

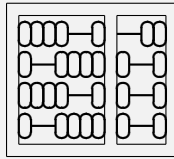
Implementação de Linguagens de Programação

MO403 / MC900

www.ic.unicamp.br/~mo403

Tomasz Kowaltowski

Instituto de Computação
Universidade Estadual de Campinas



1º semestre de 2007

Copyright © 2007 Tomasz Kowaltowski

Instituto de Computação
Universidade Estadual de Campinas

Parte do material contido nestas transparências foi copiada ou adaptada do livro do autor *Implementação de Linguagens de Programação*, Editora Guanabara Dois, 1983.

Estas transparências somente podem ser copiadas para uso pessoal dos alunos das disciplinas oferecidas pelo Instituto de Computação da UNICAMP.

Introdução

Objetivos

- ▶ Noções básicas sobre o processo de compilação
- ▶ Implementação de um compilador completo para uma linguagem exemplo
- ▶ Utilização de algumas ferramentas
- ▶ Integração num único projeto de vários conhecimentos da Computação
- ▶ Compreensão mais profunda de alguns mecanismos lingüísticos
- ▶ Aplicações das técnicas e ferramentas em outros contextos

Pré-requisitos

- ▶ Bons conhecimentos e experiência de programação
- ▶ Conceitos de linguagens de programação
- ▶ Conceitos de organização de computadores
- ▶ Estruturas de dados (pilhas, árvores, tabelas, ...)
- ▶ Técnicas de programação (recursão, modularização, ...)
- ▶ Noções de gramáticas e linguagens livres de contexto (desejável)
- ▶ Noções de expressões regulares e linguagens regulares (desejável)

Avaliação

- ▶ Provas P_1 e P_2 : média $P = (4P_1 + 6P_2)/10$
- ▶ Projeto: implementação do compilador (nota C)
- ▶ Aproveitamento A :

$$A = \begin{cases} (P + C)/2 & \text{se } P \geq 5 \text{ e } C \geq 5 \\ \min(P, C) & \text{caso contrário.} \end{cases}$$

- ▶ Nota final para MO403: aproveitamento A será transformado num conceito (A, B, C, \dots)
- ▶ Nota final F para MC900:

$$F = \begin{cases} (A + E)/2 & \text{se o aluno fez o exame final} \\ A & \text{caso contrário} \end{cases}$$

onde E é a nota obtida no exame

Datas

- ▶ Prova P_1 : 7 de maio (segunda-feira)
- ▶ Prova P_2 : 25 de junho (segunda-feira)
- ▶ Exame final: 11 de julho (segunda-feira; somente MC900)
- ▶ Entrega final do projeto: 27 de junho.

Regras

- ▶ O projeto é **estritamente individual**.
- ▶ Não haverá provas substitutivas (eventualmente, exame final, se houver).
- ▶ Qualquer tentativa de fraude nas provas ou no projeto implicará em aproveitamento zero no semestre para todos os envolvidos, sem prejuízo de outras sanções.
- ▶ As transgressões às regras de uso dos sistemas computacionais do Instituto de Computação ou de outras unidades da UNICAMP que possam prejudicar outros usuários ou sistemas, dentro ou fora da Universidade, implicarão em aproveitamento zero no semestre para todos os envolvidos, sem prejuízo de outras sanções.

Programa

1. Introdução: aspectos básicos de compilação
2. Linguagem exemplo
3. Conceitos básicos de gramáticas livres de contexto e notação BNF
4. Análise sintática
5. Análise sintática descendente
6. Análise sintática ascendente
7. Análise léxica: *ad hoc* e expressões regulares
8. Ferramentas
9. Sistema de execução para a linguagem exemplo
10. Organização do compilador, análise semântica, tabelas de símbolos e geração de código
11. Tópicos complementares






Bibliografia

-  A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman
Compilers: Principles, Techniques, and Tools (2nd. ed.)
Addison Wesley, 2006
-  A. W. Appel and M. Ginsburg
Modern Compiler Implementation in C (new ed.)
Cambridge University Press, 2004
-  A. W. Appel and J. Palsberg
Modern Compiler Implementation in Java (2nd. ed.)
Cambridge University Press, 2002
-  J. R. Levine, T. Mason and D. Brown
lex & yacc (2nd. ed.)
O'Reilly, 1992

Bibliografia (cont.)

