

# Um compilador simples de uma passagem

Visão geral

**Prof. Edson Alves**

Faculdade UnB Gama

# Sumário

1. **Visão geral**
2. **Definição da sintaxe**
3. **Tradução dirigida pela sintaxe**
4. **Análise gramatical**

# Caracterização de uma linguagem de programação

- ▶ Uma linguagem de programação pode ser caracterizada por sua sintaxe (aparência e forma de seus elementos) e por sua semântica (o significado destes elementos)

# Caracterização de uma linguagem de programação

- ▶ Uma linguagem de programação pode ser caracterizada por sua sintaxe (aparência e forma de seus elementos) e por sua semântica (o significado destes elementos)
- ▶ Uma forma de especificar a sintaxe de uma linguagem é a gramática livre de contexto (BNF – Forma de Backus-Naur)

# Caracterização de uma linguagem de programação

- ▶ Uma linguagem de programação pode ser caracterizada por sua sintaxe (aparência e forma de seus elementos) e por sua semântica (o significado destes elementos)
- ▶ Uma forma de especificar a sintaxe de uma linguagem é a gramática livre de contexto (BNF – Forma de Backus-Naur)
- ▶ Além de especificar a semântica, a gramática livre de contexto auxilia a tradução de programa, por meio da técnica denominada tradução dirigida pela sintaxe

# Caracterização de uma linguagem de programação

- ▶ Uma linguagem de programação pode ser caracterizada por sua sintaxe (aparência e forma de seus elementos) e por sua semântica (o significado destes elementos)
- ▶ Uma forma de especificar a sintaxe de uma linguagem é a gramática livre de contexto (BNF – Forma de Backus-Naur)
- ▶ Além de especificar a semântica, a gramática livre de contexto auxilia a tradução de programa, por meio da técnica denominada tradução dirigida pela sintaxe
- ▶ A especificação da semântica é mais complicada, de modo que em muitos casos é feita por meio de exemplos e descrições informais

## Compilador de expressões infixas para posfixas

- ▶ A tradução dirigida pela sintaxe será ilustrada por meio do desenvolvimento de um compilador simples de uma passagem que traduz expressões na forma infixa para a forma posfixa

## Compilador de expressões infixas para posfixas

- ▶ A tradução dirigida pela sintaxe será ilustrada por meio do desenvolvimento de um compilador simples de uma passagem que traduz expressões na forma infixa para a forma posfixa
- ▶ Por exemplo, a expressão  $1-2+3$ , que está na forma infixa (o operador está posicionado entre os operandos), corresponde a expressão posfixa  $12-3+$  (o operador sucede os dois operandos, assuma que cada operando consiste em um único dígito)



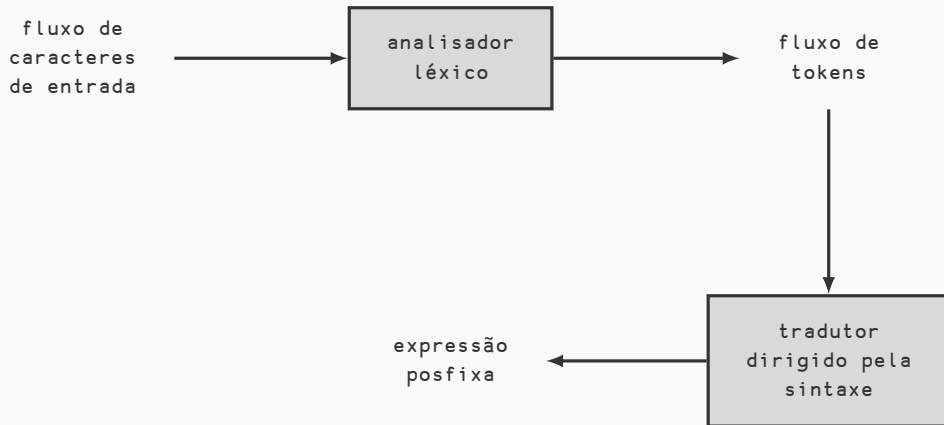
## Compilador de expressões infixas para posfixas

- ▶ A tradução dirigida pela sintaxe será ilustrada por meio do desenvolvimento de um compilador simples de uma passagem que traduz expressões na forma infixa para a forma posfixa
- ▶ Por exemplo, a expressão  $1-2+3$ , que está na forma infixa (o operador está posicionado entre os operandos), corresponde a expressão posfixa  $12-3+$  (o operador sucede os dois operandos, assuma que cada operando consiste em um único dígito)
- ▶ A forma posfixa pode ser convertida diretamente para um programa que executa a expressão usando uma pilha

## Compilador de expressões infixas para posfixas

- ▶ A tradução dirigida pela sintaxe será ilustrada por meio do desenvolvimento de um compilador simples de uma passagem que traduz expressões na forma infixa para a forma posfixa
- ▶ Por exemplo, a expressão  $1-2+3$ , que está na forma infixa (o operador está posicionado entre os operandos), corresponde a expressão posfixa  $12-3+$  (o operador sucede os dois operandos, assuma que cada operando consiste em um único dígito)
- ▶ A forma posfixa pode ser convertida diretamente para um programa que executa a expressão usando uma pilha
- ▶ O analisador léxico gerará um fluxo de tokens que alimentarão o tradutor dirigido pela sintaxe (o qual combinará o analisador sintático com o gerador de código intermediário), que por sua vez gerará a representação posfixa

# Estrutura da interface de vanguarda do compilador



# Definições

- ▶ Uma gramática deve descrever a estrutura hierárquica de seus elementos

# Definições

- ▶ Uma gramática deve descrever a estrutura hierárquica de seus elementos
- ▶ Por exemplo, o comando `if-else` da linguagem C, possui a forma

`if` (expressão) comando `else` comando

a qual pode ser expressão como

$$cmd \rightarrow \text{if} (expr) \text{ cmd } \text{else} \text{ cmd}$$

# Definições

- ▶ Uma gramática deve descrever a estrutura hierárquica de seus elementos
- ▶ Por exemplo, o comando `if-else` da linguagem C, possui a forma

`if` (expressão) comando `else` comando

a qual pode ser expressão como

$$cmd \rightarrow \text{if} (expr) \text{ cmd } \text{else} \text{ cmd}$$

- ▶ A expressão acima é uma regra de produção, onde a seta significa “pode ter a forma”

# Definições

- ▶ Uma gramática deve descrever a estrutura hierárquica de seus elementos
- ▶ Por exemplo, o comando `if-else` da linguagem C, possui a forma

`if` (expressão) comando `else` comando

a qual pode ser expressão como

$$cmd \rightarrow \text{if } (expr) \text{ } cmd \text{ else } cmd$$

- ▶ A expressão acima é uma regra de produção, onde a seta significa “pode ter a forma”
- ▶ Os elementos léxicos da produção (palavras-chaves, parêntesis) são chamados tokens ou terminais

# Definições

- ▶ Uma gramática deve descrever a estrutura hierárquica de seus elementos
- ▶ Por exemplo, o comando `if-else` da linguagem C, possui a forma

`if` (expressão) comando `else` comando

a qual pode ser expressão como

$$cmd \rightarrow \text{if } (expr) \text{ } cmd \text{ else } cmd$$

- ▶ A expressão acima é uma regra de produção, onde a seta significa “pode ter a forma”
- ▶ Os elementos léxicos da produção (palavras-chaves, parêntesis) são chamados tokens ou terminais
- ▶ Variáveis como *expr* e *cmd* representam sequências de tokens e são denominadas não-terminais



