Linguagens Formais

UNICAP

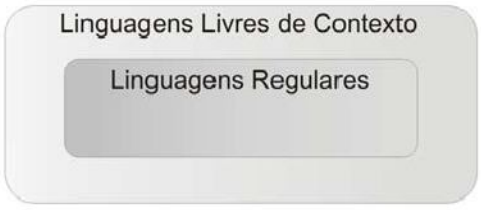
Eduardo Araújo Oliveira
http://sites.google.com/site/eaoufpe

UNIVERSIDADE CATÓLICA DE PERNAMBUCO

Gramáticas Livre de Contexto

As gramáticas livres de contexto reconhecem todas as linguagens regulares e mais algumas;

O diagrama abaixo, que é parte da chamada Hierarquia de Chomsky, representa a relação entre as duas classes de linguagens:



Gramáticas Livre de Contexto

Os autômatos reconhecem as cadeias que estão na linguagem e rejeitam as que não estão; já as expressões regulares são um “formato” geral que todas as cadeias da linguagem obedecem.

E as gramáticas livres de contexto, como funcionam?

slide 3

Gramáticas Livre de Contexto

As gramáticas livres de contexto possuem regras para formar cadeias, de modo que apenas as cadeias que são da linguagem podem ser formadas usando as regras, enquanto as cadeias que não são da linguagem não podem ser formadas de maneira nenhuma pelas regras.

slide 4

Linguagens Livres de Contexto

- Também chamadas de “Tipo-2”
- É uma classe mais ampla do que as Linguagens Regulares
- Inclui as Linguagens Regulares
  - Toda Linguagem Regular é Livre de Contexto

5

Linguagens Livres de Contexto

- Permitem estruturar melhor as palavras
- Diferente das Linguagens Regulares, são capazes de tratar
  - Balanceamento de parênteses
  - Construções em bloco
- Uso em Linguagens Computacionais

6

Linguagens Livres de Contexto

Dois formalismos serão vistos

- Um formalismo gerador
  - Gramáticas Livres de Contexto
- Um formalismo reconhecedor
  - Autômatos com Pilha

Gramáticas Livre de Contexto - EXEMPLO

A linguagem das cadeias na forma 0^n1^n (n ocorrências do símbolo 0 seguidas de n ocorrências do símbolo 1), para todo $n > 0$. (01, 0011, 000111, 00001111, ...)

0^n1^n

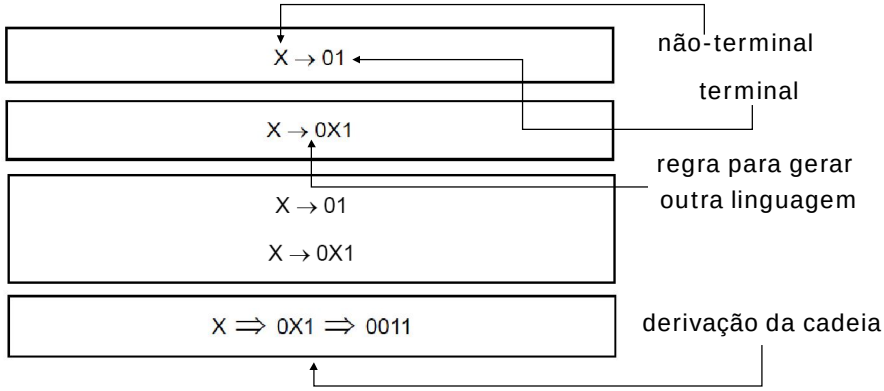
(n ocorrências do símbolo 0 seguidas de n ocorrências do símbolo 1)

As regras usam símbolos auxiliares, chamados não-terminais, que servem de ponto de partida para gerar alguma cadeia.

Gramáticas Livre de Contexto - EXEMPLO

0^n1^n

(n ocorrências do símbolo 0 seguidas de n ocorrências do símbolo 1)



slide 9

Gramáticas livre de contexto – Definição Formal

G = (V, T, P, S)

V é o alfabeto que contém os símbolos **não-terminais** ou **variáveis**, que são os símbolos auxiliares no processo de geração de cadeias (como o símbolo X do exemplo).

T é o alfabeto dos símbolos **terminais**

P é o conjunto das **regras** ou **produções** da gramática

S é o símbolo não-terminal de início, é usado para iniciar as derivações de cadeias da linguagem. Símbolo inicial ou variável de inicio.

