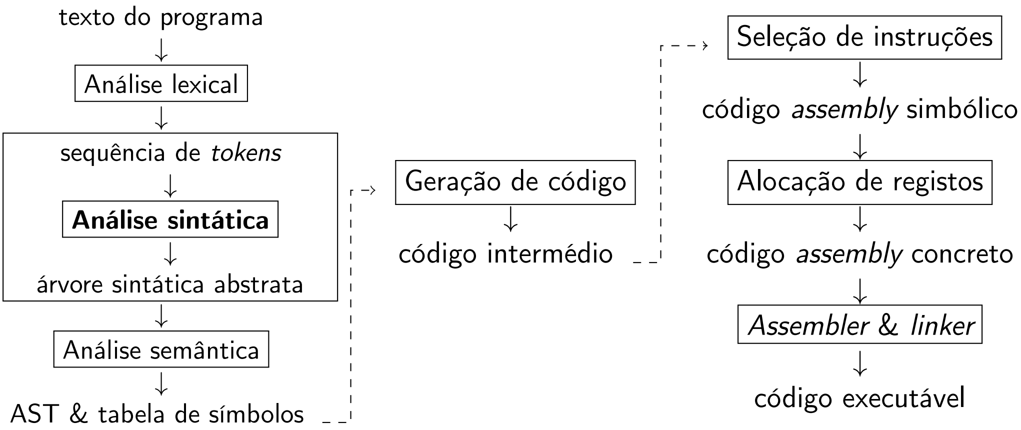
Mário Florido

DCC/FCUP

2024

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Re-lembrar: fases dum compilador



Esta aula

Análise sintática

Descida recursiva

Gramáticas preditivas

- Inferir a árvore sintática a partir da sequência de *tokens*
  (ou rejeitar o programa com erro sintático)
- Principais abordagens:

  top-down construir a árvore partindo da raiz (não-terminal inicial S)
  bottom-up construir a árvore partindo das folhas (símbolos terminais)
- Um analisador sintático é tambem designado por *parser* e análise sintática por *parsing*
- Nesta aula vamos ver análise sintática *top-down*:
  - descida recursiva (*recursive descent parsing*)
  - *parsing* preditivo

Esta aula

Análise sintática

Descida recursiva

Gramáticas preditivas

Implementar o analisador sintático diretamente numa linguagem de programação:

- Cada símbolo não-terminal corresponde a uma função
- Cada produção corresponde a uma cláusula da função
  (se a produção for recursiva, a função será recursiva também)
- Consumimos símbolos terminais da esquerda para a direita
- Decidimos qual a produção a usar pelo próximo símbolo terminal

Construir um analisador sintáticos para "programas" deste género:

```
begin
 if 1=1 then
    begin
    print 0=11 ;  print 123=4
    end
 else
    print 11=42
end
```

Exemplo (cont.)

Gramática independente de contexto:

$$
\begin{array}{ll}
S \to \text{if } E \text{ then } S \text{ else } S & \qquad L \to \text{end} \\
S \to \text{begin } S \; L & \qquad L \to \; ; \; S \; L \\
S \to \text{print } E & \qquad E \to \text{num} = \text{num}
\end{array}
$$

Representamos *tokens* como uma enumeração:

```
data Token = IF | THEN | ELSE | BEGIN | END | PRINT | SEMI | NUM | EQ
```

Para cada não-terminal (S, L, E) vamos definir uma função

```
parseS, parseL, parseE :: [Token] -> [Token]
```

Cada função `parse`X:

- recebe a lista de *tokens* de entrada
- retorna a lista *tokens* que não foram consumidos (ou aborta com erro)

Nota: vamos apenas reconhecer as frases da linguagem; mais tarde veremos a construção da árvore sintática.

Implementação em Haskell (cont.)

```
parseS :: [Token] -> [Token]
parseS (first:toks) = case first of
  IF -> let toks1 = parseE toks
            toks2 = consume THEN toks1
            toks3 = parseS toks2
            toks4 = consume ELSE toks3
        in          parseS toks4
  BEGIN -> let toks1 = parseS toks
           in          parseL toks1
  PRINT -> parseE toks
  _ -> error "syntax error"
```

```
parseE :: [Token] -> [Token]
parseE toks
  = let toks1 = consume NUM toks
        toks2 = consume EQUAL toks1
    in          consume NUM toks2

parseL :: [Token] -> [Token]
parseL (first:toks) = case first of
  END -> toks
  SEMI -> let toks1= parseS toks
          in         parseL toks1
  _ -> error "syntax error"
```

- Cada alternativa case corresponde a uma produção da gramática
- Passamos a lista de *tokens* resultante de cada função à seguinte
- Usamos uma função auxiliar para consumir um *token* (ou abortar com erro):

```
consume :: Token -> [Token] -> [Token]
consume tok (first:rest) | tok == first = rest
consume tok _   = error ("expected " ++ show tok)
```

(Segue-se uma demonstração.)