# Componentes da linguagem livre de contexto

1. Um conjunto de tokens, denominados símbolos terminais

# Componentes da linguagem livre de contexto

1. Um conjunto de tokens, denominados símbolos terminais
2. Um conjunto de não-terminais

# Componentes da linguagem livre de contexto

1. Um conjunto de tokens, denominados símbolos terminais
2. Um conjunto de não-terminais
3. Um conjunto de produções. Cada produção é definida por um não-terminal (lado esquerdo), seguido de uma seta, sucedida por uma sequência de tokens e/ou não-terminais (lado direito)

# Componentes da linguagem livre de contexto

1. Um conjunto de tokens, denominados símbolos terminais
2. Um conjunto de não-terminais
3. Um conjunto de produções. Cada produção é definida por um não-terminal (lado esquerdo), seguido de uma seta, sucedida por uma sequência de tokens e/ou não-terminais (lado direito)
4. Designação de um dos não-terminais como símbolo de partida

# Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções

# Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções
- ▶ O símbolo de partida é definido como o não-terminal da primeira produção listada

# Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções
- ▶ O símbolo de partida é definido como o não-terminal da primeira produção listada
- ▶ Dígitos, símbolos e palavras em negrito são terminais

# Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções
- ▶ O símbolo de partida é definido como o não-terminal da primeira produção listada
- ▶ Dígitos, símbolos e palavras em negrito são terminais
- ▶ Não-terminais são grafados em itálico



# Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções
- ▶ O símbolo de partida é definido como o não-terminal da primeira produção listada
- ▶ Dígitos, símbolos e palavras em negrito são terminais
- ▶ Não-terminais são grafados em itálico
- ▶ Os demais símbolos são tokens

## Convenções de notação da gramática livre de contexto

- ▶ A gramática é especificada por uma lista de produções
- ▶ O símbolo de partida é definido como o não-terminal da primeira produção listada
- ▶ Dígitos, símbolos e palavras em negrito são terminais
- ▶ Não-terminais são grafados em itálico
- ▶ Os demais símbolos são tokens
- ▶ Produções distintas de um mesmo não-terminal podem ser agrupadas por meio do caractere '|', que significa, neste contexto, “ou”

## Exemplo de sintaxe para expressões infixas com adição e subtração

- Considere a seguinte gramática para expressões compostas por dígitos decimais e as operações de adição e subtração, em forma infixa:

$$\begin{aligned} \textit{expr} &\rightarrow \textit{expr} + \textit{digito} \mid \textit{expr} - \textit{digito} \mid \textit{digito} \\ \textit{digito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

## Exemplo de sintaxe para expressões infixas com adição e subtração

- Considere a seguinte gramática para expressões compostas por dígitos decimais e as operações de adição e subtração, em forma infixa:

$$\begin{aligned} \textit{expr} &\rightarrow \textit{expr} + \textit{digito} \mid \textit{expr} - \textit{digito} \mid \textit{digito} \\ \textit{digito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

- Os tokens desta gramática são os dez dígitos decimais e os caracteres '+' e '-'

## Exemplo de sintaxe para expressões infixas com adição e subtração

- ▶ Considere a seguinte gramática para expressões compostas por dígitos decimais e as operações de adição e subtração, em forma infixa:

$$\begin{aligned} \textit{expr} &\rightarrow \textit{expr} + \textit{digito} \mid \textit{expr} - \textit{digito} \mid \textit{digito} \\ \textit{digito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

- ▶ Os tokens desta gramática são os dez dígitos decimais e os caracteres '+' e '-'
- ▶ Os não-terminais são *expr* e *digito*

## Exemplo de sintaxe para expressões infixas com adição e subtração

- ▶ Considere a seguinte gramática para expressões compostas por dígitos decimais e as operações de adição e subtração, em forma infixa:

$$\begin{aligned}expr &\rightarrow expr + digito \mid expr - digito \mid digito \\ digito &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

- ▶ Os tokens desta gramática são os dez dígitos decimais e os caracteres '+' e '-'
- ▶ Os não-terminais são *expr* e *digito*
- ▶ O símbolo de partida é o não-terminal *expr*

# Cadeias de tokens

- ▶ Uma cadeia de tokens é uma sequência de zero ou mais tokens

# Cadeias de tokens

- ▶ Uma cadeia de tokens é uma sequência de zero ou mais tokens
- ▶ Uma cadeia contendo zero tokens, grafada como  $\epsilon$ , é denominada cadeia vazia



# Cadeias de tokens

- ▶ Uma cadeia de tokens é uma sequência de zero ou mais tokens
- ▶ Uma cadeia contendo zero tokens, grafada como  $\epsilon$ , é denominada cadeia vazia
- ▶ Uma gramática deriva cadeias de tokens começando pelo símbolo de partida, substituindo repetidamente um não-terminal pelo lado direito de uma produção deste não-terminal

## Cadeias de tokens

- ▶ Uma cadeia de tokens é uma sequência de zero ou mais tokens
- ▶ Uma cadeia contendo zero tokens, grafada como  $\epsilon$ , é denominada cadeia vazia
- ▶ Uma gramática deriva cadeias de tokens começando pelo símbolo de partida, substituindo repetidamente um não-terminal pelo lado direito de uma produção deste não-terminal
- ▶ O conjunto de todas as cadeias de tokens possíveis gerados desta maneira formam a linguagem definida pela gramática

## Exemplo de construção da expressão $1-2+3$ por meio da gramática

1. 1 é *expr*, pois 1 é *digito* (terceira alternativa para a produção de *expr*)

## Exemplo de construção da expressão $1-2+3$ por meio da gramática

1. 1 é *expr*, pois 1 é *digito* (terceira alternativa para a produção de *expr*)
2. Pela segunda alternativa de produção de *expr*,  $1-2$  é também *expr*, pois 1 é *expr* e 2 é *digito*

## Exemplo de construção da expressão $1-2+3$ por meio da gramática

1. 1 é *expr*, pois 1 é *digito* (terceira alternativa para a produção de *expr*)
2. Pela segunda alternativa de produção de *expr*,  $1-2$  é também *expr*, pois 1 é *expr* e 2 é *digito*
3. Por fim, pela primeira alternativa de produção de *expr*,  $1-2+3$  é *expr*, pois  $1-2$  é *expr* e 3 é *digito*

# Árvore gramatical

Dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

# Árvore gramatical

Dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

1. A raiz é rotulada pelo símbolo de partida

# Árvore gramatical

Dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

1. A raiz é rotulada pelo símbolo de partida
2. Cada folha é rotulada por um token ou por  $\epsilon$



# Árvore gramatical

Dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

1. A raiz é rotulada pelo símbolo de partida
2. Cada folha é rotulada por um token ou por  $\epsilon$
3. Cada nó interior é rotulado por um não-terminal

# Árvore gramatical

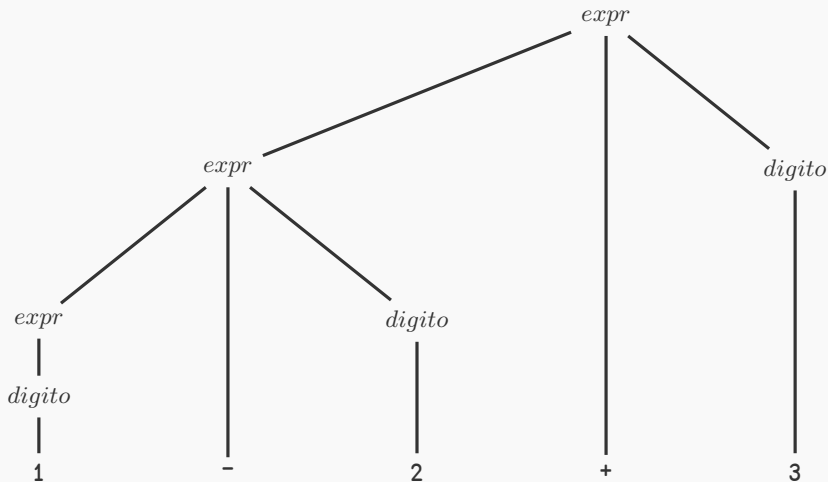
Dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

