Construção de Compiladores

Parte 1

Introdução
Linguagens e Gramáticas

F.A. Vanini

IC – Unicamp

Klais Soluções

Motivação

- Porque “compiladores” ?
  - São ferramentas fundamentais no processo de desenvolvimento de software e sua compreensão contribui para um uso melhor.
  - Reunem conhecimentos de várias áreas (linguagens formais, estruturas de dados, algoritmos, arquitetura de computadores) num único produto.
  - Oferecem excelentes oportunidades de aplicação de conceitos de engenharia de software como pro exemplo orientação a objetos, design patterns.

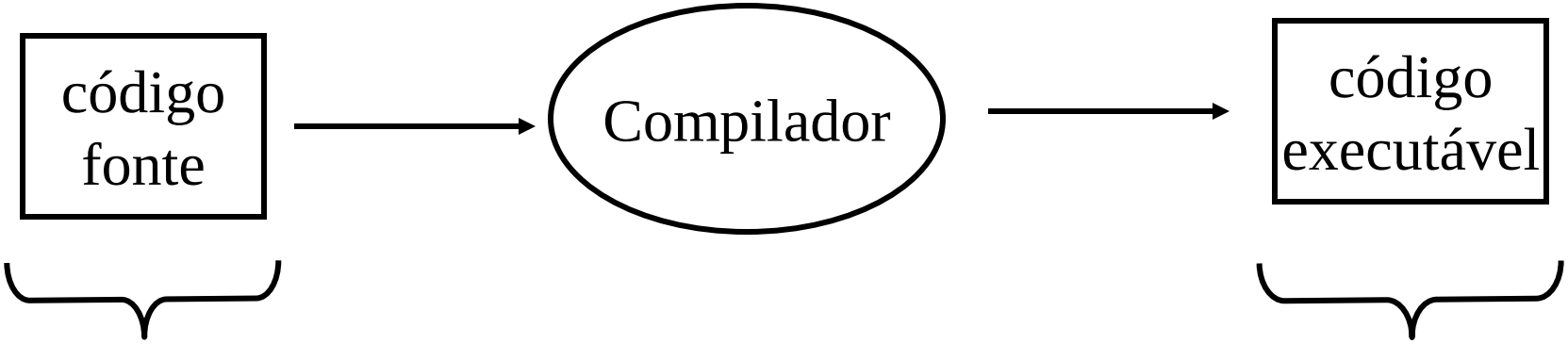
Conteúdo

- O processo de compilação
- Descrição de linguagens
- Análise sintática
- Análise semântica
- Organização de programas em tempo de execução
- Estrutura geral de um compilador
- Geração de código

Bibliografia

- T. Kowaltowski, Implementação de Linguagens de Programação, Guanabara Dois, 1983
- A. Aho, R. Sethi, J. D. Ullman, Compilers - Principles, Techniques and Tools, Addison-Wesley, 1986 (existe também a edição em português: Compiladores – Princípios, Técnicas e Ferramentas, LTC, 1995)
- A. W. Appel, Modern Compiler Implementation in Java, Cambridge University Press, 1997
- N. Wirth, Compiler Construction, Addison-Wesley, 1996

O processo de compilação



linguagem fonte (java, C++, vb, etc)

linguagem de máquina

O processo de compilação

A “compilação” de um programa tipicamente envolve o seguinte:

- Análise léxica
- Análise Sintática
- Análise Semântica
- Tradução

Análise Léxica

A análise léxica é responsável por

- Ler o texto fonte, caracter a caracter
- Identificar os “elementos léxicos” da linguagem: identificadores, palavras reservadas, constantes, operadores
- Ignorar comentários, brancos, tabs
- Processar includes (se for o caso)

Análise Sintática

A análise sintática é responsável por identificar na seqüencia de elementos léxicos as construções da linguagem.