Implementação em C

- A descida recursiva é também facil de implementar em C
- Principal diferença: as funções de *parsing* consomem os *tokens* da entrada-padrão (em vez de transformações `[Token] -> [Token]`)

```
Token getToken(void);  // ler o próximo token da entrada-padrão
```

- Mantemos um *token* de *look-ahead* numa variável de estado global:

```
Token next;              // próximo token (global)
void advance(void) {     // avançar um token
   next = getToken();
}
```

- Cada função de *parsing* decide o que fazer com base no *token* de *look-ahead*
- Usamos uma função auxiliar `consume(...)` para consumir um *token* específico

Implementação em C (cont.)


```
void parse_S(void) {
  switch(next) {
  case IF:
    advance(); parse_E(); consume(THEN);  parse_S();
    consume(ELSE);  parse_S();
    break;
  case BEGIN:
    advance();  parse_S();  parse_L();
    break;
  case PRINT:
    advance(); parse_E();
    break;
  default:
    error("syntax error");
  }
}
```

```
void parse_E(void) {
  consume(NUM);  consume(EQUAL); consume(NUM);
}

void parse_L(void) {
  switch(next) {
  case END:
    advance();
    break;
  case SEMI:
    advance(); parse_S(); parse_L();
    break;
  default:
    error("syntax error");
  }
}
```

Marcador do final

- Um programa correto deve terminar sem *tokens* redundantes no final
- Isto quer dizer que o analisador deve também detetar o final do ficheiro
- Basta verificar se a lista final de *tokens* é vazia:

```
accepted :: [Token] -> Bool
accepted toks = null (parseS toks)
```

- Alternativa:
  - acrescentar um *token* especial \$ que representa fim do ficheiro
  - uma produção $S' \to S\$$
  - S' passa a ser o símbolo inicial da gramática

```
void accepted(void) {
   parse_S();
   consume(EOF);
}
```

Esta aula

Análise sintática

Descida recursiva

Gramáticas preditivas

- Análise sintática por descida recursiva obriga a escolher a produção com base no próximo símbolo terminal
- Se a gramática não respeitar essas condições é necessário re-escreve-la
- Vamos ver um exemplo em que é necessário re-escrever a gramática para puder fazer descida recursiva

Exemplo

Uma gramática para expressões aritméticas.

$$E \to E + T \qquad T \to T * F \qquad F \to \text{num}$$
$$E \to E - T \qquad T \to T \,/\, F \qquad F \to (E)$$
$$E \to T \qquad T \to F$$

Problemas:

- Como escolher qual das produções usar para E e T?
- Como evitar que a recursão à esquerda em E e T entre em ciclo?

Consideremos a definição mais simples:

$$\begin{aligned} E &\to E \ + \ T \\ E &\to T \end{aligned}$$

Observe que E produz somas de termos, i.e. $E \Rightarrow^* T + T + \cdots + T$.

Podemos obter uma definição equivalente introduzindo um símbolo auxiliar E':

$$E \to T \ E'$$

$$\begin{aligned} E' &\to + \ T \ E' \\ E' &\to \varepsilon \end{aligned}$$

Esta definição contém só recursão à direita.

$$E \to T\ E'$$

$$\begin{aligned} E' &\to +\ T\ E' \\ E' &\to \varepsilon \end{aligned}$$

- As produções de E' têm a recursão à direita em vez da esquerda
- Podemos decidir qual a produção usar com base no próximo símbolo:

$$\begin{array}{rl} + & \text{usamos } E' \to +\ T\ E' \\ \text{outro} & \text{usamos } E' \to \varepsilon \end{array}$$

Aplicando estas transformações à gramatica de expressões obtemos:

$$
\begin{array}{lll}
E \to T\ E' & T \to F\ T' & \\
E' \to +\ T\ E' & T' \to *\ F\ T' & F \to \mathsf{num} \\
E' \to -\ T\ E' & T' \to /\ F\ T' & F \to (\ E\ ) \\
E' \to \varepsilon & T' \to \varepsilon &
\end{array}
$$

Desta forma estamos em condições para implementar descida recursiva.