Para o exemplo anterior: **G1 = ({X}, {0,1}, {X → 01, X → 0X1}, X)**

slide 10

Gramáticas livre de contexto

Esta é uma maneira mais sutil de representar linguagens do que com autômatos. Enquanto os autômatos são usados para testar qualquer palavra dada diretamente, nas gramáticas costumamos partir das regras para descobrir as palavras que são válidas.

slide 11

Gramáticas livre de contexto – Definição Formal

- As letras maiúsculas representarão símbolos não-terminais.
- Todos os outros símbolos serão considerados terminais, automaticamente.
- A cabeça da primeira produção será o símbolo inicial.

As convenções têm o objetivo de facilitar o entendimento dos exemplos, mas elas *não* são obrigatórias. Não deixa de ser uma gramática livre de contexto se usar uma letra maiúscula para um terminal, por exemplo.

slide 12

Gramáticas livre de contexto – Definição Formal

P_2:

$$E \rightarrow N \mid E+E \mid ExE$$

$$N \rightarrow 0 \mid 1 \mid 0N \mid 1N$$

G2 = ({E, N}, {+, x, 0, 1}, P2, E)

slide 13

Derivação

Maneira mais genérica do processo pelo qual você pode derivar cadeias da linguagem:

1. Escreva o símbolo não-terminal de início. Este símbolo sozinho será a cadeia inicial com a qual você vai trabalhar.
2. Escolha uma ocorrência de um não-terminal N qualquer na cadeia e escolha alguma produção de N. Na cadeia, substitua a ocorrência de N pelo corpo da produção.
3. Repita o passo 2 até que não reste nenhuma variável.

slide 14

Gramáticas livre de contexto – Definição Formal

P_2:

$E \rightarrow N$
$\quad | E+E$
$\quad | ExE$

$N \rightarrow 0$
$\quad | 1$
$\quad | 0N$
$\quad | 1N$

Gerar: 1 + 0

E

$\Rightarrow E+E$

$\Rightarrow E+N$

$\Rightarrow E+0$

$\Rightarrow N+0$

$\Rightarrow 1+0$

Gramáticas livre de contexto – Definição Formal

Mais alguns exemplos: Construa uma derivação e uma árvore de derivação para as palavras que a gramática puder gerar

A) £

S -> T
T -> £

B) aba

S -> aSa
S -> T => aTa
T -> b => aba (só símbolos terminais)

C) baa

A gramática G não pode gerar esta cadeia.

D) babab

S -> bSb
S -> aSa => baSab
S -> T => baTab
T -> b => babab (só símbolos terminais)

G:

$S \rightarrow aSa$
$\quad | bSb$
$\quad | T$

$T \rightarrow a$
$\quad | b$
$\quad | \varepsilon$

slide 16

Prática

Considere a gramática abaixo, com alfabeto terminal {(,),+,x,0,1} e símbolo de início **E**. Derive cada uma das cadeias dadas :

$E \rightarrow E+E$

| ExE

| (E)

| N

$N \rightarrow 0 \mid 1$

Gerar:

1+1

(1+0)x1

slide 17

BNF – Backus-Naur Form

- Forma de Backus-Naur (BNF) e Gramática Livre de Contexto
  - Método mais usado para descrever a sintaxe das linguagens de programação
- Extended BNF (Backus-Naur Form)
  - Aumenta readability e writability da BNF

18

BNF – Backus-Naur Form

- Gramáticas Livres de Contexto
  - Proposta por Noam Chomsky em meados da década de 50
  - Objetiva descrever a sintaxe de linguagens naturais
  - Define classe de linguagens chamada de linguagens livres de contexto

19

BNF – Backus-Naur Form

- Backus-Naur Form (1959)
  - Proposta por John Backus para descrever o Algol 58
  - Equivalente a gramáticas livre de contexto
  - Meta-linguagem capaz de descrever outras linguagens