-  A. V. Aho, R. Sethi and J. D. Ullman
Compilers — Principles, Techniques, and Tools
Addison-Wesley, 1986
-  A. T. Schreiner and H. G. Friedman, Jr.
Introduction to Compiler Construction with Unix
Prentice Hall, 1985
-  Tomasz Kowaltowski.
Implementação de Linguagens de Programação
Editora Guanabara Dois, 1983
-  M. E. Lesk and E. Schmidt
Lex — A Lexical Analyzer
Bell Laboratories, 1978
-  S. C. Johnson
Yacc: Yet Another Compiler-Compiler
Bell Laboratories, 1978

Bibliografia (cont.)

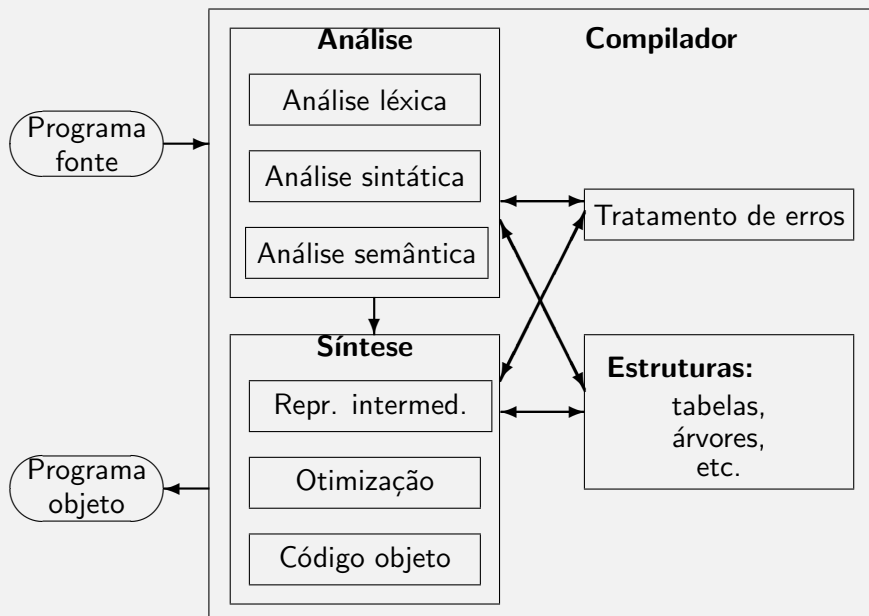
-  A. V. Aho and J. D. Ullman
The Theory of Parsing, Translation and Compiling, vol. 1: Parsing
Prentice-Hall, 1972
-  A. V. Aho and J. D. Ullman
The Theory of Parsing, Translation and Compiling, vol. 2: Compiling
Prentice-Hall, 1973
-  C. Donnelly and R. Stallman
Bison
Free Software Foundation, Inc, 2006
<http://www.gnu.org/software/bison/manual>
-  Sourceforge *Flex: a fast lexical analyzer generator*
<http://flex.sourceforge.net/manual>
-  Tom Niemann *A Compact Guide to Lex & Yacc*
<http://www.epaperpress.com/lexandyacc>

Material disponível

- ▶ Texto: T. Kowaltowski, *Implementação de Linguagens de Programação* (cópia reprográfica autorizada para os alunos da disciplina).
- ▶ Cópias das transparências
- ▶ Exemplo de utilização das ferramentas na implementação de uma micro-linguagem PL.
- ▶ Interpretador da máquina virtual (MEPA) para a linguagem exemplo (executa sob o sistema Linux).