Exercício: implementar o *parser* para esta gramática.

- A gramáticas de expressões final pertence a uma classe designada $LL(1)$: *Left-to-right parse, Leftmost derivation, 1-symbol look-ahead*
- Esta classe contém todas as gramáticas que podemos analisar com descida recursiva
- Para definirmos de forma mais rigorosa esta classe necessitamos de algumas definições matemáticas
- Vamos também ver como implementar um *parser* para estas gramáticas sem recusão usando uma pilha explícita

Conjunto FIRST

Consideramos uma gramática $G = (\Sigma, N, S, P)$ e X um símbolo não-terminal.

Quando o próximo terminal é x, podemos usar a produção $X \to \gamma$ se

$$x \in \text{FIRST}(\gamma)$$

ou seja, se x for um *símbolo inicial* das derivações a partir de γ.

Para escolher entre duas produções $X \to \gamma$ e $X \to \gamma'$ apenas pelo próximo símbolo devemos garantir que não partilham símbolos iniciais:

$$\text{FIRST}(\gamma) \cap \text{FIRST}(\gamma') = \emptyset$$

Definição

$$\mathsf{FIRST}(\gamma) = \{x \in \Sigma : \gamma \Rightarrow^* x\beta, \text{ para algum } \beta\}$$

Ou seja: FIRST(γ) é o conjunto dos símbolos terminais por que começam as palavras derivadas por γ.

- Esta definição não é útil para determinar FIRST(γ) para cada produção $X \to \gamma$
- Vamos ver como calcular diretamente a partir das produções da gramática

Conjunto FIRST (cont.)

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

- Podemos calcular FIRST diretamente para produções não-recursivas, e.g.

$$\text{FIRST}(F) = \{\text{num}, (\}$$

- As produções recursivas têm de respeitar algumas equações; e.g. para T:

$$\begin{aligned} & \text{FIRST}(T) = \text{FIRST}(T * F) \cup \text{FIRST}(F) \\ \iff \; & \text{FIRST}(T) = \text{FIRST}(T) \cup \text{FIRST}(F) \end{aligned}$$

- O menor conjunto solução da equação anterior é:

$$\text{FIRST}(T) = \text{FIRST}(F) = \{\text{num}, (\}$$

Vamos ver como obter estas soluções usando um método iterativo.

- A simplificação

$$\text{FIRST}(T * F) = \text{FIRST}(T)$$

  é válida porque não é possível derivar a sequência vazia ε a partir de T
- Em geral: para calcular FIRST necessitamos de saber quais os não-terminais que podem derivar ε

$$\text{NULLABLE}(X) = \begin{cases} \text{True} & \text{, se } X \Rightarrow^* \varepsilon \\ \text{False} & \text{, caso contrário} \end{cases}$$

- Vamos também exprimir este predicado como equações boleanas a partir da gramática

Equações para FIRST e NULLABLE (cont.)

$$
\begin{aligned}
\text{FIRST}(\varepsilon) &= \emptyset \\
\text{FIRST}(a) &= \{a\} \qquad (a \in \Sigma) \\
\text{FIRST}(\alpha\beta) &= \begin{cases} \text{FIRST}(\alpha) \cup \text{FIRST}(\beta)\,, & \text{se NULLABLE}(\alpha) \\ \text{FIRST}(\alpha)\,, & \text{caso contrário} \end{cases} \\
\text{FIRST}(X) &= \text{FIRST}(\gamma_1) \cup \ldots \cup \text{FIRST}(\gamma_n)\,, \\
& \quad \text{em que } X \to \gamma_i \text{ são todas as produções para } X
\end{aligned}
$$

$$
\begin{aligned}
\text{NULLABLE}(\varepsilon) &= \text{True} \\
\text{NULLABLE}(a) &= \text{False} \qquad (a \in \Sigma) \\
\text{NULLABLE}(\alpha\beta) &= \text{NULLABLE}(\alpha) \wedge \text{NULLABLE}(\beta) \\
\text{NULLABLE}(X) &= \text{NULLABLE}(\gamma_1) \vee \ldots \vee \text{NULLABLE}(\gamma_n) \\
& \quad \text{em que } X \to \gamma_i \text{ são todas as produções para } X
\end{aligned}
$$