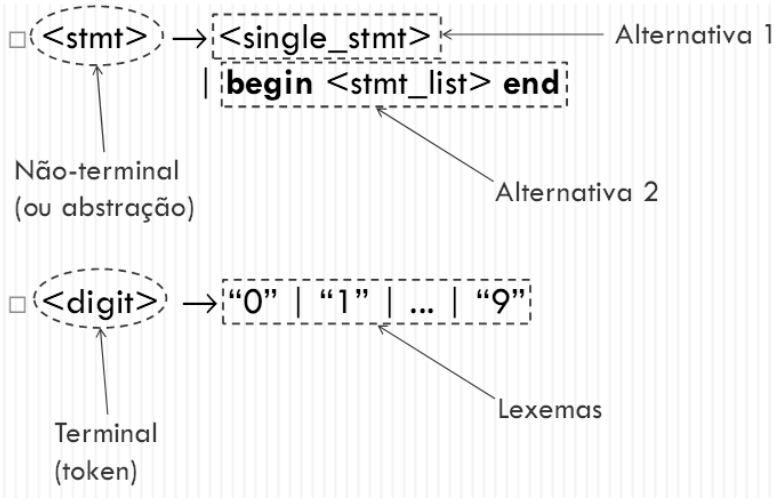
20

BNF – Backus-Naur Form

- Gramática: Descreve linguagem através de coleção não vazia de regras de produção
- **Uma regra de produção possui**
  - Lado esquerdo (LHS) -> lado direito (RHS)
  - LHS só pode ter UM símbolo terminal ou não-terminal, enquanto que RHS, em geral, tem combinações deles
  - O símbolo | em RHS significa representação alternativa

21

Exemplo de regra de produção



22

Exemplo 2 (Expressões)

- Derivando "(x+x)*x" a partir de E:

  E (símbolo inicial)
  ⇒ E*E
  ⇒ (E)*E
  ⇒ (E+E)*E
  ⇒ (E+E)*x
  ⇒ (E+x)*x
  ⇒ (x+x)*x

- Existem outras maneiras?

23

Exemplo 3

- Representação BNF do exemplo 2 (dado anteriormente)

```
<expression> ::= <expression> + <expression>
               | <expression> * <expression>
               | ( <expression> )
               | x
```

  – Não-terminais podem ter nomes que facilitam o entendimento da gramática

24

Exemplo 4 (Comandos)

- Comandos em uma linguagem de programação simples

```
<list-comando> ::= <comando> ;
                 | <comando> ; <list-
               comando>

    <comando> ::= <atrib>

<atrib> ::=  x  =  <expression>
```

(considerar <expression> do exemplo 3)

25

Descrevendo listas...

- Listas sintáticas são descritas usando recursão

  <ident_list> -> ident
  | ident, <ident_list>

Uma derivação é a aplicação de regras repetidas vezes, começando com um símbolo inicial e finalizando com uma sentença formada de símbolos terminais

26

Uma gramática para uma pequena lista…

EXAMPLE 3.1	A Grammar for a Small Language

<program> → **begin** <stmt_list> **end**
<stmt_list> → <stmt>
| <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
| <var> – <var>
| <var>

27

Uma gramática para atribuições simples…

EXAMPLE 3.2	A Grammar for Simple Assignment Statements

<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
| <id> * <expr>
| (<expr>)
| <id>

28

BNF – Backus-Naur Form

- Formatação alternativa para as produções de uma GLC
  - Muito usada em Compiladores
- Regras
  - Variáveis delimitadas por "<" e ">"
  - Terminais ou não têm delimitadores ou são delimitados por aspas
  - O sinal "→" é substituído por "**::=**"

29

Exercício

Considere a gramática livre de contexto em notação BNF abaixo. Assuma que o não-terminal inicial é <programa>.

```
<programa> ::= <lista_decl>
<lista_decl> ::= <declaracao> <lista_decl>
                 | <declaracao>
<declaracao> ::= <tipo> <nome> ";"
                | "func" <nome> "()" "{" <comando> "}"
<tipo> ::= "int" | "char"
<comando> ::= <nome> "=" <expr> ";"
                | "return" <expr> ";"
<expr> ::= <expr> "+" <expr>
           | <expr> "*" <expr>
           | <nome>
<nome> ::= "aux" | "temp" | "x"
```

Dê um exemplo de **palavra** que possa ser gerada pela gramática acima e que tenha ao menos um comando.

30

Derivações Mais a Esquerda/Direita

31

Definições

- Uma derivação mais a esquerda de uma palavra é uma seqüência de derivação em que as produções são aplicadas sempre no não-terminal mais à esquerda
- A derivação mais a direita é um conceito análogo...