1. A raiz é rotulada pelo símbolo de partida
2. Cada folha é rotulada por um token ou por  $\epsilon$
3. Cada nó interior é rotulado por um não-terminal
4. Se  $A$  é um não-terminal que rotula um nó interior e  $X_1, X_2, \dots, X_N$  são os rótulos de seus filhos (da esquerda para a direita), então

$$A \rightarrow X_1 X_2 \dots X_N$$

é uma produção

# Visualização da árvore gramatical da expressão $1-2+3$



## Características da árvore gramatical

- ▶ As folhas da árvore gramatical, quando lidas da esquerda para a direita, formam o produto da árvore, que é a cadeia gerada ou derivada a partir da raiz não-terminal

## Características da árvore gramatical

- ▶ As folhas da árvore gramatical, quando lidas da esquerda para a direita, formam o produto da árvore, que é a cadeia gerada ou derivada a partir da raiz não-terminal
- ▶ O processo de encontrar uma árvore gramatical para uma dada cadeia de tokens é chamado de análise gramatical ou análise sintática daquela cadeia

## Características da árvore gramatical

- ▶ As folhas da árvore gramatical, quando lidas da esquerda para a direita, formam o produto da árvore, que é a cadeia gerada ou derivada a partir da raiz não-terminal
- ▶ O processo de encontrar uma árvore gramatical para uma dada cadeia de tokens é chamado de análise gramatical ou análise sintática daquela cadeia
- ▶ Uma gramática que permite a construção de duas ou mais árvores gramaticais distintas para uma mesma cadeia de tokens é denominada gramática ambígua

## Características da árvore gramatical

- ▶ As folhas da árvore gramatical, quando lidas da esquerda para a direita, formam o produto da árvore, que é a cadeia gerada ou derivada a partir da raiz não-terminal
- ▶ O processo de encontrar uma árvore gramatical para uma dada cadeia de tokens é chamado de análise gramatical ou análise sintática daquela cadeia
- ▶ Uma gramática que permite a construção de duas ou mais árvores gramaticais distintas para uma mesma cadeia de tokens é denominada gramática ambígua
- ▶ A gramática apresentada não é ambígua

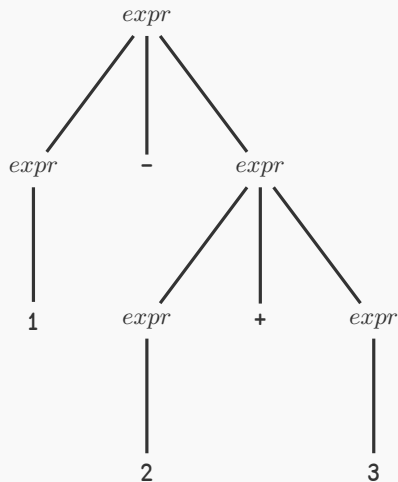
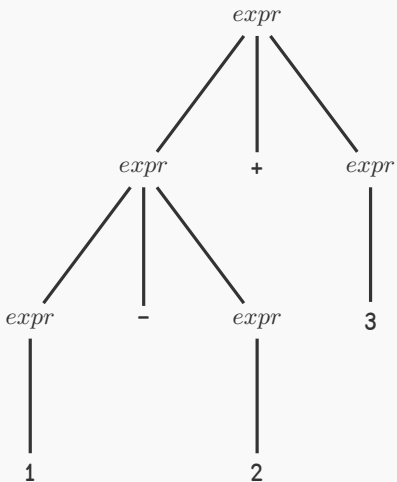
## Características da árvore gramatical

- ▶ As folhas da árvore gramatical, quando lidas da esquerda para a direita, formam o produto da árvore, que é a cadeia gerada ou derivada a partir da raiz não-terminal
- ▶ O processo de encontrar uma árvore gramatical para uma dada cadeia de tokens é chamado de análise gramatical ou análise sintática daquela cadeia
- ▶ Uma gramática que permite a construção de duas ou mais árvores gramaticais distintas para uma mesma cadeia de tokens é denominada gramática ambígua
- ▶ A gramática apresentada não é ambígua
- ▶ Contudo, se removida a distinção entre *expr* e *digito*, a gramática passaria a ser ambígua:

$$expr \rightarrow expr + expr \mid expr - expr \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$



## Exemplo de gramática ambígua



## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando

## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando
- ▶ Uma operação  $\odot$  é associativa à esquerda se  $a \odot b \odot c = (a \odot b) \odot c$

## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando
- ▶ Uma operação  $\odot$  é associativa à esquerda se  $a \odot b \odot c = (a \odot b) \odot c$
- ▶ Na maioria das linguagens de programação, os operadores aritméticos (+, -, \* e /) são associativos à esquerda

## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando
- ▶ Uma operação  $\odot$  é associativa à esquerda se  $a \odot b \odot c = (a \odot b) \odot c$
- ▶ Na maioria das linguagens de programação, os operadores aritméticos (+, -, \* e /) são associativos à esquerda
- ▶ Uma operação  $\oslash$  é associativa à direita se  $a \oslash b \oslash c = a \oslash (b \oslash c)$

## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando
- ▶ Uma operação  $\odot$  é associativa à esquerda se  $a \odot b \odot c = (a \odot b) \odot c$
- ▶ Na maioria das linguagens de programação, os operadores aritméticos (+, -, \* e /) são associativos à esquerda
- ▶ Uma operação  $\oslash$  é associativa à direita se  $a \oslash b \oslash c = a \oslash (b \oslash c)$
- ▶ Por exemplo, a atribuição (operador =) da linguagem C é associativa à direita: a expressão  $a = b = c$  equivale a expressão  $a = (b = c)$

## Associatividade de operadores

- ▶ Quando um operando está, simultaneamente, à esquerda e à direita de dois operadores (por exemplo, o dígito 2 na expressão  $1-2+3$ ), é preciso decidir qual destes operadores receberá o operando
- ▶ Uma operação  $\odot$  é associativa à esquerda se  $a \odot b \odot c = (a \odot b) \odot c$
- ▶ Na maioria das linguagens de programação, os operadores aritméticos (+, -, \* e /) são associativos à esquerda
- ▶ Uma operação  $\oslash$  é associativa à direita se  $a \oslash b \oslash c = a \oslash (b \oslash c)$
- ▶ Por exemplo, a atribuição (operador =) da linguagem C é associativa à direita: a expressão  $a = b = c$  equivale a expressão  $a = (b = c)$
- ▶ Uma gramática possível para esta atribuição seria:

$$\begin{aligned} \text{expr} &\rightarrow \text{var} = \text{expr} \mid \text{var} \\ \text{var} &\rightarrow a \mid b \mid \dots \mid z \end{aligned}$$

# Precedência de operadores

- ▶ Algumas expressões da aritmética contém ambiguidades que não podem ser resolvidas apenas por meio da associatividade



# Precedência de operadores

- ▶ Algumas expressões da aritmética contém ambiguidades que não podem ser resolvidas apenas por meio da associatividade
- ▶ Por exemplo, qual seria o resultado da expressão  $1 + 2 * 3$ ? 9 ou 7?