Exemplo : para a sequência abaixo, em Pascal,

“ **`if  a > b then a:=a-b else b:=b-a`** “

A análise sintática deve idenficá-la como um ***comando condicional***, para a qual a ***condição*** é **“`a > b`”** e ao qual estão associados os comandos **“`a := a-b`”** e **“`b := b-a`”.**

Análise Semântica

A análise semântica tem por objetivo verificar se as construções identificadas pela análise sintática estão em acordo com as “regras semânticas” da linguagem sendo compilada.

Exemplo: em Pascal existe uma regra dizendo que *todas as variáveis de um programa devem ser declaradas antes do seu uso*. É função da *análise semântica* verificar se as variáveis usadas no programa foram devidamente declaradas.

Análise Léxica x Análise Sintática

A análise léxica e a análise sintática tem papéis parecidos, já que as duas tratam de sequências de “símbolos”.

A separação das duas em dois níveis de descrição e tratamento simplifica não só a descrição da linguagem como também a implementação do compilador.

Gramáticas e Linguagens

O conceito de *gramática* será usado inicialmente como *ferramenta* para descrever uma *linguagem* e posteriormente como base para a construção dos analisadores léxico sintático.

Definições Iniciais

- **alfabeto** - conjunto finito não vazio de símbolos (denotado por Σ).

- **cadeia (ou sentença)** sobre um alfabeto Σ : é uma seqüência finita de símbolos de Σ.

- **cadeia nula,** denotada por λ (não contém nenhum símbolo)

- **comprimento de uma cadeia α** ($|\alpha|$) : número de símbolos que compõem α.
- **concatenação (produto**) de duas cadeias α e β, denotado por $\alpha\beta$:

se $\alpha = x_1 x_2 \ldots x_n$ e $\beta = y_1 y_2 \ldots y_m$

então $\alpha\beta = x_1 x_2 \ldots x_n \; y_1 y_2 \ldots y_m$

a concatenação de cadeias é associativa: $\alpha(\beta\gamma) = (\alpha\beta)\gamma$

Exemplos

alfabeto - $\Sigma = \{ a, b, c \}$

cadeias (sentenças) sobre Σ : *abc, aaa, cabc, caaaa, aaacccbbb*, **...**

comprimento da cadeia:

$\| abc \| = 3$	$\| caaaa \| = 5$	$\| aaacccbbb \| = 9$
$\| cabc \| = 4$	$\| \lambda \| = 0$	$\| a \| = 1$

concatenação (ou produto): para α = abc , β = aaa e γ = cabc

$\alpha\beta$ = *abcaaa* $\quad\quad$ $\beta\alpha$ = *aaaabc*

$\alpha \lambda = \lambda \alpha = \alpha =$ *abc*

$\alpha (\beta\gamma) = (\alpha\beta) \gamma =$ *abcaaacabc*

$\alpha^1 = \alpha =$ *abc* $\quad\quad$ $\alpha^2 =$ *abcabc*

$\alpha^3 =$ *abcabcabc* $\quad\quad$ $\alpha^0 = \lambda$

$\alpha^n = \alpha^{n-1} \alpha$ *(para n > 0)*

Linguagem

Se Σ é um alfabeto então

- Σ^* representa todas as cadeias sobre Σ
- Σ^+ representa todas as cadeias sobre Σ com comprimento maior ou igual a 1 ($\Sigma^+ = \Sigma^* - \{\lambda\}$).

- Uma **linguagem** sobre um alfabeto Σ é um subconjunto de Σ^* .

Gramática

Define-se uma gramática G como G= (T,N,P,S) onde:

- T - alfabeto denominado **vocabulário terminal** de G
- N - alfabeto denominado **vocabulário não-terminal** de G (T e N são disjuntos e $\Sigma = T \cup N$ é o **vocabulário** de G)
- P é um conjunto de *regras* da forma $X \rightarrow \alpha$ nas quais $X \in N$ e $\alpha \in \Sigma^*$ ($X \rightarrow \alpha$ indica "X pode ser substituído por α")

- $S \in N$ (S é chamado **símbolo inicial** de G).