Visão geral

Esquema geral de um compilador



Análise léxica (*scanning*)

- ▶ **Entrada** (texto):

```
while i<n do
  s := s+x
```

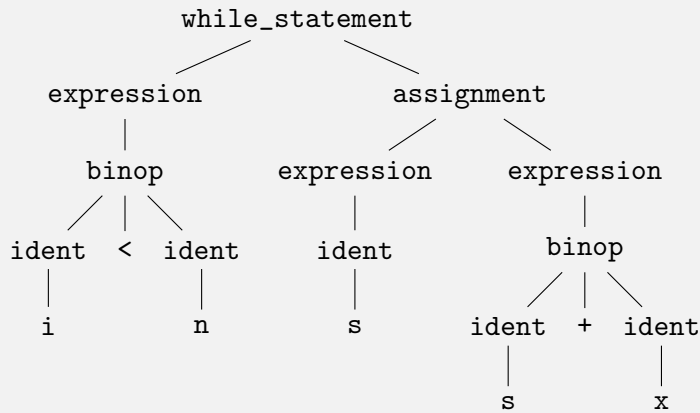
- ▶ **Saída** (seqüência de átomos – *tokens*):

- símbolo *while* (palavra reservada)
- identificador *i*
- símbolo *menor*
- identificador *n*
- símbolo *do* (palavra reservada)
- identificador *s*
- símbolo *atribui*
- identificador *s*
- símbolo *mais*
- identificador *x*

- ▶ Análise léxica trata espaços, quebras de linha, comentários. Os átomos são representados por elementos de enumerações (inteiros).

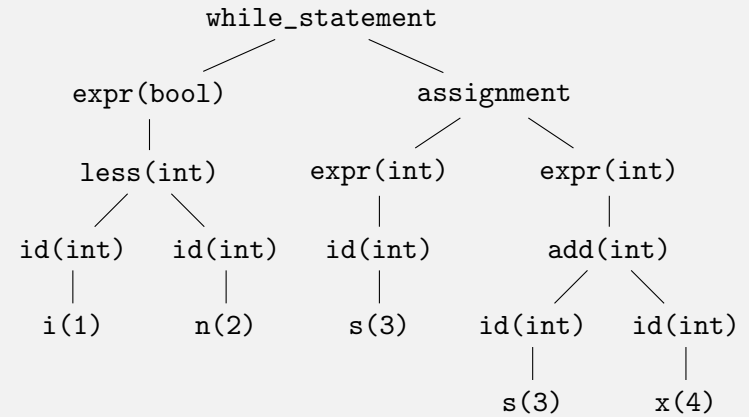
Análise sintática (*parsing*)

- ▶ **Entrada:** seqüência de átomos
- ▶ **Saída:** árvore sintática ou de derivação (simplificada)



Análise semântica

- ▶ **Entrada:** árvore sintática
- ▶ **Saída:** árvore sintática *decorada* e/ou tabelas



Análise semântica (cont.)

- ▶ Nome em parte impróprio – problemas de linguagens livres de contexto.
- ▶ Verificação das regras da linguagem:
 - ▶ determinação de tipos
 - ▶ equivalência e consistência de tipos
 - ▶ significado dos operadores (polimorfismo)
 - ▶ escopos
 - ▶ ...

Geração da representação intermediária

- ▶ **Entrada:** árvore sintática *decorada* e tabelas
- ▶ **Saída:** representação intermediária

L1:

```
t1 := i
t2 := n
t3 := t1 < t2
t4 := not t3
if t4 goto L2
t5 := s
t6 := x
t7 := t5 + t6
s := t7
goto L1
```

L2:

- ▶ Representação interna pode ser diferente (lista ou outra estrutura)

Otimização

- ▶ **Entrada:** representação intermediária
- ▶ **Saída:** representação intermediária otimizada

```
L1:
    t1 := i=>n
    if t1 goto L2
    t2 := s+x
    s := t2
    goto L1
L2:
```

- ▶ Na realidade, a otimização pode ser bem mais significativa.

