Gramáticas Livres de Contexto e Análise Sintática

João Marcelo Uchôa de Alencar
joao.marcelo@ufc.br
UFC-Quixadá

Análise Sintática

O que é?

- Estrutura do programa;
- usa convenções e operações similares à ER;
- porém admite **recursividade**;
- árvore sintática;
- técnicas **ascendentes** e **descendentes**.

O Processo de Análise Sintática

analisador sintático(sequência de marcas) = árvore sintática

- Em geral, o analisador sintático ativa *getToken* para capturar a próxima marca sob demanda;
- syntaxTree = parse();
  - A árvore é uma estrutura dinâmica;
  - existem vários tipos de nós, de acordo com as regras da gramática;
  - registro de tamanho variável, com atributos para as próximas fases.
- **recuperação de erros**: todo o código precisa ser analisado para exibir a maior quantidade de erros encontrados.

Gramáticas Livres de Contexto

Gramática

$exp \rightarrow exp\ op\ exp \mid (exp) \mid \textbf{número}$

$op \rightarrow + \mid - \mid *$

Expressão Regular

$\textbf{número} = \textbf{dígito}\ \textbf{dígito}^*$

$\textbf{dígito} = 0|1|2|3|4|5|6|7|8|9$

Comparação com a Notação de Expressão Regular

- ER: escolha, concatenação, repetição;
- gramáticas: não há a repetição da mesma forma;
- ER: sinal de $=$ para atribuir nome a uma expressão;
- gramáticas: sinal de $\rightarrow$ para definir uma regra;
- as ERs aparecem nas regras das gramáticas.

Especificação de Regras de Gramáticas

- Alfabeto de símbolos das gramáticas: **marcas** que representam cadeias;
- a cadeia *if*, em uma regra, representa a marca **IF**;
- estrutura de uma **regra**:
  - nome de uma estrutura;
  - meta-símbolo $\rightarrow$;
  - símbolos do alfabeto, nome de estrutura ou o meta-símbolo $|$.
- cada autor tem sua preferência nos símbolos utilizados ($=$, $:=$, $::=$, $<>$, etc)
- parênteses podem ser usados para definir precedência, apesar de poderem ser substituídos por regras adicionais.

Derivações e a Linguagem de uma Gramática

Cadeias de Marcas Legais

As regras de gramática livres de contexto determinam o conjunto de cadeias de símbolos representando marcas consideradas sintaticamente legais dentro das estruturas de linguanges.

- $(34 - 3) * 42$ corresponde à $(\textit{número} - \textit{número}) * \textit{número}$;
- já $(34 - 3 * 42$ não é legal, apesar das marcas serem válidas.

Derivação

Sequência de substituições de nomes de estruturas por escolhas à direita das regras gramaticais. A cada passo na derivação, uma única substituição é efetuada com base em uma escolha de regra gramatical.

Exemplo de Derivação

1.	$exp \Rightarrow exp\ op\ exp$	$[exp \rightarrow exp\ op\ exp]$
2.	$\Rightarrow$ exp op **número**	$[exp \rightarrow$ **número**$]$
3.	$\Rightarrow$ exp $*$ **número**	$[op \rightarrow *]$
4.	$\Rightarrow$ (exp) $*$ **número**	$[exp \rightarrow (exp)]$
5.	$\Rightarrow$ (exp op exp) $*$ **número**	$[exp \rightarrow exp\ op\ exp]$
6.	$\Rightarrow$ (exp op **número**) $*$ **número**	$[exp \rightarrow$ **número**$]$
7.	$\Rightarrow$ (exp - **número**) $*$ **número**	$[op \rightarrow -]$
8.	$\Rightarrow$ (**número** - **número**) $*$ **número**	$[exp \rightarrow$ **número**$]$

Linguagem Definida pela Gramática

- O conjunto de cadeias de símbolos para marcas obtidos por derivações para *exp* é a **linguagem definida pela gramática**;
- $L(G) = \{s | exp \stackrel{*}{\Rightarrow} s\}$
- cada nome de estrutura (*exp*, *op*) define sua própria linguagem;
- a gramática de uma linguagem adota uma estrutura base, em geral chamada *programa*, (**símbolo inicial**);
- **símbolos não terminais**: estruturas;
- **símbolos terminais**: marcas.