Um exemplo

G = (N, T, P, E) onde

N = { E, O } T = { **a, b, c, +, *, (,)** }
P = { E $\rightarrow$ **a,** E $\rightarrow$ **b**, E $\rightarrow$ **c**, E $\rightarrow$ **(** E O E **)**,
O $\rightarrow$ *****, O $\rightarrow$ **+** }

P = {
E $\rightarrow$ **a,**
E $\rightarrow$ **b,**
E $\rightarrow$ **c,**
E $\rightarrow$**(**E O E**),**
O $\rightarrow$ ***,**
O $\rightarrow$**+** ,
}

<u>Notação alternativa</u>: regras com o mesmo símbolo não terminal à esquerda podem ser fatoradas através do uso do meta-símbolo ' | *'. Por exemplo, as regras com o símbolo* E *à esquerda podem ser reescritas como:*

E $\rightarrow$ **a | b | c | (E O E)**

Derivações

Derivação direta: $\alpha \Rightarrow \beta$ se α é da forma $\gamma X \rho$, $X \in N$, β é da forma $\gamma \psi \rho$, $\psi \in \Sigma^*$ e $X \rightarrow \psi \in P$

Informalmente: a sentença α deriva diretamente a sentença β se β pode ser obtida a partir de α através da aplicação de uma das regras de substituição de P.

Exemplo: na gramática do exemplo anterior,

(E O E **)** $\Rightarrow$ **(** E **+** E **)** aplicando-se a regra O $\rightarrow$ **+**
(E **+** E **)** $\Rightarrow$ **(** a **+** E **)** aplicando-se a regra E $\rightarrow$ **a**
(a **+** E **)** $\Rightarrow$ **(** a **+** b **)** aplicando-se a regra E $\rightarrow$ **b**

Derivações

Derivação em um ou mais passos:

$\alpha \Rightarrow^{+} \beta$ se existem $\gamma_1\gamma_2...\gamma_n$ tais que

$\alpha \Rightarrow \gamma_1, \gamma_1 \Rightarrow \gamma_2, ...\gamma_n \Rightarrow \beta$

Informalmente: a sentença α deriva diretamente a sentença β se β pode ser obtida a partir de α através da aplicações sucessivas das regras de substituição de P.

Exemplo: Pelo exemplo anterior,

(E O E **)** $\Rightarrow^{+}$ **(** a **+** b **)**

Derivação em um ou mais passos:

$\alpha \Rightarrow^{*} \beta$ se $\alpha = \beta$ ou $\alpha \Rightarrow^{+} \beta$

Linguagem gerada por uma gramática

Forma sentencial:

α é uma forma sentencial em G se $S \Rightarrow^{*} \alpha$

Linguagem gerada por G:

$$L(G) = \{ \alpha \mid \alpha \in T^{*} \text{ e } S \Rightarrow^{*} \alpha \}$$

Informalmente: a linguagem gerada por uma gramática G corresponde ao conjunto de sentenças formadas por símbolos terminais, derivadas do símbolo inicial.

Um Exemplo

G = (N,T,P,E) onde

T = { **(,), +, *, a, b, c** } N = { E, O, E }

P = { E $\rightarrow$ (E O E) | a | b | c

O $\rightarrow$ **+** | *****

}

L(G) = conjunto de “expressões” delimitadas por parênteses, como p. ex. (a+((a+c)*b))

Outro Exemplo

G = (N,T,P,Z) onde

N = { Z, S, D, M }

T = { **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -** }

P = { Z $\rightarrow$ SM | M, M $\rightarrow$ D| DM, S $\rightarrow$ **+** | **-**

D $\rightarrow$ **0** | **1**| **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

}

L(G) = conjunto das representações em decimal dos números inteiros.