Geração do código objeto

- ▶ **Entrada:** representação intermediária otimizada
- ▶ **Saída:** programa objeto (linguagem de máquina ou de montagem hipotética)

```
L1:
    LOAD 1,end_i
    SUBT 1,end_n
    JNPS 1,L2
    LOAD 1,end_s
    ADD 1,end_x
    STO 1,end_s
    JMP L1
L2:
```

Observações

- ▶ As fases do compilador típico não são necessariamente seqüenciais.
- ▶ Algumas fases podem não existir; por exemplo, otimização de código.
- ▶ Algumas estruturas podem existir apenas conceitualmente mas não explicitamente; por exemplo, árvores de derivação.
- ▶ O código objeto final pode ser gerado em linguagem de máquina ou então em linguagem de montagem (*assembly language*); neste último caso será usado ainda um montador.
- ▶ Dependendo da maneira de entrelaçar as várias fases, existem os compiladores de um passo ou de múltiplos passos.
- ▶ Existem ainda outros aspectos pragmáticos do processo de compilação, dependentes do ambiente de programação e de execução: bibliotecas, chamadas de sistema, gerenciamento de memória, etc.

Linguagem exemplo: subconjunto de Pascal

- ▶ Sintaxe simples e regular
- ▶ Conceitos claros relativos a mecanismos de passagem de parâmetros
- ▶ Escopos encaixados (rotinas e blocos)
- ▶ Características incluídas:
 - ▶ tipos básicos: *integer* e *boolean*
 - ▶ estruturas: **array** (opcional)
 - ▶ declarações de variáveis
 - ▶ declarações de tipos (opcional)
 - ▶ blocos, procedimentos e funções; recursão
 - ▶ parâmetros por valor, por referência
 - ▶ rotinas como parâmetros (opcional)
 - ▶ rótulos e desvios
 - ▶ comandos condicionais “**if ... then ...**” e “**if ... then ... else ...**”
 - ▶ comando repetitivo “**while ... do ...**”
 - ▶ operações aritméticas sobre inteiros
 - ▶ operações booleanas
 - ▶ comentários “{ ... }” ou “(* ... *)”

Gramáticas e linguagens

Notação e terminologia

- ▶ **Alfabeto** ou **vocabulário**: conjunto finito e não vazio de símbolos; exemplos:
 - ▶ $\Sigma_1 = \{a, b\}$ (conjunto de duas letras)
 - ▶ $\Sigma_2 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (conjunto de dígitos decimais)
 - ▶ $\Sigma_3 = \{\text{if, then, else, while, id, :=, <, >, =, ...}\}$ (conjunto de símbolos de uma linguagem de programação: cada símbolo é considerado um *átomo*)
- ▶ Se $\alpha = \sigma_1\sigma_2 \dots \sigma_n$, com $\sigma_i \in \Sigma$, então α é uma *cadeia* sobre o alfabeto Σ , de comprimento n ; exemplo: 'abaa' é uma cadeia sobre Σ_1 , de comprimento 4.
- ▶ O símbolo ϵ (no texto λ) denota a cadeia vazia de comprimento zero.
- ▶ Dada uma cadeia α :
 - ▶ $\alpha^0 = \epsilon$
 - ▶ $\alpha^n = \alpha^{n-1}\alpha$ para $n > 0$ (concatenação n vezes)
- ▶ Dado um alfabeto Σ :
 - ▶ Σ^* é o conjunto de todas as cadeias finitas sobre Σ
 - ▶ Σ^+ é o conjunto de todas as cadeias finitas sobre Σ , exceto a cadeia vazia.

Notação e terminologia (cont.)

- ▶ Uma *linguagem* sobre Σ é um subconjunto de Σ^* .
- ▶ Dadas duas linguagens L_1 e L_2 sobre um alfabeto Σ , a *concatenação* (ou o *produto*) é dado por $L_1L_2 = \{\alpha\beta \mid \alpha \in L_1 \text{ e } \beta \in L_2\}$
- ▶ Dada uma linguagem L :
 - ▶ $L^0 = \{\epsilon\}$
 - ▶ $L^n = L^{n-1}L$ para $n > 0$
 - ▶ $L^* = \bigcup_{n \geq 0} L^n$
 - ▶ $L^+ = \bigcup_{n > 0} L^n$
- ▶ Dada uma cadeia α , a notação será estendida para:
 - ▶ $\alpha^* = \{\alpha\}^*$
 - ▶ $\alpha^+ = \{\alpha\}^+$