32

Árvores de Derivação

- Uma mesma árvore de derivação pode ser gerada tanto por
  - uma **derivação mais à esquerda**

  quanto por
  - uma **derivação mais à direita**

Derivação Mais à Esquerda e Mais à Direita

Definição:

Uma derivação diz-se mais à esquerda se em cada etapa a variável mais a esquerda é trocada na forma sentencial.

Se em cada etapa a variável mais a direita é trocada, dizemos que a derivação é mais à direita.

Derivação Mais à Esquerda e Mais à Direita

Considere a gramática com produções **S→aAB, A→bBb, B→A|ε**

uma derivação mais à **esquerda** da cadeia **abbbb**:
S⇒a**A**B⇒a**bBb**B⇒a**bAb**B⇒a**bbBbb**B⇒a**bbbb**B⇒**ab bbb**

uma derivação mais à **direita**:
S⇒aA**B**⇒aA ⇒ab**B**b⇒ab**A**b⇒abb**B**bb⇒abbbb.

slide 35

Árvore de Derivação

Outra maneira de chegarmos a uma palavra da linguagem é construindo uma **árvore de derivação (ou árvore de sintaxe concreta). Este tipo de árvore tem as seguintes características:**

1. Seus nós são símbolos terminais ou não-terminais.
2. A sua raiz (representada no topo) é o símbolo inicial.
3. Um símbolo não-terminal N é um nó que possui filhos. Se N → a_1, a_2 ... a_n for uma produção, então um nó N pode aparecer com os filhos a_1, a_2, ... e a_n, nesta ordem.
4. Os símbolos terminais e a cadeia ε não têm filhos (são folhas da árvore).

slide 36

Árvore de Derivação

- Representação em árvore da seqüência de produções que derivou uma dada palavra
- Ajuda a visualizar a derivação da palavra
- Muito útil em compiladores

37

Árvore de Derivação

Uma árvore de derivação está intimamente ligada a uma derivação de uma palavra. Aliás, é fácil passar de uma derivação para a árvore ou da árvore para uma derivação, considerando uma mesma cadeia.

slide 38

Árvore de Derivação

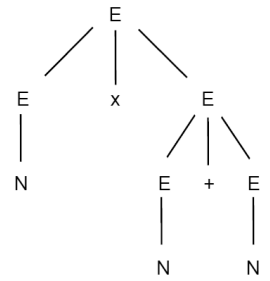
Exemplo: a construção em paralelo de uma derivação e sua árvore de derivação correspondente usando a gramática G2.

P_2:

$E \rightarrow N$
| E+E
| ExE

$N \rightarrow 0$
| 1
| 0N
| 1N

E -> ExE
E -> ExE+E



Vamos começar aplicando a produção E→ExE
No próximo passo, vamos aplicar a produção E → E+E na segunda ocorrência de E dessa cadeia.
Aplicando N em cada ocorrência de E da cadeia: ⇒ NxE+E

⇒ NxN+E

⇒ NxN+N

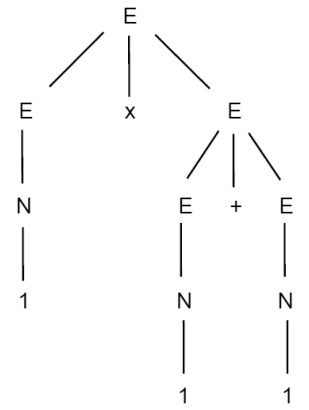
slide 39

Árvore de Derivação

Continuação...

Aplicando N -> 1 em cada ocorrência de E:

E
⇒ ExE
⇒ ExE+E
⇒ NxE+E
⇒ NxN+E
⇒ NxN+N
⇒ 1xN+N
⇒ 1x1+N
⇒ 1x1+1

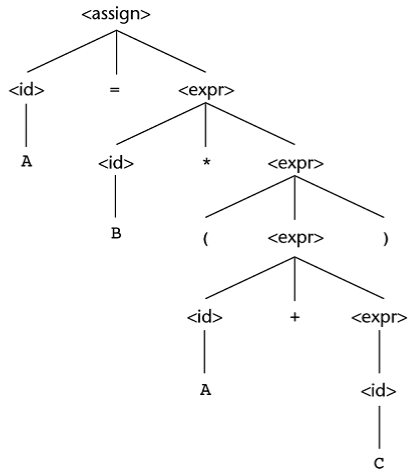


slide 40

Árvore de Análise

Figure 3.1

A parse tree for the simple statement
`A = B * (A + C)`



41

Árvore de Análise

EXAMPLE 3.2	A Grammar for Simple Assignment Statements
	<assign> → <id> = <expr> <id> → A | B | C <expr> → <id> + <expr> | <id> * <expr> | (<expr>) | <id>