Exemplos

1. $E \rightarrow (E)|a$, gera a linguagem $L(G) = \{({}^n a)^n | n >= 0\}$;
2. $E \rightarrow (E)$, gera a linguagem $L(G) = \{\}$;
3. $E \rightarrow E + a|a$;
4. *declaração* $\rightarrow$ *if-decl* | **outra**
   if-decl $\rightarrow$ **if (** *exp* **)** *declaração*
   | **if (** *exp* **)** *declaração* **else** *declaração*
   exp $\rightarrow$ **0** | **1**

Mais Exemplos

1. **Recursão à esquerda**: $A \rightarrow Aa|a$;
2. **Recursão à direita**: $A \rightarrow aA|a$;
3. $A \rightarrow (A)A|\varepsilon$:
4. *declaração* $\rightarrow$ *if-decl* | **outra**
   if-decl $\rightarrow$ **if (** *exp* **)** *declaração else-parte*
   else-parte $\rightarrow$ **else** *declaração* $|\varepsilon$
   exp $\rightarrow$ **0** | **1**
5. *decl-sequência* $\rightarrow$ *decl* **;** *decl-sequência* | *decl*
   decl $\rightarrow$ **s**
6. *decl-sequência* $\rightarrow$ *decl* **;** *decl-sequência* $|\varepsilon$
   decl $\rightarrow$ **s**

Árvores de Análise Sintática e Árvores Sintáticas Abstratas

Mesma cadeia final, diferentes derivações.

1.	$exp \Rightarrow exp\ op\ exp$	$[exp \rightarrow exp\ op\ exp]$
2.	$\Rightarrow$ (exp) op exp	$[exp \rightarrow (\text{exp})]$
3.	$\Rightarrow$ (exp op exp) op expr	$[exp \rightarrow \text{exp op exp}]$
4.	$\Rightarrow$ (**número** op exp) op exp	$[exp \rightarrow \textbf{número}]$
5.	$\Rightarrow$ (**número** $-$ exp) op exp	$[op \rightarrow -]$
6.	$\Rightarrow$ (**número** $-$ **número**) op exp	$[exp \rightarrow \textbf{número}]$
7.	$\Rightarrow$ (**número** $-$ **número**) $*$ exp	$[op \rightarrow *]$
8.	$\Rightarrow$ (**número** $-$ **número**) $*$ **número**	$[exp \rightarrow \textbf{número}]$

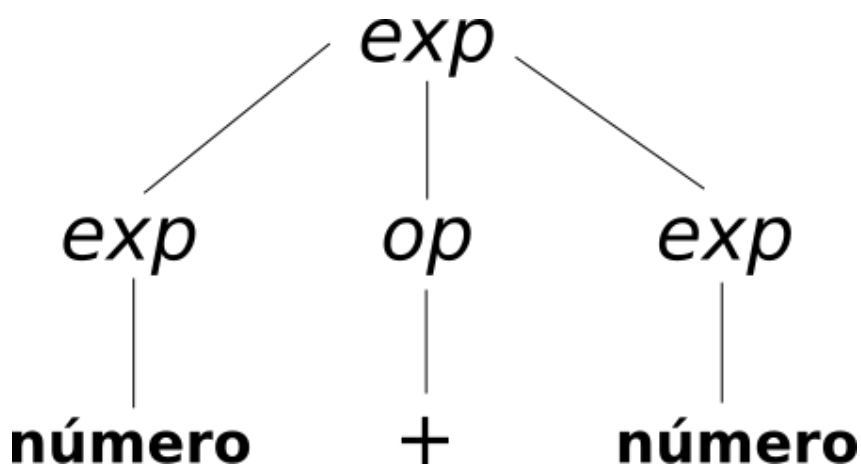
Precisamos de uma representação que abstraia as características essenciais de uma derivação e fatore as diferenças superficiais de ordem.

Árvores

Árvore de Análise Sintática

Corresponde a uma derivação, com seus nós interiores rotulados por não-terminais, nós folhas rotulados por terminais e filhos de cada nó interno representando a substituição do não-terminal associado em um passo de derivação.