Gramáticas livres de contexto

- ▶ Exemplo: gramática de expressões simples

$$E \leftarrow a$$

$$E \leftarrow b$$

$$E \leftarrow E + E$$

$$E \leftarrow E * E$$

$$E \leftarrow (E)$$

ou sob forma abreviada: $E \leftarrow a \mid b \mid E + E \mid E * E \mid (E)$

- ▶ Terminologia:

- ▶ vocabulário terminal T : $\{a, b, +, *, (,)\}$
- ▶ vocabulário não-terminal N : $\{E\}$
- ▶ vocabulário: $V = T \cup N$
- ▶ símbolo inicial S (raiz): E
- ▶ meta-símbolos: \leftarrow, \mid
- ▶ produção (exemplo): $E \leftarrow E * E$
- ▶ gramática livre de contexto: $G = (T, N, P, S)$

Derivações

► Exemplo 1: $a + b$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + b$$

$$E \Rightarrow E + E \Rightarrow E + b \Rightarrow a + b$$

► Exemplo 2: $a + b * a$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + b * E \Rightarrow a + b * a$$

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * a \Rightarrow E + b * a \Rightarrow a + b * a$$

$$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + a * E \Rightarrow a + b * E \Rightarrow a + b * a$$

► Exemplo 3: $a + b * a$ (não equivalente ao exemplo 2)

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + b * E \Rightarrow a + b * a$$

$$E \Rightarrow E * E \Rightarrow E * a \Rightarrow E + E * a \Rightarrow E + b * a \Rightarrow a + b * a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow E + b * E \Rightarrow a + b * E \Rightarrow a + b * a$$

► Notação:

- Deriva em zero ou mais passos: $X \xRightarrow{*} \alpha$ (ex: $E \xRightarrow{*} E$, $E \xRightarrow{*} a + b$)
- Deriva em mais de zero passos (não trivialmente): $X \xRightarrow{+} \alpha$ (ex: $E \xRightarrow{+} a + b$)

Derivações canônicas

► Derivações esquerdas:

$$E \xRightarrow{e} E + E \xRightarrow{e} a + E \xRightarrow{e} a + b$$

$$E \xRightarrow{e} E + E \xRightarrow{e} a + E \xRightarrow{e} a + E * E \xRightarrow{e} a + b * E \xRightarrow{e} a + b * a$$

$$E \xRightarrow{e} E * E \xRightarrow{e} E + E * E \xRightarrow{e} a + E * E \xRightarrow{e} a + b * E \xRightarrow{e} a + b * a$$

► Derivações direitas:

$$E \xRightarrow{d} E + E \xRightarrow{d} E + b \xRightarrow{d} a + b$$

$$E \xRightarrow{d} E + E \xRightarrow{d} E + E * E \xRightarrow{d} E + E * a \xRightarrow{d} E + b * a \xRightarrow{d} a + b * a$$

$$E \xRightarrow{d} E * E \xRightarrow{d} E + E * E \xRightarrow{d} E + b * E \xRightarrow{d} a + b * E \xRightarrow{d} a + b * a$$

- Observar que $a + b$ tem uma única derivação de cada tipo, enquanto que $a + b * a$ tem mais de uma.

Terminologia

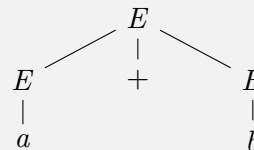
- Se $S \xRightarrow{*} \alpha$ então α é uma *forma sentencial* de G
- Se α é uma *forma sentencial* e $\alpha \in T^*$, então α é uma *sentença* de G (T^* é o conjunto de todas as cadeias finitas sobre T , inclusive a cadeia vazia ϵ)
- A linguagem $L(G)$ gerada por uma gramática G é o conjunto das sentenças de G

Árvores sintáticas ou de derivação

► Exemplo 1: $a + b$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + b$$

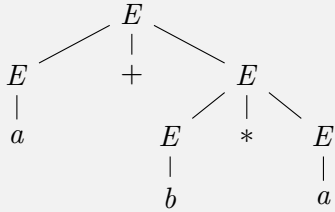
$$E \Rightarrow E + E \Rightarrow E + b \Rightarrow a + b$$



Árvores sintáticas ou de derivação (cont.)