Método iterativo:

1. Inicialmente $\text{NULLABLE}(X) := \text{False}$ e $\text{FIRST}(X) := \emptyset$ para todos os símbolos não-terminais
2. Calcular novos valores para os lados direitos das produções usando as equações do *slide* anterior
3. Repetir até estabilizar (atingir um *ponto-fixo*)

Calcular FIRST e NULLABLE

Método iterativo:

1. Inicialmente $\text{NULLABLE}(X) := \text{False}$ e $\text{FIRST}(X) := \emptyset$ para todos os símbolos não-terminais
2. Calcular novos valores para os lados direitos das produções usando as equações do *slide* anterior
3. Repetir até estabilizar (atingir um *ponto-fixo*)

- Podemos calcular primeiro apenas NULLABLE e depois FIRST
- Este processo termina sempre porque as equações definem uma transformação *monónota* sobre uma *ordem parcial completa finita* — conceitos estudados em *Fundamentos de Linguagens*

Exemplo: gramática de expressões

$$
\begin{array}{lll}
E \to E \ + \ T & \quad T \to T \ * \ F & \quad F \to \text{num} \\
E \to T & \quad T \to F & \quad F \to (\ E\ )
\end{array}
$$

$$
\begin{aligned}
&\text{NULLABLE}(E) = (\text{NULLABLE}(E) \wedge \text{NULLABLE}(+) \wedge \text{NULLABLE}(T)) \vee \text{NULLABLE}(T) \\
&\text{NULLABLE}(T) = (\text{NULLABLE}(T) \wedge \text{NULLABLE}(*) \wedge \text{NULLABLE}(F)) \vee \text{NULLABLE}(F) \\
&\text{NULLABLE}(F) = \text{NULLABLE}(\text{num}) \vee \text{NULLABLE}((E)) = \text{False}
\end{aligned}
$$

	iterações	
não-terminais	0	1
NULLABLE(E)	False	False
NULLABLE(T)	False	False
NULLABLE(F)	False	False

Estabiliza logo na iteração 1.

Exemplo: gramática de expressões (cont.)

$$
\begin{array}{lll}
E \to E \; + \; T & T \to T \; * \; F & F \to \mathrm{num} \\
E \to T & T \to F & F \to (\; E \;)
\end{array}
$$

$$
\begin{aligned}
\mathrm{FIRST}(E) &= \mathrm{FIRST}(E + T) \cup \mathrm{FIRST}(T) = \mathrm{FIRST}(E) \cup \mathrm{FIRST}(T) \\
\mathrm{FIRST}(T) &= \mathrm{FIRST}(T * F) \cup \mathrm{FIRST}(F) = \mathrm{FIRST}(T) \cup \mathrm{FIRST}(F) \\
\mathrm{FIRST}(F) &= \mathrm{FIRST}(\mathrm{num}) \cup \mathrm{FIRST}((E)) = \{\mathrm{num}, (\}
\end{aligned}
$$

	iterações				
não-terminais	0	1	2	3	4
FIRST(E)	$\emptyset$	$\emptyset$	$\emptyset$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$
FIRST(T)	$\emptyset$	$\emptyset$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$
FIRST(F)	$\emptyset$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$	$\{\mathrm{num}, (\}$

Estabiliza na iteração 4.

Soluções:

$$\begin{aligned}
\text{FIRST}(E) &= \{\text{num}, (\} \\
\text{FIRST}(T) &= \{\text{num}, (\} \\
\text{FIRST}(F) &= \{\text{num}, (\}
\end{aligned}$$

Logo: a gramática de expressões não é $LL(1)$ porque (por exemplo) os conjuntos FIRST dos lados direitos das produções

$$\begin{aligned}
E &\to E + T \\
E &\to T
\end{aligned}$$

não são disjuntos — de facto são exatamente iguais a $\{\text{num}, (\}$.