## Precedência de operadores

- ▶ Algumas expressões da aritmética contêm ambiguidades que não podem ser resolvidas apenas por meio da associatividade
- ▶ Por exemplo, qual seria o resultado da expressão  $1 + 2 * 3$ ? 9 ou 7?
- ▶ Dizemos que o operador  $\otimes$  tem maior precedência do que o operador  $\oplus$  se  $\otimes$  captura os operandos antes que  $\oplus$  o faça

## Precedência de operadores

- ▶ Algumas expressões da aritmética contém ambiguidades que não podem ser resolvidas apenas por meio da associatividade
- ▶ Por exemplo, qual seria o resultado da expressão  $1 + 2 * 3$ ? 9 ou 7?
- ▶ Dizemos que o operador  $\otimes$  tem maior precedência do que o operador  $\oplus$  se  $\otimes$  captura os operandos antes que  $\oplus$  o faça
- ▶ Na aritmética, a multiplicação e a divisão tem maior precedência do que a adição e a subtração

## Precedência de operadores

- ▶ Algumas expressões da aritmética contém ambiguidades que não podem ser resolvidas apenas por meio da associatividade
- ▶ Por exemplo, qual seria o resultado da expressão  $1 + 2 * 3$ ? 9 ou 7?
- ▶ Dizemos que o operador  $\otimes$  tem maior precedência do que o operador  $\oplus$  se  $\otimes$  captura os operandos antes que  $\oplus$  o faça
- ▶ Na aritmética, a multiplicação e a divisão tem maior precedência do que a adição e a subtração
- ▶ Se dois operadores tem mesma precedência, a associatividade determina a ordem que as operações serão realizadas

# Construção de gramáticas com precedência de operadores

É possível construir uma gramática com precedência de operadores a partir dos seguintes passos:

# Construção de gramáticas com precedência de operadores

É possível construir uma gramática com precedência de operadores a partir dos seguintes passos:

1. Construa uma tabela com a associatividade e a precedência dos operadores, em ordem crescente de precedência (operadores com mesma precedência aparecem na mesma linha)

# Construção de gramáticas com precedência de operadores

É possível construir uma gramática com precedência de operadores a partir dos seguintes passos:

1. Construa uma tabela com a associatividade e a precedência dos operadores, em ordem crescente de precedência (operadores com mesma precedência aparecem na mesma linha)

associatividade à esquerda	+	-
associatividade à esquerda	*	/

# Construção de gramáticas com precedência de operadores

É possível construir uma gramática com precedência de operadores a partir dos seguintes passos:

1. Construa uma tabela com a associatividade e a precedência dos operadores, em ordem crescente de precedência (operadores com mesma precedência aparecem na mesma linha)

associatividade à esquerda	+	-
associatividade à esquerda	*	/

2. Crie um não-terminal para cada nível (*expr* e *termo*) e um não-terminal extra para as unidades básicas da expressão (*fator*)



# Construção de gramáticas com precedência de operadores

É possível construir uma gramática com precedência de operadores a partir dos seguintes passos:

1. Construa uma tabela com a associatividade e a precedência dos operadores, em ordem crescente de precedência (operadores com mesma precedência aparecem na mesma linha)

associatividade à esquerda	+	-
associatividade à esquerda	*	/

2. Crie um não-terminal para cada nível (*expr* e *termo*) e um não-terminal extra para as unidades básicas da expressão (*fator*)

$$fator \rightarrow \text{dígito} \mid (expr)$$

## Construção de gramáticas com precedência de operadores

3. Defina as produções para o último terminal criado para os níveis a partir dos operadores com maior precedência

## Construção de gramáticas com precedência de operadores

3. Defina as produções para o último terminal criado para os níveis a partir dos operadores com maior precedência

$$\begin{array}{lcl} termo & \rightarrow & termo * fator \\ & | & termo / fator \\ & | & fator \end{array}$$

## Construção de gramáticas com precedência de operadores

3. Defina as produções para o último terminal criado para os níveis a partir dos operadores com maior precedência

$$\begin{array}{lcl} termo & \rightarrow & termo * fator \\ & | & termo / fator \\ & | & fator \end{array}$$

4. Faça o mesmo para os demais operadores, em ordem decrescente de precedência e crescente na lista de terminais criados para os níveis

## Construção de gramáticas com precedência de operadores

3. Defina as produções para o último terminal criado para os níveis a partir dos operadores com maior precedência

$$\begin{array}{lcl} termo & \rightarrow & termo * fator \\ & | & termo / fator \\ & | & fator \end{array}$$

4. Faça o mesmo para os demais operadores, em ordem decrescente de precedência e crescente na lista de terminais criados para os níveis

$$\begin{array}{lcl} expr & \rightarrow & expr + termo \\ & | & expr - termo \\ & | & termo \end{array}$$

## Construção de gramáticas com precedência de operadores

3. Defina as produções para o último terminal criado para os níveis a partir dos operadores com maior precedência

$$\begin{array}{lcl} \textit{termo} & \rightarrow & \textit{termo} * \textit{fator} \\ & | & \textit{termo} / \textit{fator} \\ & | & \textit{fator} \end{array}$$

4. Faça o mesmo para os demais operadores, em ordem decrescente de precedência e crescente na lista de terminais criados para os níveis

$$\begin{array}{lcl} \textit{expr} & \rightarrow & \textit{expr} + \textit{termo} \\ & | & \textit{expr} - \textit{termo} \\ & | & \textit{termo} \end{array}$$

A presença de parêntesis na definição de *fator* permite escrever expressões com níveis arbitrários de aninhamento, sendo que os parêntesis tem precedência sobre todos os operadores definidos.

# Notação posfixa

## Definição de notação posfixa

A notação posfixa para uma expressão  $E$  é definida da seguinte maneira:

1. Se  $E$  for uma variável ou uma constante, então a notação posfixa para  $E$  é o próprio  $E$
2. Se  $E$  é uma expressão da forma  $E_1 \text{ op } E_2$ , onde  $\text{op}$  é um operador binário, então a forma posfixa para  $E$  é  $E'_1 E'_2 \text{ op}$ , onde  $E'_1$  e  $E'_2$  são as notações posfixas de  $E_1$  e  $E_2$ , respectivamente
3. Se  $E$  é uma expressão da forma  $(E_1)$ , então a notação posfixa para  $E_1$  será a notação posfixa para  $E$

# Definições dirigidas pela sintaxe

- ▶ Uma definição dirigida pela sintaxe usa a gramática livre de contexto para especificar a estrutura sintática da entrada



## Definições dirigidas pela sintaxe

- ▶ Uma definição dirigida pela sintaxe usa a gramática livre de contexto para especificar a estrutura sintática da entrada
- ▶ Ela associa, a cada símbolo da gramática, um conjunto de atributos e, a cada produção, um conjunto de regras semânticas para computar os valores dos atributos associados aos símbolos presentes na produção

## Definições dirigidas pela sintaxe

- ▶ Uma definição dirigida pela sintaxe usa a gramática livre de contexto para especificar a estrutura sintática da entrada
- ▶ Ela associa, a cada símbolo da gramática, um conjunto de atributos e, a cada produção, um conjunto de regras semânticas para computar os valores dos atributos associados aos símbolos presentes na produção
- ▶ A gramática e o conjunto de regras semânticas constituem a definição dirigida pela sintaxe

## Definições dirigidas pela sintaxe

- ▶ Uma definição dirigida pela sintaxe usa a gramática livre de contexto para especificar a estrutura sintática da entrada
- ▶ Ela associa, a cada símbolo da gramática, um conjunto de atributos e, a cada produção, um conjunto de regras semânticas para computar os valores dos atributos associados aos símbolos presentes na produção
- ▶ A gramática e o conjunto de regras semânticas constituem a definição dirigida pela sintaxe
- ▶ Um atributo é dito sintetizado se seu valor depende apenas dos valores dos atributos dos nós filhos de seu nó na árvore gramatical