▶ Exemplo 2: $a + b * a$

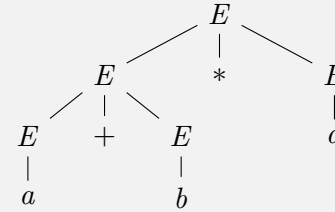
$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + b * E \Rightarrow a + b * a$
 $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * a \Rightarrow E + b * a \Rightarrow a + b * a$
 $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + a * E \Rightarrow a + b * E \Rightarrow a + b * a$



Árvores sintáticas ou de derivação (cont.)

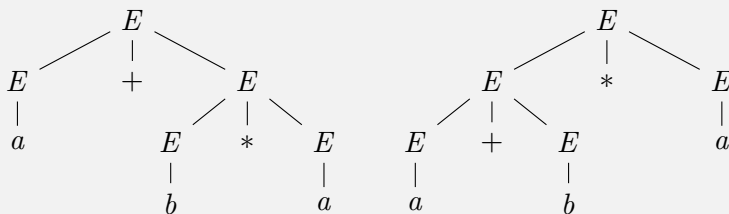
▶ Exemplo 3: $a + b * a$

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + b * E \Rightarrow a + b * a$
 $E \Rightarrow E * E \Rightarrow E * a \Rightarrow E + E * a \Rightarrow E + b * a \Rightarrow a + b * a$
 $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow E + b * E \Rightarrow a + b * E \Rightarrow a + b * a$



Ambigüidade

▶ Exemplo: $a + b * a$



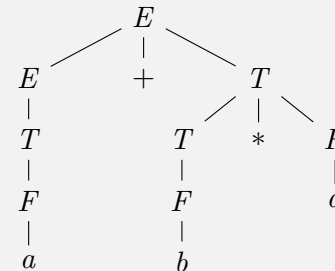
- ▶ Uma gramática G é ambígua se existe uma sentença de G que tem mais de uma árvore de derivação.
- ▶ (Outra caracterização) Uma gramática G é ambígua se existe uma sentença de G que tem mais de uma derivação esquerda (ou mais de uma derivação direita).
- ▶ Exemplo de linguagem inerentemente ambígua:
 $\{a^i b^j c^k \mid i, j, k > 0, i = j \text{ ou } j = k\}$.
- ▶ A ambigüidade de uma gramática é uma propriedade indecidível.

Gramática não ambígua de expressões simples

- ▶ A nova gramática impõe prioridades e associatividades tradicionais dos operadores (expressões, termos e fatores):

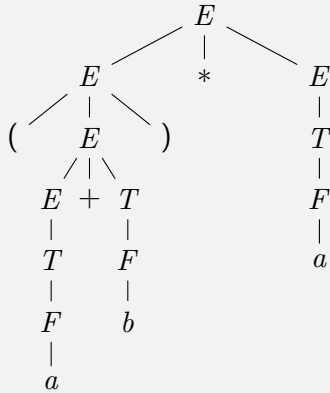
$$\begin{aligned}
 E &\leftarrow E + T \mid T \\
 T &\leftarrow T * F \mid F \\
 F &\leftarrow a \mid b \mid (E)
 \end{aligned}$$

- ▶ Exemplo: $a + b * a$ (árvore única mas mais complexa)



Gramática não ambígua de expressões simples (cont.)

- ▶ Outro exemplo: $(a + b) * a$



$E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + T) * E \Rightarrow (T + T) * E \Rightarrow (F + T) * E \Rightarrow (a + T) * E$
 $\Rightarrow (a + F) * E \Rightarrow (a + T) * E \Rightarrow (a + b) * E \Rightarrow (a + b) * T \Rightarrow (a + b) * F \Rightarrow (a + b) * a$

Exemplo clássico de ambigüidade

- ▶ Considere o trecho de gramática:

$$S \leftarrow \text{if } E \text{ then } S$$

$$S \leftarrow \text{if } E \text{ then } S \text{ else } S$$

onde os símbolos não terminais S e E correspondem a *comando* e *expressão*.