Escrever equações e calcular NULLABLE e FIRST para a seguinte gramática.

$$\begin{aligned} S &\to AB \\ A &\to aAb \mid \varepsilon \\ B &\to bB \mid \varepsilon \end{aligned}$$

Escrever equações e calcular NULLABLE e FIRST para a seguinte gramática.

$$\begin{aligned} S &\to AB \\ A &\to aAb \mid \varepsilon \\ B &\to bB \mid \varepsilon \end{aligned}$$

Soluções:

$$\text{NULLABLE}(S) = \text{NULLABLE}(A) = \text{NULLABLE}(B) = \mathit{True}$$

$$\begin{aligned} \text{FIRST}(S) &= \{a, b\} \\ \text{FIRST}(A) &= \{a\} \\ \text{FIRST}(B) &= \{b\} \end{aligned}$$

- O conjunto FIRST não é suficiente para caracterizar as gramáticas $LL(1)$
- Se tivermos produções da forma $X \to \gamma$ com NULLABLE(γ) então necessitamos de saber o que pode ocorrer *depois* de X (FIRST(γ) não dá essa informação)
- Conjunto FOLLOW(X): os terminais que podem ocorrem depois X numa derivação a partir de S

$$\text{FOLLOW}(X) = \{c \in \Sigma \ : \ \text{existem } \alpha, \beta \text{ tais que } S \Rightarrow^* \alpha X c \beta\}$$

Equações para FOLLOW

Acresentamos um novo símbolo \$ e uma produção para o fim da entrada:

$$S' \to S\$$$

Para cada não-terminal X, para cada produção da forma $Y \to \alpha X \beta$:

- $\text{FOLLOW}(X) \supseteq \text{FIRST}(\beta)$
- Se $\text{NULLABLE}(\beta)$ então $\text{FOLLOW}(X) \supseteq \text{FOLLOW}(Y)$

$$
\begin{aligned}
S' &\rightarrow S\$ \\
S &\rightarrow AB \\
A &\rightarrow aAb \mid \varepsilon \\
B &\rightarrow bB \mid \varepsilon
\end{aligned}
$$

$S' \rightarrow S\$$	$\text{FOLLOW}(S) \supseteq \{\$\}$	$\text{FIRST}(\$) = \{\$\}$
$S \rightarrow AB$	$\text{FOLLOW}(A) \supseteq \{b\}$	$\text{FIRST}(B) = \{b\}$
	$\text{FOLLOW}(A) \supseteq \text{FOLLOW}(S)$	porque $\text{NULLABLE}(B)$
	$\text{FOLLOW}(B) \supseteq \text{FOLLOW}(S)$	
$A \rightarrow aAb$	$\text{FOLLOW}(A) \supseteq \{b\}$	$\text{FIRST}(b) = \{b\}$
$B \rightarrow bB$	$\text{FOLLOW}(B) \supseteq \text{FOLLOW}(B)$	
	($A \rightarrow \varepsilon$ e $B \rightarrow \varepsilon$ não contribuem)	

Obtemos as equações:

$$
\begin{aligned}
\mathsf{FOLLOW}(S) &\supseteq \{\$\}\\
\mathsf{FOLLOW}(A) &\supseteq \{b\}\\
\mathsf{FOLLOW}(A) &\supseteq \mathsf{FOLLOW}(S)\\
\mathsf{FOLLOW}(B) &\supseteq \mathsf{FOLLOW}(S)\\
\mathsf{FOLLOW}(B) &\supseteq \mathsf{FOLLOW}(B)
\end{aligned}
$$

Resolvemos iterativamente, começando com $\emptyset$ para todos os não-terminais.

	iterações			
não-terminais	0	1	2	3
FOLLOW(S)	$\emptyset$	$\{\$\}$	$\{\$\}$	$\{\$\}$
FOLLOW(A)	$\emptyset$	$\{b\}$	$\{b, \$\}$	$\{b, \$\}$
FOLLOW(B)	$\emptyset$	$\emptyset$	$\{\$\}$	$\{\$\}$

Usando NULLABLE, FIRST e FOLLOW podemos construir uma tabela de *parsing* preditivo para uma gramática:

- as colunas correspondem aos *símbolos terminais*
- as linhas correspondem aos *símbolos não-terminais*
- colocamos cada produção $X \to \gamma$:
  - na linha X e nas colunas t para cada $t \in \text{FIRST}(\gamma)$;
  - se NULLABLE(γ), também acrescentamos na linha X e colunas t para cada $t \in \text{FOLLOW}(X)$.

A gramática é $LL(1)$ se e só se cada entrada tiver no máximo uma produção.

Exemplo:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela de *parsing* (ver NULLABLE, FIRST, FOLLOW nos slides anteriores):

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

Cada entrada tem no máximo uma produção, logo esta gramática é $LL(1)$.