## Definições dirigidas pela sintaxe

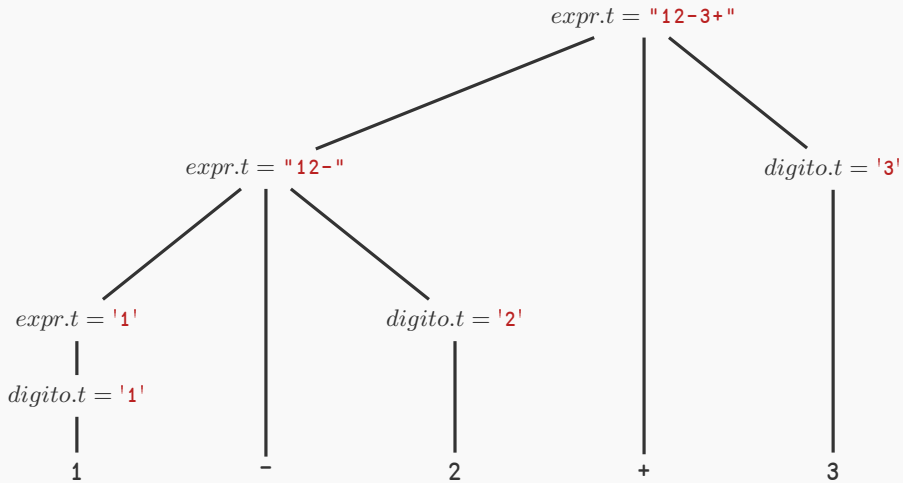
- ▶ Uma definição dirigida pela sintaxe usa a gramática livre de contexto para especificar a estrutura sintática da entrada
- ▶ Ela associa, a cada símbolo da gramática, um conjunto de atributos e, a cada produção, um conjunto de regras semânticas para computar os valores dos atributos associados aos símbolos presentes na produção
- ▶ A gramática e o conjunto de regras semânticas constituem a definição dirigida pela sintaxe
- ▶ Um atributo é dito sintetizado se seu valor depende apenas dos valores dos atributos dos nós filhos de seu nó na árvore gramatical
- ▶ Os atributos sintetizados podem ser computados por meio de uma travessia por profundidade

## Definição dirigida pela sintaxe para a tradução de notação infixa para posfixa

Produção	Regra semântica
$expr \rightarrow expr_1 + digito$	$expr.t := expr_1.t \parallel digito.t \parallel '+'$
$expr \rightarrow expr_1 - digito$	$expr.t := expr_1.t \parallel digito.t \parallel '-'$
$expr \rightarrow digito$	$expr.t := digito.t$
$digito \rightarrow 0$	$digito.t := '0'$
$digito \rightarrow 1$	$digito.t := '1'$
...	...
$digito \rightarrow 9$	$digito.t := '9'$

A notação  $X.t$  indica que  $t$  é um atributo de  $X$  e  $\parallel$  indica concatenação de caracteres.

## Valores dos atributos nos nós da árvore gramatical da expressão $1-2+3$



# Esquema de tradução

- ▶ Um esquema de tradução é uma gramática livre de contexto na qual fragmentos de programas, denominados ações semânticas, são inseridos nos lados direitos das produções

# Esquema de tradução

- ▶ Um esquema de tradução é uma gramática livre de contexto na qual fragmentos de programas, denominados ações semânticas, são inseridos nos lados direitos das produções
- ▶ Num esquema de tradução, a ordem de avaliação das ações semânticas é explicitamente mostrada



# Esquema de tradução

- ▶ Um esquema de tradução é uma gramática livre de contexto na qual fragmentos de programas, denominados ações semânticas, são inseridos nos lados direitos das produções
- ▶ Num esquema de tradução, a ordem de avaliação das ações semânticas é explicitamente mostrada
- ▶ A posição na qual uma ação semântica deve ser executada marcada no lado direito da produção, por meio de chaves

## Esquema de tradução

- ▶ Um esquema de tradução é uma gramática livre de contexto na qual fragmentos de programas, denominados ações semânticas, são inseridos nos lados direitos das produções
- ▶ Num esquema de tradução, a ordem de avaliação das ações semânticas é explicitamente mostrada
- ▶ A posição na qual uma ação semântica deve ser executada marcada no lado direito da produção, por meio de chaves
- ▶ Na árvore gramatical uma ação semântica é indicada por um filho extra, conectado por meio de uma linha pontilhada

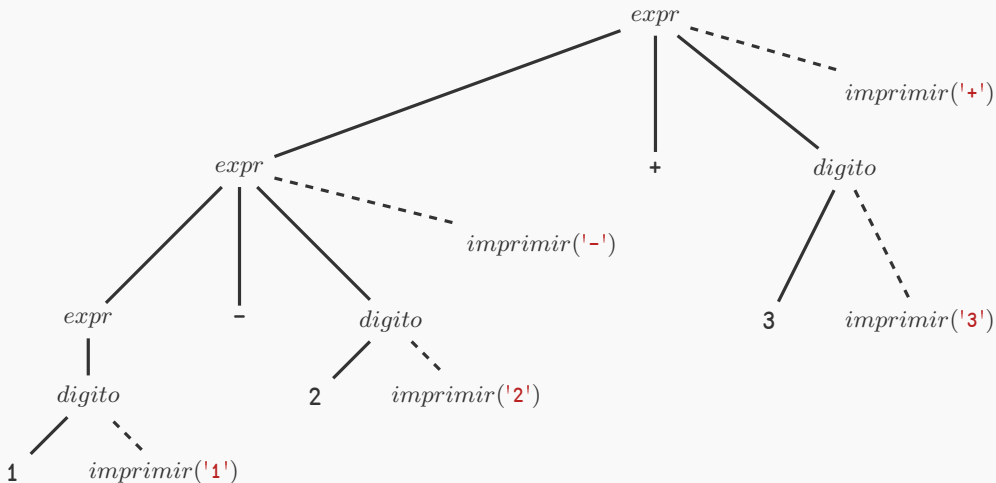
## Esquema de tradução

- ▶ Um esquema de tradução é uma gramática livre de contexto na qual fragmentos de programas, denominados ações semânticas, são inseridos nos lados direitos das produções
- ▶ Num esquema de tradução, a ordem de avaliação das ações semânticas é explicitamente mostrada
- ▶ A posição na qual uma ação semântica deve ser executada marcada no lado direito da produção, por meio de chaves
- ▶ Na árvore gramatical uma ação semântica é indicada por um filho extra, conectado por meio de uma linha pontilhada
- ▶ Nós rotulados por ações gramaticas não possui filhos

# Ações semânticas para a tradução de expressões para a notação posfixa

$$\begin{aligned} \text{expr} &\rightarrow \text{expr} + \text{digito} && \{\text{imprimir('+'})\} \\ \text{expr} &\rightarrow \text{expr} - \text{digito} && \{\text{imprimir('-')} }\} \\ \text{expr} &\rightarrow \text{digito} \\ \text{digito} &\rightarrow 0 && \{\text{imprimir('0')} \} \\ \text{digito} &\rightarrow 1 && \{\text{imprimir('1')} \} \\ &\dots \\ \text{digito} &\rightarrow 9 && \{\text{imprimir('9')} \} \end{aligned}$$

# Árvore gramatical com ações semânticas que traduz a expressão $1-2+3$



# Análise gramatical

- ▶ A análise gramatical é o processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática

# Análise gramatical

- ▶ A análise gramatical é o processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática
- ▶ O compilador deve ser capaz de construir uma árvore gramatical, mesmo que de forma implícita

# Análise gramatical

- ▶ A análise gramatical é o processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática
- ▶ O compilador deve ser capaz de construir uma árvore gramatical, mesmo que de forma implícita
- ▶ Um analisador gramatical pode ser construído para qualquer gramática