Monte as árvores sintáticas para (se possível):

B = B * (B + (A * C))

C = B + (B * (A + C))

A = (B + C) * (A + C)

42

Ambigüidade em Gramáticas

Uma gramática é ambígua se e somente se ela gera uma sentença para a qual há duas ou mais árvores de análise distintas

Ambigüidade em Gramáticas

Uma gramática ambígua para instruções de atribuição simples

EXAMPLE 3.3 An Ambiguous Grammar for Simple Assignment Statements

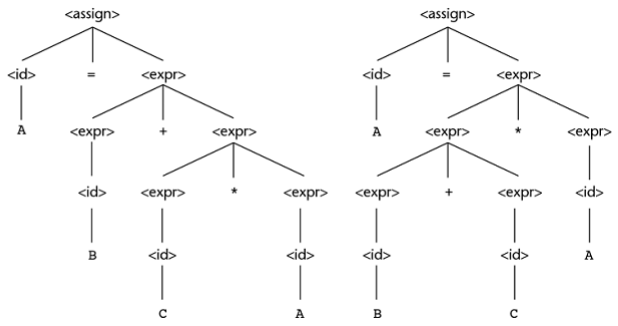
```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <expr>
       | <expr> * <expr>
       | ( <expr> )
       | <id>
```

Ambigüidade em Gramáticas

Duas árvores de análise para a mesma sentença

Figure 3.2

Two distinct parse trees for the same sentence,
`A = B + C * A`



Árvore Ambígua

Gramática que possui duas (ou mais) árvores de derivação distintas para alguma palavra.

Não exige que existam duas árvores de derivação para *todas as palavras – basta existir duas árvores de* derivação para ***uma palavra*** *qualquer da linguagem.*

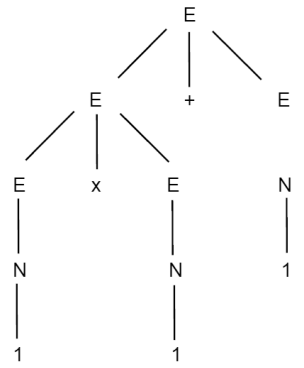
Árvore Ambígua

Mesmo exemplo

P_2:

$E \rightarrow N$
$\quad | \; E+E$
$\quad | \; ExE$

$N \rightarrow 0$
$\quad | \; 1$
$\quad | \; 0N$
$\quad | \; 1N$

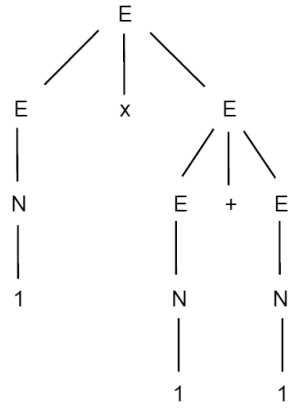


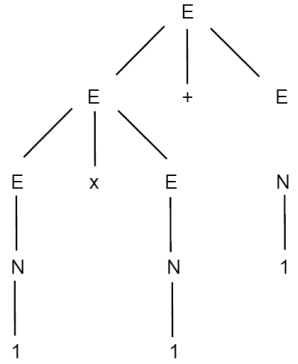
Vamos começar analisando a palavra **1x1+1**. Já criamos uma árvore de derivação para ela há pouco, será que é possível criarmos outra árvore para **1x1+1**?

A árvore de derivação que criamos antes para a palavra **1x1+1** começa pela produção **E → ExE**. É possível gerar a mesma palavra começando pela produção **E → E+E**?

slide 47

Árvore Ambígua





Árvores diferentes que geram a mesma palavra: **1x1+1**

NOTA: Não existe um algoritmo para identificar se uma gramática livre de contexto qualquer é ambígua. Por esse motivo, identificar ambigüidades em gramáticas livres de contexto não é uma tarefa muito fácil.

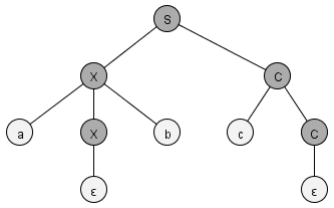
slide 48

Árvores Ambíguas - Prática

Prove que a gramática H definida abaixo é ambígua.