Parsing preditivo usando a tabela

- *Parsing* preditivo por descida recursiva usa a *pilha de execução* da linguagem de programação (e.g. Haskell ou C)
- Podemos implementar *parsing* preditivo sem recursão usando a tabela de produções e uma pilha explícita

Pseudo-código para o algoritmo de *parsing*:

```
stack := empty; push (S', stack);
while (stack not empty) do
  if top(stack) is a terminal then
    /* consumir input */
    consume(top(stack)); pop(stack);
  else if(table[top(stack),next] is empty) then
     report_error();
  else
    /* usar uma producão */
    symbols := right_hand_side(table[top(stack),next]);
    pop(stack);
    pushList(symbols, stack);
```

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\rightarrow S\$ \\
S &\rightarrow AB \\
A &\rightarrow aAb \mid \varepsilon \\
B &\rightarrow bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \rightarrow S\$$	$S' \rightarrow S\$$	$S' \rightarrow S\$$
S	$S \rightarrow AB$	$S \rightarrow AB$	$S \rightarrow AB$
A	$A \rightarrow aAb$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
B		$B \rightarrow bB$	$B \rightarrow \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \rightarrow S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	

Exemplo: *parsing* de $aabbb$

[dcc] FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	

Gramática:

$$
\begin{aligned}
S' &\rightarrow S\$ \\
S &\rightarrow AB \\
A &\rightarrow aAb \mid \varepsilon \\
B &\rightarrow bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \rightarrow S\$$	$S' \rightarrow S\$$	$S' \rightarrow S\$$
S	$S \rightarrow AB$	$S \rightarrow AB$	$S \rightarrow AB$
A	$A \rightarrow aAb$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
B		$B \rightarrow bB$	$B \rightarrow \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \rightarrow S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \rightarrow AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \rightarrow aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \rightarrow aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	

Exemplo: *parsing* de $aabbb$

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	consumir b
$\underline{B}\$$	$\underline{b}\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	**input**	**ação**
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	consumir b
$\underline{B}\$$	$\underline{b}\$$	$B \to bB$
$\underline{b}B\$$	$\underline{b}\$$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	consumir b
$\underline{B}\$$	$\underline{b}\$$	$B \to bB$
$\underline{b}B\$$	$\underline{b}\$$	consumir b
$\underline{B}\$$	$\underline{\$}$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	consumir b
$\underline{B}\$$	$\underline{b}\$$	$B \to bB$
$\underline{b}B\$$	$\underline{b}\$$	consumir b
$\underline{B}\$$	$\underline{\$}$	$B \to \varepsilon$
$\underline{\$}$	$\underline{\$}$	

Exemplo: *parsing* de *aabbb*

Gramática:

$$
\begin{aligned}
S' &\to S\$ \\
S &\to AB \\
A &\to aAb \mid \varepsilon \\
B &\to bB \mid \varepsilon
\end{aligned}
$$

Tabela:

	a	b	$\$$
S'	$S' \to S\$$	$S' \to S\$$	$S' \to S\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

stack	input	ação
$\underline{S'}$	$\underline{a}abbb\$$	$S' \to S\$$
$\underline{S}\$$	$\underline{a}abbb\$$	$S \to AB$
$\underline{A}B\$$	$\underline{a}abbb\$$	$A \to aAb$
$\underline{a}AbB\$$	$\underline{a}abbb\$$	consumir a
$\underline{A}bB\$$	$\underline{a}bbb\$$	$A \to aAb$
$\underline{a}AbbB\$$	$\underline{a}bbb\$$	consumir a
$\underline{A}bbB\$$	$\underline{b}bb\$$	$A \to \varepsilon$
$\underline{b}bB\$$	$\underline{b}bb\$$	consumir b
$\underline{b}B\$$	$\underline{b}b\$$	consumir b
$\underline{B}\$$	$\underline{b}\$$	$B \to bB$
$\underline{b}B\$$	$\underline{b}\$$	consumir b
$\underline{B}\$$	$\underline{\$}$	$B \to \varepsilon$
$\underline{\$}$	$\underline{\$}$	consumir $\$$
ε	ε	**aceitação**

Análise sintática *top-down*:

- *Parsing* preditivo usando descida recursiva
- Implementações recursivas em Haskell e C
- A definição de classe $LL(1)$ de gramáticas preditivas
- A definição da tabela de *parsing* preditivo
- Implementação de *parsing* preditivo usando a tabela e uma pilha auxiliar