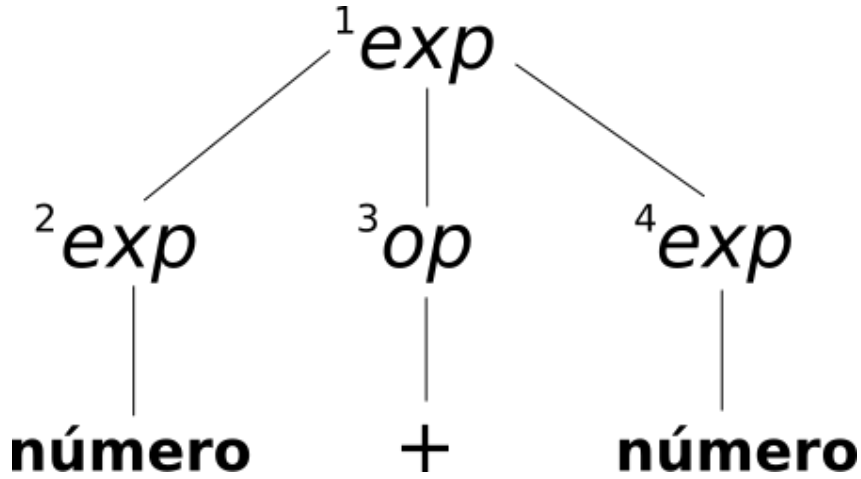
$$
\begin{aligned}
exp &\Rightarrow exp\ op\ exp \\
&\Rightarrow \textbf{número}\ op\ exp \\
&\Rightarrow \textbf{número}\ +\ exp \\
&\Rightarrow \textbf{número}\ +\ \textbf{número}
\end{aligned}
$$



Percurso em Árvores

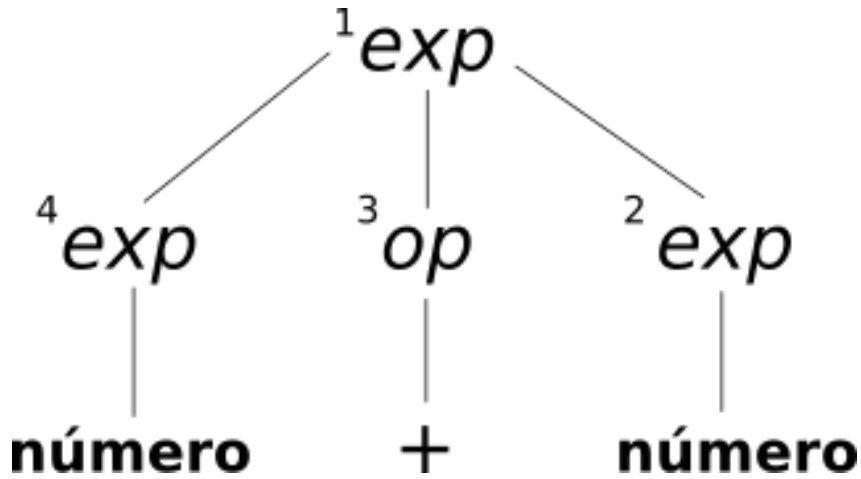
Pré-ordem:

1. $exp \Rightarrow exp\ op\ exp$
2. $\Rightarrow$ **número** $op\ exp$
3. $\Rightarrow$ **número** $+\ exp$
4. $\Rightarrow$ **número** $+$ **número**



Pós-ordem invertida:

1. $exp \Rightarrow exp\ op\ exp$
2. $\Rightarrow exp\ op$ **número**
3. $\Rightarrow exp +$ **número**
4. $\Rightarrow$ **número** $+$ **número**