# Análise gramatical

- ▶ A análise gramatical é o processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática
- ▶ O compilador deve ser capaz de construir uma árvore gramatical, mesmo que de forma implícita
- ▶ Um analisador gramatical pode ser construído para qualquer gramática
- ▶ Para qualquer gramáticas livres de contexto existe um analisador gramatical que analisa  $N$  tokens com complexidade  $O(N^3)$

# Análise gramatical

- ▶ A análise gramatical é o processo de se determinar se uma cadeia de tokens pode ser gerada por uma gramática
- ▶ O compilador deve ser capaz de construir uma árvore gramatical, mesmo que de forma implícita
- ▶ Um analisador gramatical pode ser construído para qualquer gramática
- ▶ Para qualquer gramáticas livres de contexto existe um analisador gramatical que analisa  $N$  tokens com complexidade  $O(N^3)$
- ▶ Contudo, existem analisadores lineares para quase todas as gramáticas livres de contexto que surgem na prática

# Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais

## Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais
- ▶ Analisadores *top-down* a construção parte da raiz da árvore gramatical para suas folhas

## Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais
- ▶ Analisadores *top-down* a construção parte da raiz da árvore gramatical para suas folhas
- ▶ Analisadores *bottom-up* partem das folhas em direção à raiz

## Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais
- ▶ Analisadores *top-down* a construção parte da raiz da árvore gramatical para suas folhas
- ▶ Analisadores *bottom-up* partem das folhas em direção à raiz
- ▶ Os analisadores *top-down* são mais populares, pois é possível construir analisadores eficientes desta classe de forma manual

## Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais
- ▶ Analisadores *top-down* a construção parte da raiz da árvore gramatical para suas folhas
- ▶ Analisadores *bottom-up* partem das folhas em direção à raiz
- ▶ Os analisadores *top-down* são mais populares, pois é possível construir analisadores eficientes desta classe de forma manual
- ▶ Já os analisadores *bottom-up* podem manipular uma gama mais ampla de gramáticas

## Análise *top-down* e *bottom-up*

- ▶ Há duas classes principais de analisadores gramaticais
- ▶ Analisadores *top-down* a construção parte da raiz da árvore gramatical para suas folhas
- ▶ Analisadores *bottom-up* partem das folhas em direção à raiz
- ▶ Os analisadores *top-down* são mais populares, pois é possível construir analisadores eficientes desta classe de forma manual
- ▶ Já os analisadores *bottom-up* podem manipular uma gama mais ampla de gramáticas
- ▶ Geradores de analisadores gramaticais tendem a usar métodos *bottom-up*



# Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida

# Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:

## Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção

## Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção
  - (b) Encontre o próximo nó no qual uma subárvore deve ser construída

# Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção
  - (b) Encontre o próximo nó no qual uma subárvore deve ser construída

Observações:

## Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção
  - (b) Encontre o próximo nó no qual uma subárvore deve ser construída

Observações:

- (i) A depender da gramática, esta construção pode ser implementada com uma única passagem da entrada, da esquerda para a direita

## Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção
  - (b) Encontre o próximo nó no qual uma subárvore deve ser construída

Observações:

- (i) A depender da gramática, esta construção pode ser implementada com uma única passagem da entrada, da esquerda para a direita
- (ii) O token que está sendo observado é frequentemente denominado *lookahead*

## Construção *top-down* de uma árvore gramatical

1. Inicie na raiz, rotulada pelo não-terminal de partida
2. Repita os seguintes passos:
  - (a) Para o nó  $n$ , rotulado pelo não-terminal  $A$ , selecione uma das produções para  $A$  e construa os filhos de  $n$  com os símbolos do lado direito da produção
  - (b) Encontre o próximo nó no qual uma subárvore deve ser construída

Observações:

- (i) A depender da gramática, esta construção pode ser implementada com uma única passagem da entrada, da esquerda para a direita
- (ii) O token que está sendo observado é frequentemente denominado *lookahead*
- (iii) Inicialmente *lookahead* é o token mais à esquerda da entrada



## Exemplo: gramática para geração de subtipos em Pascal

$$\begin{array}{lcl} \textit{tipo} & \rightarrow & \textit{primitivo} \\ & | & \uparrow \textbf{id} \\ & | & \textbf{array} [ \textit{primitivo} ] \textbf{of } \textit{tipo} \end{array}$$
$$\begin{array}{lcl} \textit{primitivo} & \rightarrow & \textbf{integer} \\ & | & \textbf{char} \\ & | & \textbf{num} \textbf{.. num} \end{array}$$

## Exemplo: gramática para geração de subtipos em Pascal

$$\begin{array}{lcl} \textit{tipo} & \rightarrow & \textit{primitivo} \\ & | & \uparrow \textbf{id} \\ & | & \textbf{array} [ \textit{primitivo} ] \textbf{of } \textit{tipo} \end{array}$$
$$\begin{array}{lcl} \textit{primitivo} & \rightarrow & \textbf{integer} \\ & | & \textbf{char} \\ & | & \textbf{num} \textbf{.. num} \end{array}$$

Observação: os dois pontos ('..') formam um único token.

## Exemplo de construção *top-down* da árvore gramatical

Considere a expressão `array [ num .. num ] of integer`, gerada a partir da gramática de subtipos em Pascal.

## Exemplo de construção *top-down* da árvore gramatical

Considere a expressão `array [ num .. num ] of integer`, gerada a partir da gramática de subtipos em Pascal.

(a) A construção inicial na raiz da árvore. O rótulo da raiz é o não-terminal de partida

## Exemplo de construção *top-down* da árvore gramatical

Considere a expressão **array** [ num .. num ] **of integer**, gerada a partir da gramática de subtipos em Pascal.

- (a) A construção inicial na raiz da árvore. O rótulo da raiz é o não-terminal de partida  
*tipo*

## Exemplo de construção *top-down* da árvore gramatical

Considere a expressão **array** [ num .. num ] **of integer**, gerada a partir da gramática de subtipos em Pascal.

- (a) A construção inicial na raiz da árvore. O rótulo da raiz é o não-terminal de partida *tipo*
- (b) A única produção de *tipo* que inicia com o *lookahead* (neste momento, **array**) é a terceira. Esta produção será usada para a criação dos filhos do nó raiz.

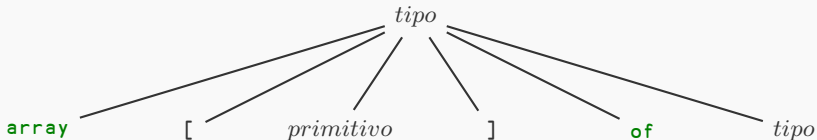
## Exemplo de construção *top-down* da árvore gramatical

Considere a expressão **array** [ num .. num ] **of** **integer**, gerada a partir da gramática de subtipos em Pascal.

- (a) A construção inicial na raiz da árvore. O rótulo da raiz é o não-terminal de partida

*tipo*

- (b) A única produção de *tipo* que inicia com o *lookahead* (neste momento, **array**) é a terceira. Esta produção será usada para a criação dos filhos do nó raiz.