- ▶ Um comando da forma: `if A then if B then C1 else C2` poderia ter duas interpretações:

```

if A
  then if B then C1 else C2
    
```

```

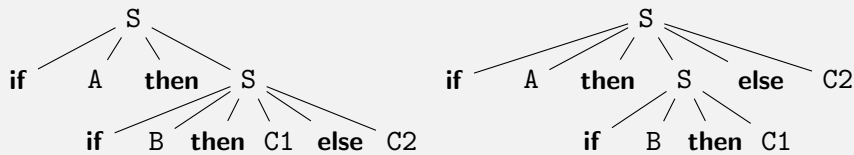
ou
if A
  then if B then C1
  else C2
    
```

isto é, estas produções permitem associar a alternativa “else C2” tanto com o primeiro quanto com o segundo “if”.

Exemplo clássico de ambigüidade (cont.)

- ▶ As árvores de derivação:

`if A then if B then C1 else C2`



- ▶ Normalmente, adota-se a primeira interpretação: associar “else” com o “if” mais próximo.
- ▶ É possível escrever um conjunto de produções que elimina a ambigüidade. Na prática, isto não é feito por complicar a análise. Uma outra solução será vista mais adiante.

Função FIRST

- ▶ Dada uma gramática G e um símbolo $X \in V$, a função $FIRST(X)$ (relação Ψ_P no texto) é definida como o conjunto de símbolos terminais que aparecem no início de alguma cadeia derivada de X .
- ▶ Exemplo de aplicação de um algoritmo iterativo à gramática não ambígua de expressões:

$$E \leftarrow E + T \mid T \quad T \leftarrow T * F \mid F \quad F \leftarrow a \mid b \mid (E)$$

	FIRST					
E	E	T	F	a	b	$($
T	T	F	a	b	$($	
F	a	b	$($			

- ▶ No caso dos símbolos terminais, somente eles mesmos podem ser derivados.
- ▶ O algoritmo é ligeiramente mais complicado quando existem produções da forma $X \leftarrow \epsilon$.
- ▶ A função FIRST pode ser estendida de maneira natural para cadeias de caracteres.

Função FOLLOW

- ▶ Dada uma gramática G e um não-terminal X , a função $FOLLOW(X)$ é definida como o conjunto de símbolos terminais que seguem o não-terminal X em alguma forma sentencial de G , ou seja, símbolos terminais τ tais que: $S \xRightarrow{*} \alpha X \tau \beta$ para algum α e β .
- ▶ Exemplo de aplicação de um algoritmo iterativo à mesma gramática ($\#$ denota o fim da cadeia):

$$E \leftarrow E + T \mid T \quad T \leftarrow T * F \mid F \quad F \leftarrow a \mid b \mid (E)$$

	FOLLOW
E	$\# +)$
T	$* \# +)$
F	$* \# +)$

- ▶ O algoritmo é ligeiramente mais complicado quando existem produções da forma $X \leftarrow \epsilon$. Neste exemplo, não aparece o caso de dois não-terminais consecutivos da form $\dots XY \dots$ quando seria usada a função $FIRST(Y)$.

Linguagens regulares

Linguagens regulares

- ▶ Caso particular de linguagens livres de contexto
- ▶ Podem ser descritas por gramáticas cujas produções têm uma forma particular; exemplo:

$$\begin{array}{l} S \leftarrow a \\ S \leftarrow b \\ S \leftarrow aS \\ S \leftarrow bS \end{array} \quad \text{ou} \quad \begin{array}{l} S \leftarrow a \\ S \leftarrow b \\ S \leftarrow Sa \\ S \leftarrow Sb \end{array}$$

- ▶ Todas as ocorrências de símbolos não terminais estão no fim (ou início) dos lados direitos das produções.
- ▶ As duas gramáticas descrevem a linguagem que contém todas as cadeias finitas e não vazias sobre o alfabeto $\{a, b\}$; exemplo, usando a primeira gramática:

$$S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abbaS \Rightarrow abbab$$

- ▶ Estas gramáticas são denominadas *gramáticas regulares* e as linguagens que podem ser descritas por elas são *linguagens regulares*.