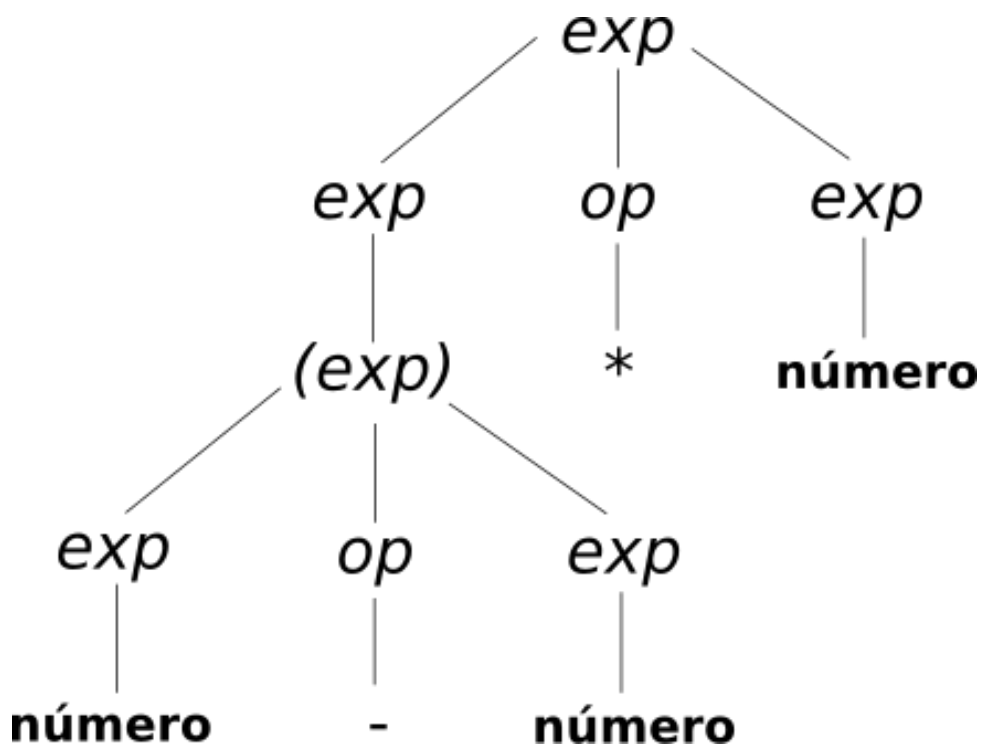
Derivações e Árvores

Correspondência

Uma árvore de análise sintática corresponde, em geral, a muitas derivações, as quais representam a mesma estrutura básica para as cadeias de terminais analisadas.

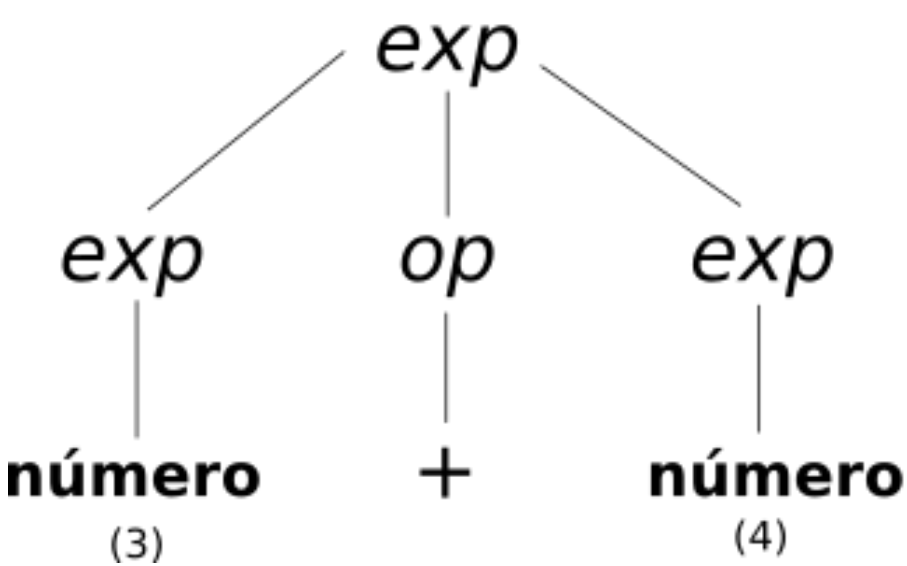
- **Derivação mais à esquerda**: não terminal mais à esquerda é substituído a cada passo da derivação;
- **derivação mais à direita**: não terminal mais à direita é substituído a cada passo da derivação

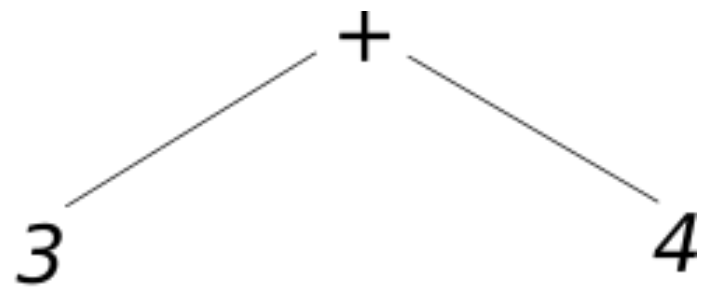
Árvores (34 - 3)*42



Árvores Sintáticas Abstratas

As árvores de análise sintática podem ser resumidas em árvores sintáticas (abstratas). Para a expressão $3 + 4$:

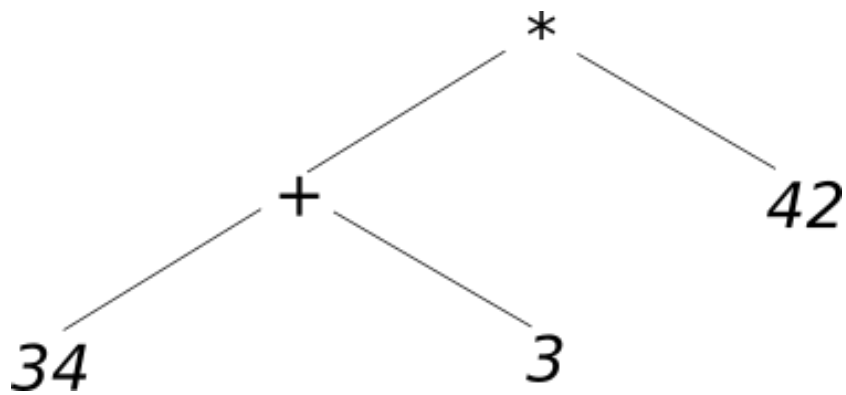




Árvores Sintáticas Abstratas

Construção

Um analisaddor sintático efetua todos os passos na árvore de análise sintática, mas, em geral, constrói apenas uma árvore sintática abstrata.



Sintaxe Abstrata

$OpExp(Mult, OpExp(Subtr, ConstExp(34), ConstExp(3)), ConstExp(42))$

Representação em Código

```
typedef enum {Plus, Minus, Times} OpKind;
typedef enum {OpK, ConstK} ExpKind;
typedef struct streenode {
   ExpKind kind;
   OpKind op;
   struct streenode *lchild, *rchild;
   int val;
} STreeNode;
typedef STreeNode *SyntaxTree;
```

Exemplos

declaração → *if-decl* | **outra**
if-decl → **if** **(** *exp* **)** *declaração*
| **if** **(** *exp* **)** *declaração*
else *declaração*
exp → **0** | **1**

declaração → *if-decl* | **outra**
if-decl → **if** **(** *exp* **)** *declaração*
else-parte
else-parte → **else** *declaração* |ε
exp → **0** | **1**

if (0) outra else outra

Qual a árvore de análise sintática? Qual seria uma possível árvore sintática abstrata?

Representação em Código

```
typedef enum {ExpK, StmtK} NodeKind;
typedef enum {Zero, One} ExpKind;
typedef enum {IfK, OtherK} StmtKind;
typedef struct streenode {
   NodeKind kind;
   ExpKind ekind;
   StmtKind skind;
   struct streenode *test, *thenpart, *elsepart;
} STreeNode;
typedef STreeNode *SyntaxTree;
```

Mais Exemplos

decl-sequência $\rightarrow$ *decl* **;** *decl-sequência* | *decl*

decl $\rightarrow$ **s**

Árvores para s;s;s ?

Ambigüidade

Gramática

$exp \rightarrow exp\ op\ exp \mid (exp) \mid$ **número**

$op \rightarrow + \mid - \mid *$

- $34 - 3 * 42$
- Quais as derivações à esquerda e à direita?
- Quais as respectivas árvores?

Ambigüidade

- Uma gramática que gera uma cadeia com duas árvores de análise sintática distintas é denominada **gramática ambígua**;
- pense nela como um NFA, chega um momento na derivação que você pode fazer duas escolhas;
- infelizmente, diferente do caso NFA $\rightarrow$ DFA, não há transformação automática de gramática ambígua para não ambígua;
- soluções:
  - Estabelecer uma regra externa para decidir qual caminho tomar;
  - alterar a gramática para eliminar a ambigüidade.
- **Tradução Orientada pela Sintaxe**: escolher a árvore ou derivação que reflita melhor o significado esperado para as próximas fases da compilação.

Ambigüidade

Precedência

$exp \rightarrow exp\ soma\ exp | termo$
$soma \rightarrow + | -$
$termo \rightarrow termo\ mult\ termo | fator$
$mult \rightarrow *$
$fator \rightarrow (exp) | \textbf{número}$

Associatividade

$exp \rightarrow exp\ soma\ termo | termo$
$soma \rightarrow + | -$
$termo \rightarrow termo\ mult\ fator | fator$
$mult \rightarrow *$
$fator \rightarrow (exp) | \textbf{número}$

Como ficariam as árvores para $34 - 3 * 42$ e $34 - 3 - 42$?

O Problema do Else Pendente

declaração $\rightarrow$ *if-decl* | **outra**
if-decl $\rightarrow$ **if (** *exp* **)** *declaração*
| **if (** *exp* **)** *declaração* **else** *declaração*
exp $\rightarrow$ **0** | **1**

Qual a árvore para **if (0) if (1) outra else outra** ?