## Exemplo de construção *top-down* da árvore gramatical

- (c) O filho mais à esquerda tem como rótulo **array**. Como este rótulo coincide com *lookahead*, a construção prossegue para o próximo filho



## Exemplo de construção *top-down* da árvore gramatical

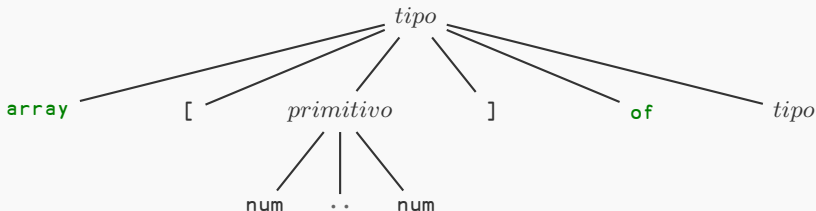
- (c) O filho mais à esquerda tem como rótulo **array**. Como este rótulo coincide com *lookahead*, a construção prossegue para o próximo filho
- (d) *Lookahead* é atualizado para `[` e confrontado com o segundo filho à esquerda da raiz. Como há nova coincidência entre o rótulo e *lookahead*, a construção prossegue

## Exemplo de construção *top-down* da árvore gramatical

- (c) O filho mais à esquerda tem como rótulo **array**. Como este rótulo coincide com *lookahead*, a construção prossegue para o próximo filho
- (d) *Lookahead* é atualizado para `[` e confrontado com o segundo filho à esquerda da raiz. Como há nova coincidência entre o rótulo e *lookahead*, a construção prossegue
- (e) O nó seguinte contém o não-terminal *primitivo* e *lookahead* contém o token `num`. Assim a terceira produção de *primitivo* é utilizada para gerar os novos filhos

## Exemplo de construção *top-down* da árvore gramatical

- (c) O filho mais à esquerda tem como rótulo **array**. Como este rótulo coincide com *lookahead*, a construção prossegue para o próximo filho
- (d) *Lookahead* é atualizado para `[` e confrontado com o segundo filho à esquerda da raiz. Como há nova coincidência entre o rótulo e *lookahead*, a construção prossegue
- (e) O nó seguinte contém o não-terminal *primitivo* e *lookahead* contém o token `num`. Assim a terceira produção de *primitivo* é utilizada para gerar os novos filhos



## Exemplo de construção *top-down* da árvore gramatical

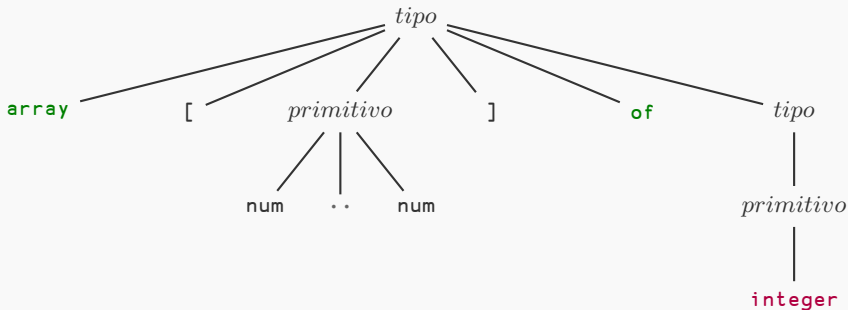
(g) Os próximos tokens ( : , num , of ) coincidem com os respectivos filhos

## Exemplo de construção *top-down* da árvore gramatical

- (g) Os próximos tokens (`:`, `num`, `of`) coincidem com os respectivos filhos
- (h) O último valor que *lookahead* assum é `integer`, o qual é confrontado com o filho mais à direita da raiz. Como o nó tem como rótulo o não-terminal *tipo*, a primeira produção deste deve ser usada para construir o novo nó, que por sua vez usa a primeira produção de *primitivo* para construir seu único filho

## Exemplo de construção *top-down* da árvore gramatical

- (g) Os próximos tokens (`:`, `num`, `of`) coincidem com os respectivos filhos
- (h) O último valor que *lookahead* assum é `integer`, o qual é confrontado com o filho mais à direita da raiz. Como o nó tem como rótulo o não-terminal *tipo*, a primeira produção deste deve ser usada para construir o novo nó, que por sua vez usa a primeira produção de *primitivo* para construir seu único filho



# Análise gramatical preditiva

- ▶ Uma análise gramatical descendente recursiva é um método *top-down* de análise sintática na qual são executados procedimentos recursivos para processar a entrada

# Análise gramatical preditiva

- ▶ Uma análise gramatical descendente recursiva é um método *top-down* de análise sintática na qual são executados procedimentos recursivos para processar a entrada
- ▶ Cada não-terminal da entrada é associado a um procedimento



## Análise gramatical preditiva

- ▶ Uma análise gramatical descendente recursiva é um método *top-down* de análise sintática na qual são executados procedimentos recursivos para processar a entrada
- ▶ Cada não-terminal da entrada é associado a um procedimento
- ▶ Se *lookahead* determina, sem ambiguidades, o procedimento a ser executado, a análise gramatical descendente recursiva é denominada análise gramatical preditiva

## Análise gramatical preditiva

- ▶ Uma análise gramatical descendente recursiva é um método *top-down* de análise sintática na qual são executados procedimentos recursivos para processar a entrada
- ▶ Cada não-terminal da entrada é associado a um procedimento
- ▶ Se *lookahead* determina, sem ambiguidades, o procedimento a ser executado, a análise gramatical descendente recursiva é denominada análise gramatical preditiva
- ▶ A sequência de chamadas de procedimentos no processamento da entrada determina, de forma implícita, a árvore gramatical

## Análise gramatical preditiva

- ▶ Uma análise gramatical descendente recursiva é um método *top-down* de análise sintática na qual são executados procedimentos recursivos para processar a entrada
- ▶ Cada não-terminal da entrada é associado a um procedimento
- ▶ Se *lookahead* determina, sem ambiguidades, o procedimento a ser executado, a análise gramatical descendente recursiva é denominada análise gramatical preditiva
- ▶ A sequência de chamadas de procedimentos no processamento da entrada determina, de forma implícita, a árvore gramatical
- ▶ Além dos procedimentos associados aos não-terminais, a análise pode definir outros procedimentos auxiliares que podem simplificar tarefas como a leitura de tokens e a atualização de *lookahead*

## Reconhecimento de tokens

O procedimento `RECONHECER()` confronta o valor de *lookahead* e um determinado token. Em caso de coincidência, ele atualiza *lookahead* com o próximo token da entrada.

## Reconhecimento de tokens

O procedimento `RECONHECER()` confronta o valor de *lookahead* e um determinado token. Em caso de coincidência, ele atualiza *lookahead* com o próximo token da entrada.

1: **procedure** `RECONHECER(token)`

2:     **if** *lookahead* = *token* **then**

3:         *lookahead*  $\leftarrow$  `PROXIMOTOKEN()`

4:     **else**

5:         `ERRO()`

▷ *lookahead* é uma variável global

## Procedimento associado ao não terminal *tipo*

```
1: procedure TIPO(  
2:   if lookahead  $\in$  { integer, char, num } then  
3:     PRIMITIVO(  
4:   else if lookahead =  $\uparrow$  then  
5:     RECONHECER( $\uparrow$ )  
6:     RECONHECER(i d)  
7:   else if lookahead = array then  
8:     RECONHECER(array),  
9:     RECONHECER([)  
10:    PRIMITIVO(  
11:    RECONHECER(]),  
12:    RECONHECER(of)  
13:    TIPO(  
14:  else  
15:    ERRO(  

```

## Procedimento associado ao não terminal *primitivo*

```
1: procedure PRIMITIVO( )  
2:   if lookahead = integer then  
3:     RECONHECER(integer)  
4:   else if lookahead = char then  
5:     RECONHECER(char)  
6:   else if lookahead = num then  
7:     RECONHECER(num)  
8:     RECONHECER(:)  
9:     RECONHECER(num)  
10:  else  
11:    ERRO( )
```

# Primeiros símbolos

## Definição de primeiros símbolos

Seja  $\alpha$  o lado direito de uma produção. Então  $\text{PRIMEIRO}(\alpha)$  é o conjunto de tokens que figuram como primeiros símbolos de uma ou mais cadeias geradas a partir de  $\alpha$ . Se  $\epsilon$  pode ser gerado a partir de  $\alpha$ , então  $\epsilon$  pertence a  $\text{PRIMEIRO}(\alpha)$ .