Expressões regulares

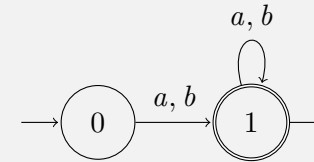
- ▶ Expressões regulares constituem um mecanismo mais simples para descrever linguagens regulares.
- ▶ Dado um alfabeto Σ :
 - ▶ para todo $\sigma \in \Sigma$, " σ " é uma expressão regular que denota a linguagem $\{\sigma\}$
 - ▶ se " α " e " β " são duas expressões regulares denotando as linguagens L_α e L_β , então " $\alpha\beta$ " é uma expressão regular que denota a linguagem produto $L_\alpha L_\beta$
 - ▶ se " α " e " β " são duas expressões regulares denotando as linguagens L_α e L_β , então " $\alpha|\beta$ " é uma expressão regular que denota a linguagem união $L_\alpha \cup L_\beta$
 - ▶ se " α " é uma expressão regular que denota a linguagem L_α então " α^* " é uma expressão regular que denota a linguagem L_α^*
 - ▶ se " α " é uma expressão regular que denota a linguagem L_α então " (α) " é uma expressão regular que denota a mesma linguagem

Exemplos de expressões regulares

- ▶ Exemplo 1: $(a|b)(a|b)^*$ descreve a mesma linguagem dos exemplos anteriores.
- ▶ Exemplo 2: $a(aa|ab|ba|bb)^*b$ descreve a linguagem de cadeias que começam com a , terminam com b , e entre estes caracteres aparece uma seqüência arbitrária de comprimento par (possivelmente vazia) de letras a ou b .
- ▶ Exemplo 3: $(a|b|c|d|\dots|z)(a|b|c|d|\dots|z|0|1|\dots|9)^*$ descreve a forma de identificadores de muitas linguagens de programação (uma letra seguida de letras e/ou dígitos)
- ▶ Estes exemplos pressupõem que a operação “*” tem precedência sobre o produto, e o produto tem precedência sobre a operação “|”.

Autômatos finitos

- ▶ Pode-se demonstrar que expressões regulares descrevem a mesma classe de linguagens aceita por *autômatos finitos*.
- ▶ Exemplo 1: o autômato seguinte aceita a linguagem descrita por $(a|b)(a|b)^*$



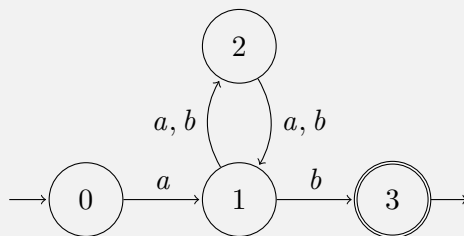
Por convenção, o estado inicial tem o rótulo 0; círculo duplo indica estado de aceitação (final).

- ▶ Autômatos finitos podem ser facilmente transformados em programas:

```
L0: if inchar in {a,b} goto L1
    reject()
L1: advance()
    if inchar in {a,b} goto L1
    accept()
```

Autômatos finitos (cont.)

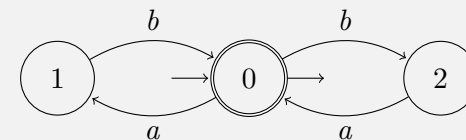
- ▶ Exemplo 2: $a(aa|ab|ba|bb)^*b$



- ▶ Deve-se observar que este é um exemplo de autômato não determinístico: no estado 1, quando o caractere de entrada é b , ele pode passar ao estado 2 ou ao estado 3.
- ▶ Demonstra-se que todo autômato não determinístico pode ser transformado num autômato determinístico, possivelmente com um número muito maior de estados.
- ▶ Transições ausentes indicam rejeição.

Autômatos finitos (cont.)

- ▶ Exemplo 3: $(ab|ba)^*$



- ▶ Nota-se que este autômato é determinístico.
- ▶ Exemplo de linguagem livre de contexto que não é regular:
 $\{a^n b^n | n \geq 0\}$
Esta linguagem tem uma estrutura de linguagem de parênteses simplificada.