O Problema do Else Pendente

Apenas casam-decl antes de um else

declaração $\rightarrow$ *casam-decl* | *sem-casam-decl*
casam-decl $\rightarrow$ **if** (*exp*) *casam-decl* **else** *casam-decl* | **outra**
sem-casam-decl $\rightarrow$ **if** (*exp*) *declaração*
| **if** (*exp*) *casam-decl* **else** *sem-casam-decl*
exp $\rightarrow$ **0** | **1**

- Torna a gramática mais complexa;
- melhor determinar uma marca para finalizar o **if**.

if-decl $\rightarrow$ **if** *condição* **then** *seq-decl* **end if**
| **if** *condição* **then** *seq-decl* **else** *seq-decl* **end if**

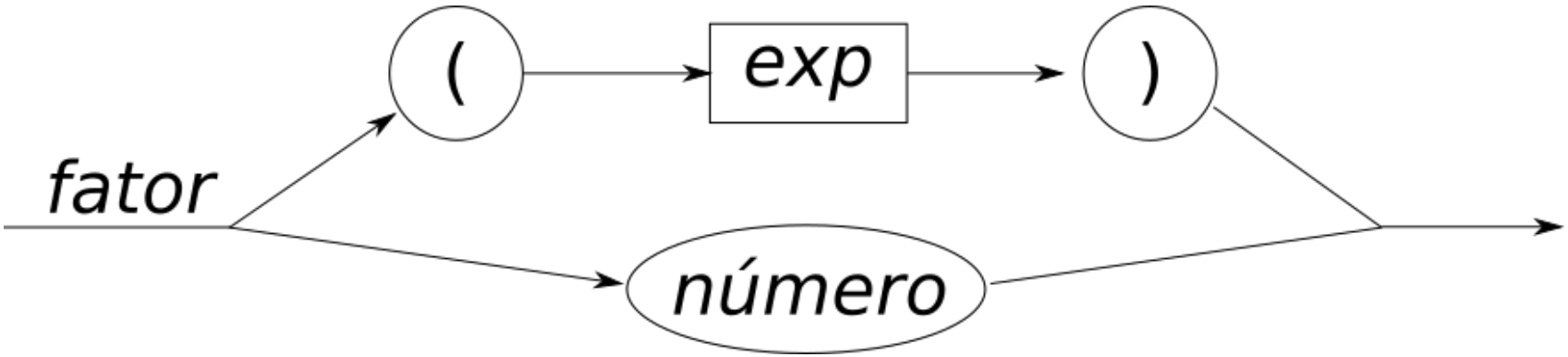
Notações Estendidas

EBNF - Repetição e Opcionais

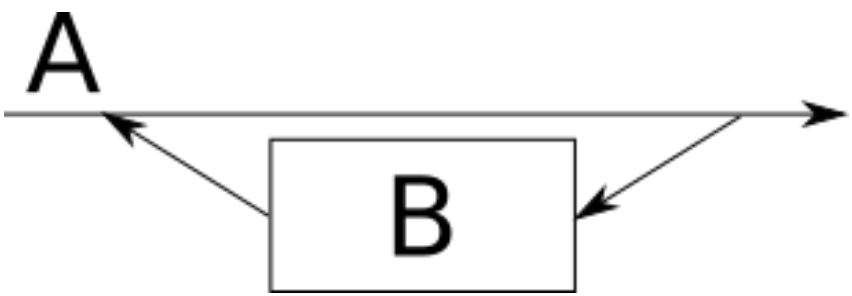
- $A \rightarrow A\alpha|\beta$ vira $A \rightarrow \beta\{\alpha\}$
- $A \rightarrow \alpha A|\beta$ vira $A \rightarrow \{\alpha\}\beta$
- $exp \rightarrow termo\ [soma\ exp]$

Diagramas Sintáticos

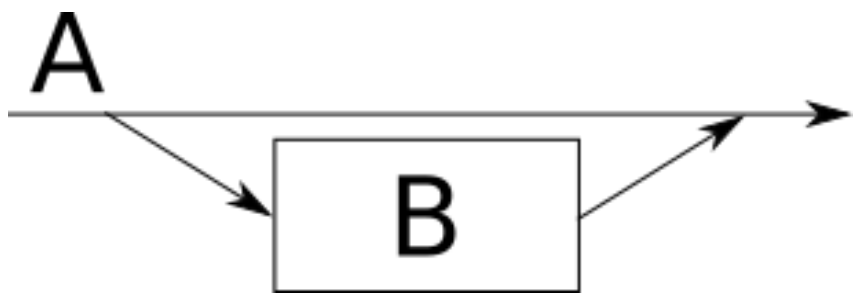
$$fator \rightarrow (exp) | \textbf{número}$$