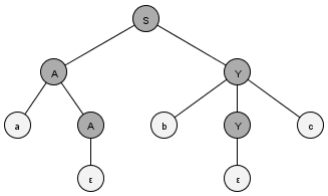
H:

$S \rightarrow XC$
| AY
$X \rightarrow aXb \mid \varepsilon$
$Y \rightarrow bYc \mid \varepsilon$
$C \rightarrow cC \mid \varepsilon$
$A \rightarrow aA \mid \varepsilon$



S => XC => aXbC => abC => abcC => abc
(corresponde à árvore 1)

S => AY => aAY => aY =>abYc => abc
(corresponde à árvore 2)



slide 49

Árvore Ambígua

EXAMPLE 3.3	An Ambiguous Grammar for Simple Assignment Statements
	<assign> → <id> = <expr> <id> → A | B | C <expr> → <expr> + <expr> | <expr> * <expr> | (<expr>) | <id>

Os seguintes exemplos conseguem demonstrar a ambigüidade?

A = (B+C)*(A+C)

B = (B*B) + (A*C)

C = B+B * A+C

slide 50

Gramática dos Números

51

Gramática dos Números

Um número pode ser um número inteiro ou com casas decimais:

<numero> ::= <num-int> | <num-dec>

Um número inteiro tem um sinal seguido de uma seqüência de dígitos:

<num-int> ::= <sinal> <seq-digitos>

52

Gramática dos Números

O sinal pode ser de positivo, de negativo ou pode ser omitido:

<sinal> ::= “+” | “-” | “”

Uma seqüência de dígitos tem um dígito apenas ou tem um dígito seguido de outra seqüência:

<seq-digitos> ::= <digito>
| <digito> <seq-digitos>
<digito> ::= “0” | “1” | “2” | ... | “9”

53

Gramática dos Números

Um número com casas decimais tem um sinal seguido de uma parte inteira, de um ponto, e da parte decimal:

<num-dec> ::= <sinal> <seq-digitos> <parte-dec>

A parte decimal é uma seqüências de dígitos iniciada com ponto:

<parte-dec> ::= “.” <seq-digitos>

54

Gramática dos Números

```
<numero> ::= <num-int> | <num-dec>

<num-int> ::= <sinal> <seq-digitos>

<num-dec> ::= <sinal> <seq-digitos> <parte-dec>
<parte-dec> ::= "." <seq-digitos>

<sinal>      ::= "+" | "-" | ""
<seq-digitos> ::= <digito>
                | <digito> <seq-digitos>

<digito>     ::= "0" | "1" | "2" | ... | "9"
```

55

Exercício

- Na gramática dada, como é possível gerar (derivar) "+12" ?
  - Mostrar derivação mais à esquerda e mais à direita
  - Mostrar árvore de derivação

56

Exercício

- Assumindo que a parte decimal possa ser vazia, ou seja:

  <parte-dec> ::= "." **<seq-digitos> |** ""

- E agora, como é possível derivar "+12" ? A gramática é ambígua?
  - Mostrar duas árvores de derivação...

57

Importância das Gramáticas Livres de Contexto

CURIOSIDADES

Esse modelo foi proposto para tentar representar a linguagem natural (a língua portuguesa, por exemplo), onde classes de palavras são usadas para formar certas frases, segundo certas regras, e frases menores são usadas para construir outras maiores, num processo recursivo.

Posteriormente, as gramáticas passaram a ser usadas para descrever linguagens computacionais, em especial, as linguagens de programação. No caso, as gramáticas são usadas para descrever o formato dos programas válidos nas linguagens de programação.

Todas as linguagens modernas são definidas a partir de alguma gramática livre de contexto. Com base em uma gramática livre de contexto de uma linguagem computacional, existem várias técnicas que podem ser usadas para construir um compilador para a linguagem. Um importante requisito, no entanto, é que a gramática não seja ambígua, pois deixaria o compilador "em dúvida" na hora de montar uma árvore de derivação.

A árvore, por sua vez, também é muito importante no processo de compilação.

slide 58

Linguagens Formais

UNICAP

Eduardo Araújo Oliveira
http://sites.google.com/site/eaoufpe

UNIVERSIDADE CATÓLICA DE PERNAMBUCO