Limitações da Gramática

- Uma gramática G, da forma como foi definida, não consegue descrever todas as situações que ocorrem numa linguagem de programação.
- O tipo de gramática apresentado é chamado na literatura de ´gramática livre de contexto´ (*context free grammar*) porque uma regra pode ser aplicada a um não terminal, independentemente do contexto em que ele ocorre na forma sentencial.
- Um exemplo: em muitas linguagens, para que uma variável possa ser usada num programa, é necessário que a mesma tenha sido declarada.

O Problema da Análise Sintática

- O problema da análise sintática consiste em:

  Dada uma gramática G = (T,N,P,S) e uma sentença α

  – Determinar se α L(G)

  – Determinar a sequência de regras de P tal que

  $S \Rightarrow^{+} \alpha$

  A sequência de regras usada na derivação de α define a ´estrutura´ da sentença, que num compilador é a base para o processo de tradução.

O Problema da Análise Sintática

- Uma tentativa: verificar se uma sentença $\alpha \in L(G)$

$C_0 = \{S\}$
i = 0;
repita
    i = i+1;
    $C_i = \{ \beta \mid \gamma \Rightarrow \beta$ *para algum* $\gamma \in C_{i-1}$ e $|\gamma| \leq |\alpha| \}$;
até $(C_i = \{\})$ ou $(\alpha \in C_i)$;

  - O algoritmo acima pode ser estendido para manter a sequência de regras aplicadas a cada elemento de C_i.
  - Não é prático (um bom exercício é verificar se o algoritmo acima tem um comportamento polinomial ou exponencial).

Análise sintática na prática

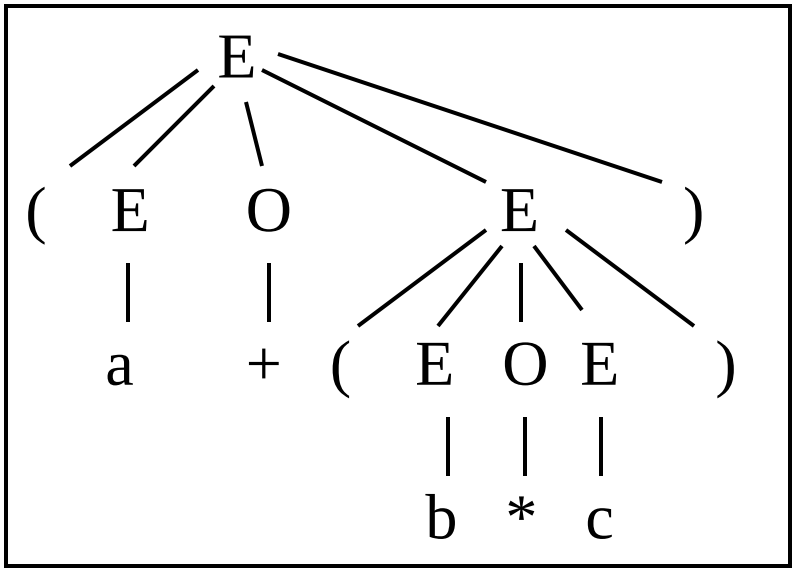
Para que a análise sintática seja usada num compilador real, em geral o que se faz é

- Restringir as regras da gramática, com eventuais restrições às linguagens geradas pelas mesmas.
- Relaxar a definição da linguagem, deixando algumas verificações para a fase de verificação semântica.

Árvore de Derivação

A *árvore de derivação* de uma sentença descreve univocamente como a mesma é gerada.

***Exemplo**:*



Derivações Canônicas I

A seqüencia de derivações diretas pode ser usada para descrever a forma pela qual uma sentença é gerada.

Exemplo:

$$E \Rightarrow (\ E\ O\ E\ ) \Rightarrow (\ E + E\ ) \Rightarrow (\ a + E\ ) \Rightarrow (\ a + b\ )$$

Essa seqüencia não é única uma vez que nela existem formas sentenciais com mais de um símbolo não terminal e existe pelo menos uma substituição possível para cada um.

Derivações Canônicas II

As derivações canônicas são derivações nas quais a substituição é aplicável a um único símbolo não terminal.