$A \rightarrow \{B\}$



$A \rightarrow [B]$



Exemplos de Diagramas Sintáticos

Quais são os diagramas?

declaração $\rightarrow$ *if-decl* | **outra**
if-decl $\rightarrow$ **if** (*exp*) *declaração*
| **if** (*exp*) *declaração* **else** *declaração*
exp $\rightarrow$ **0** | **1**

exp $\rightarrow$ *termo* { *soma termo* }
soma $\rightarrow$ **+** | **-**
termo $\rightarrow$ *fator* { *mult fator* }
mult $\rightarrow$ $*$
fator $\rightarrow$ (*exp*)| **número**

Propriedades Formais

Uma **gramática livre de contexto** consiste de:

1. Um conjunto T de **terminais**;
2. um conjunto N de **não-terminais** (disjunto de T);
3. um conjunto P de **produções**, ou **regras gramaticais**, da forma $A \to \alpha$, em que A é um elemento de N e α é um elemento de $(T \cup N)^*$;
4. um **símbolo inicial** S do conjunto N.

Propriedades Formais

Seja uma gramática $G = (T, N, P, S)$

- **Passo de derivação**: $\alpha A \gamma \Rightarrow \alpha B \gamma$, em que α e γ são elementos de $(T \cup N)^*$ e $A \rightarrow B$ pertence a P;
- **Derivação**: $\alpha \overset{*}{\Rightarrow} \beta$ se e somente se $\alpha_1 \Rightarrow \alpha_2 \Rightarrow ... \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$, com $\alpha_1 = \alpha$ e $\alpha_n = \beta$;
- $L(G) = \{w \in T^* |$ existe uma derivação $S \overset{*}{\Rightarrow} w$ de G$\}$

Propriedades Formais

Uma árvore de análise sintática sobre G:

1. Cada nó é rotulado com um terminal, um não terminal ou ε;
2. o nó raiz é rotulado coom o símbolo inicial S;
3. cada nó folha é rotulado com um terminal ou ε;
4. cada nó não folha é rotulado como um não terminal;
5. Se um nó com rótulo $A \in N$ tiver n filhos com rótulos $X_1, X_2, ..., X_n$ (terminais ou não), então $A \rightarrow X_1 X_2 ... X_n \in P$ (produção).