Por exemplo, na gramática de geração de subtipos em Pascal,

$$\text{PRIMEIRO}(\textit{primitivo}) = \{ \text{integer}, \text{char}, \text{num} \}$$

e

$$\text{PRIMEIRO}(\uparrow \text{id}) = \{ \uparrow \}$$



## Primeiros símbolos e análise gramatical preditiva

- ▶ A análise gramatical preditiva depende dos conjuntos  $\text{PRIMEIRO}(X)$  de todos não-terminais  $X$  da gramática

## Primeiros símbolos e análise gramatical preditiva

- ▶ A análise gramatical preditiva depende dos conjuntos  $\text{PRIMEIRO}(X)$  de todos não-terminais  $X$  da gramática
- ▶ Isto acontece principalmente nos casos onde a gramática possui duas ou mais produções para um mesmo não-terminal (por exemplo,  $A \rightarrow \alpha$  e  $A \rightarrow \beta$ )

## Primeiros símbolos e análise gramatical preditiva

- ▶ A análise gramatical preditiva depende dos conjuntos  $\text{PRIMEIRO}(X)$  de todos não-terminais  $X$  da gramática
- ▶ Isto acontece principalmente nos casos onde a gramática possui duas ou mais produções para um mesmo não-terminal (por exemplo,  $A \rightarrow \alpha$  e  $A \rightarrow \beta$ )
- ▶ Para que a análise gramatical recursiva descendente seja preditiva é necessário que os primeiros símbolos de cada produção sejam distintos

## Primeiros símbolos e análise gramatical preditiva

- ▶ A análise gramatical preditiva depende dos conjuntos  $\text{PRIMEIRO}(X)$  de todos não-terminais  $X$  da gramática
- ▶ Isto acontece principalmente nos casos onde a gramática possui duas ou mais produções para um mesmo não-terminal (por exemplo,  $A \rightarrow \alpha$  e  $A \rightarrow \beta$ )
- ▶ Para que a análise gramatical recursiva descendente seja preditiva é necessário que os primeiros símbolos de cada produção sejam distintos
- ▶ No exemplo dado,

$$\text{PRIMEIRO}(\alpha) \cap \text{PRIMEIRO}(\beta) = \emptyset$$

## Primeiros símbolos e análise gramatical preditiva

- ▶ A análise gramatical preditiva depende dos conjuntos  $\text{PRIMEIRO}(X)$  de todos não-terminais  $X$  da gramática
- ▶ Isto acontece principalmente nos casos onde a gramática possui duas ou mais produções para um mesmo não-terminal (por exemplo,  $A \rightarrow \alpha$  e  $A \rightarrow \beta$ )
- ▶ Para que a análise gramatical recursiva descendente seja preditiva é necessário que os primeiros símbolos de cada produção sejam distintos
- ▶ No exemplo dado,

$$\text{PRIMEIRO}(\alpha) \cap \text{PRIMEIRO}(\beta) = \emptyset$$

- ▶ Caso esta condição se verifique para todos os pares de produções distintas de um mesmo não-terminal, a produção  $\gamma$  deve ser usada se  $lookahead \in \text{PRIMEIRO}(\gamma)$

## Projeto de um analisador gramatical preditivo

Um analisador gramatical preditivo é um programa que contém um procedimento para cada não-terminal. Cada procedimento deve seguir dois passos:

## Projeto de um analisador gramatical preditivo

Um analisador gramatical preditivo é um programa que contém um procedimento para cada não-terminal. Cada procedimento deve seguir dois passos:

1. Determinar a produção a ser usada a partir de *lookahead*. Para tanto, deve ser localizada, entre as produções  $\alpha_1, \alpha_2, \dots, \alpha_N$ , a produção  $\alpha_i$  tal que  $lookahead \in \text{PRIMEIRO}(\alpha_i)$  (deve valer a seguinte propriedade:  $\text{PRIMEIRO}(\alpha_i) \cap \text{PRIMEIRO}(\alpha_j) = \emptyset$  se  $i \neq j$ ). Se  $\alpha_k = \epsilon$  para algum  $k$ ,  $\alpha_k$  deve ser usada se *lookahead* não estiver presente em nenhuma outra produção

## Projeto de um analisador gramatical preditivo

Um analisador gramatical preditivo é um programa que contém um procedimento para cada não-terminal. Cada procedimento deve seguir dois passos:

1. Determinar a produção a ser usada a partir de *lookahead*. Para tanto, deve ser localizada, entre as produções  $\alpha_1, \alpha_2, \dots, \alpha_N$ , a produção  $\alpha_i$  tal que  $lookahead \in \text{PRIMEIRO}(\alpha_i)$  (deve valer a seguinte propriedade:  $\text{PRIMEIRO}(\alpha_i) \cap \text{PRIMEIRO}(\alpha_j) = \emptyset$  se  $i \neq j$ ). Se  $\alpha_k = \epsilon$  para algum  $k$ ,  $\alpha_k$  deve ser usada se *lookahead* não estiver presente em nenhuma outra produção
2. Identificada a produção, o procedimento imita a produção, reconhecendo os terminais da produção e chamando os procedimentos dos não-terminais, na mesma ordem da produção



# Produções recursivas à esquerda

## Definição de produção recursiva à esquerda

Uma produção é recursiva à esquerda se o não-terminal à esquerda da produção figura como primeiro símbolo da produção. Por exemplo, se  $\alpha$  e  $\beta$  são sequências de terminais e não-terminais que não iniciam em  $A$ , então a produção

$$A \rightarrow A\alpha \mid \beta$$

é recursiva à esquerda.

# Produções recursivas à esquerda

## Definição de produção recursiva à esquerda

Uma produção é recursiva à esquerda se o não-terminal à esquerda da produção figura como primeiro símbolo da produção. Por exemplo, se  $\alpha$  e  $\beta$  são sequências de terminais e não-terminais que não iniciam em  $A$ , então a produção

$$A \rightarrow A\alpha \mid \beta$$

é recursiva à esquerda.

Observação: analisadores gramaticais recursivos descendentes pode rodar indefinidamente caso usem uma produção recursiva à esquerda

# Produções recursivas à direita

## Definição de produção recursiva à direita

Uma produção é recursiva à direita se o não-terminal à esquerda da produção figura como último símbolo da produção. Por exemplo, se  $\alpha$  e  $\beta$  são sequências de terminais e não-terminais que não terminam em  $R$ , então a produção

$$\begin{aligned} A &\rightarrow \beta R \\ R &\rightarrow \alpha R \mid \epsilon \end{aligned}$$

é recursiva à direita.

## Produções recursivas à direita

### Definição de produção recursiva à direita

Uma produção é recursiva à direita se o não-terminal à esquerda da produção figura como último símbolo da produção. Por exemplo, se  $\alpha$  e  $\beta$  são sequências de terminais e não-terminais que não terminam em  $R$ , então a produção

$$\begin{aligned} A &\rightarrow \beta R \\ R &\rightarrow \alpha R \mid \epsilon \end{aligned}$$

é recursiva à direita.

Observação: produções recursivas à direita dificultam a tradução de expressões que contém operadores associativos à esquerda

# Referências

1. **AHO**, Alfred V, **SETHI**, Ravi, **ULLMAN**, Jeffrey D. *Compiladores: Princípios, Técnicas e Ferramentas*, LTC Editora, 1995.