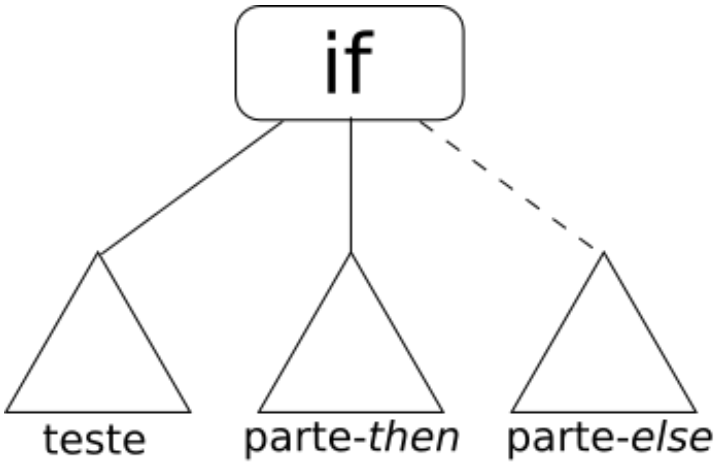
Sintaxe da TINY

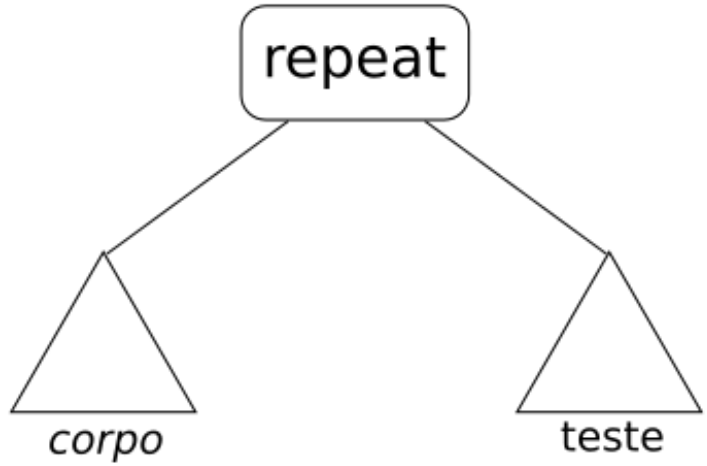
programa → *decl-sequência*
decl-sequência → *decl-sequência*; *declaração* | *declaração*
declaração → *cond-decl* | *repet-decl* | *atrib-decl* | *leit-decl* | *escr-decl*
cond-decl → **if** *exp* **then** *decl-sequência* **end** | **if** exp **then** *decl-sequência* **else** *decl-sequência* **end**
repet-decl → **repeat** *decl-sequência* **until** *exp*
atrib-decl → **identificador** := *exp*
leit-decl → **read identificador**
escr-decl → **write** *exp*
exp → *exp-simples comp-op exp-simples* | *exp-simples*
comp-op →< | =
exp-simples → *exp-simples soma termo* | *termo*
soma → +|−
termo → *termo mult fator* | *fator*
mult → ∗|/
fator → *(exp)* | **número** | **identificador**

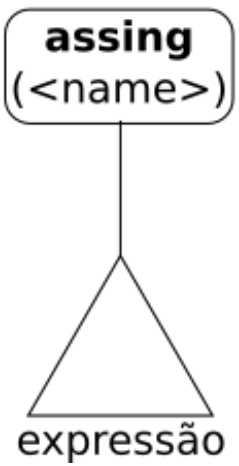
Estruturas da Árvore Sintática Abstrata da TINY

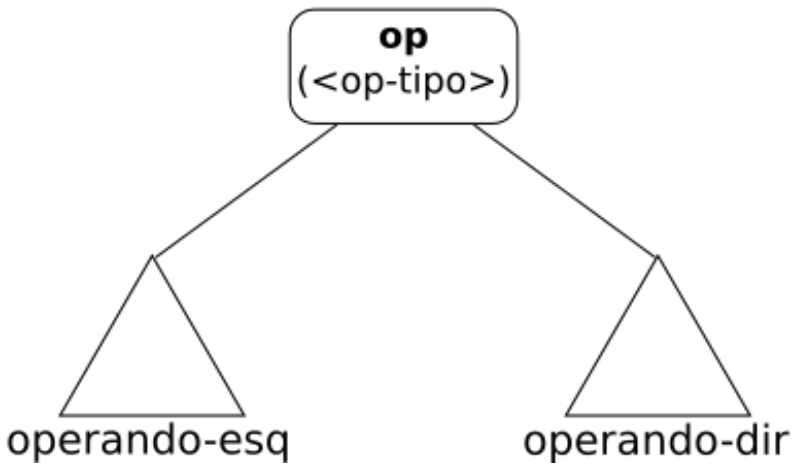
```
typedef enum {StmtK, ExpK} NodeKind;
typedef enum {IfK, RepeatK, AssignK, ReadK, WriteK}
             StmtKind;
typedef enum {OpK, ConstK, IdK} ExpKind;
/* ExpType é utilizado para verificação de tipos. */
typedef enum {Void, Integer, Boolean} ExpType;
#define MAXCHILDREN 3
typedef struct treeNode {
   struct treeNode *child[MAXCHILDREN];
   struct treeNode *sibling;
   int lineno;
   NodeKind nodekind;
   union { StmtKind stmt; ExpKind exp;} kind;
   union { TokenType op;
           int val;
           char *name; } attr;
   ExpType type;
} TreeNode;
```

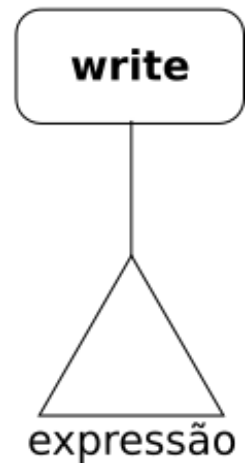
Estruturas da Árvore Sintática Abstrata da TINY











Programa Exemplo na Linguagem TINY

```
{
   Programa de exemplo na
   linguagem TINY - computa o fatorial
}
read x; {entrada de um inteiro}
if 0 < x then { não calcula se x <= 0}
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1;
   until x = 0;
   write fact {saída do fatorial de x}
end
```

Qual seria a árvore sintática abstrata baseada nas estruturas?

Fim

Dúvidas?