• numa derivação (canônica) **mais à esquerda**, a substituição é sempre aplicada ao não terminal mais à esquerda da forma sentencial.

• numa derivação (canônica) **mais à direita**, ...

Uma derivação canônica corresponde biunivocamente a uma árvore de derivação.

Gramática Ambígua

Uma gramática é ambígua se existe pelo menos uma sentença para a qual existem pelo menos duas árvores de derivação (ou ainda, existem pelo menos duas derivações mais à esquerda ou pelo menos duas derivações mais à direita).

Exemplo:

G = (N, T, P, E) onde

N = { E } T = { **a, b, c, +, *** }

P = { E $\rightarrow$ **a** | **b** | **c** | E **+** E | E ***** E }

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

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

Ambiguidade x Compilador

Num compilador, o processo de tradução é baseado na derivação da sentença (programa).

Se a gramática que define a linguagem sendo compilada for ambígua, existirá pelo menos um programa para o qual são possíveis pelo menos duas traduções.

Um exemplo clássico, em Pascal (o mesmo problema existe em C e Java):

C $\rightarrow$ *a* | **if** *b* **then** C **else** C | **if** *b* **then** C

```
if a <> b then if a > b then a:=a-b else b:=b-a
```

Ambiguidade x Compilador

- Num compilador a ambiguidade deve ser evitada.
- A primeira idéia para se evitar construções ambíguas é alterar a gramática.
- Nem sempre a gramática alterada adere às restrições impostas pelo analisador sintático.
- Em alguns casos, a ambiguidade é contornada através de regras que complementam a gramática.

Gramáticas Reduzidas

Uma gramática é *reduzida* se satisfaz às seguintes restrições:

- Para qualquer símbolo $X \in \Sigma$,

  $S \Rightarrow^* \alpha X \beta$ para algum α e β

- $X \Rightarrow^* w$ para algum $w \in T^*$

Em outras palavras: todo símbolo X é derivável a partir do símbolo inicial e todo símbolo leva a uma cadeia formada apenas por terminais. A gramática não contém regras ou símbolos “inúteis”.

Análise Sintática

A **análise sintática** ou ***parsing*** de uma sentença **w**, numa gramática **G**, consiste em verificar se **w** $\in$ **L(G)** e em caso positivo, determinar a **estrutura sintática** de **w** em **G**.

A estrutura sintática de **w** pode ser descrita por uma **árvore de derivação** de **w** em G ou por uma **derivação canônica** de **w** em **G**.

Existem duas grandes famílias de métodos de análise sintática:

- análise sintática **descendente**
- análise sintática **ascendente**

Linguagens Regulares

Uma linguagem é chamada *linguagem regular* se ela pode ser descrita por uma gramática cujas regras são da forma:

$A \rightarrow a \quad$ ou $\quad A \rightarrow aB$

onde A e B são *não terminais* e a é um *terminal*.

Exemplo:

G = { {S,A,B,C}, { 0, 1}, P, S} onde

p = { $S \rightarrow aS \mid bB$, $B \rightarrow bB \mid cC \mid c$, $C \rightarrow cC \mid c$ }

L(G) = { w | w = $a^i b^j c^k$ e $i \geq 0$, $j \geq 1$, $k \geq 1$ }

Autômato de Estados Finitos

Um autômato de estados finitos é definido por

$A = (E, T, M, E_i, F)$:

E - conjunto de estados E, finito

T - um alfabeto T

M - função de transição M que associa pares de

$E \times T$ aos elementos de E

E_i - um estado $E_i \in E$, dito *estado inicial*

F - conjunto de estados finais ($F \subseteq E$).

Um exemplo I

A = ($\{E_1, E_2, E_3, E_4\}$, $\{a,b,c\}$, M, E_1, $\{E_4\}$), onde M é definida por

M(E1,a) = E2 M(E1,b)= E3 M(E1,c) = E4

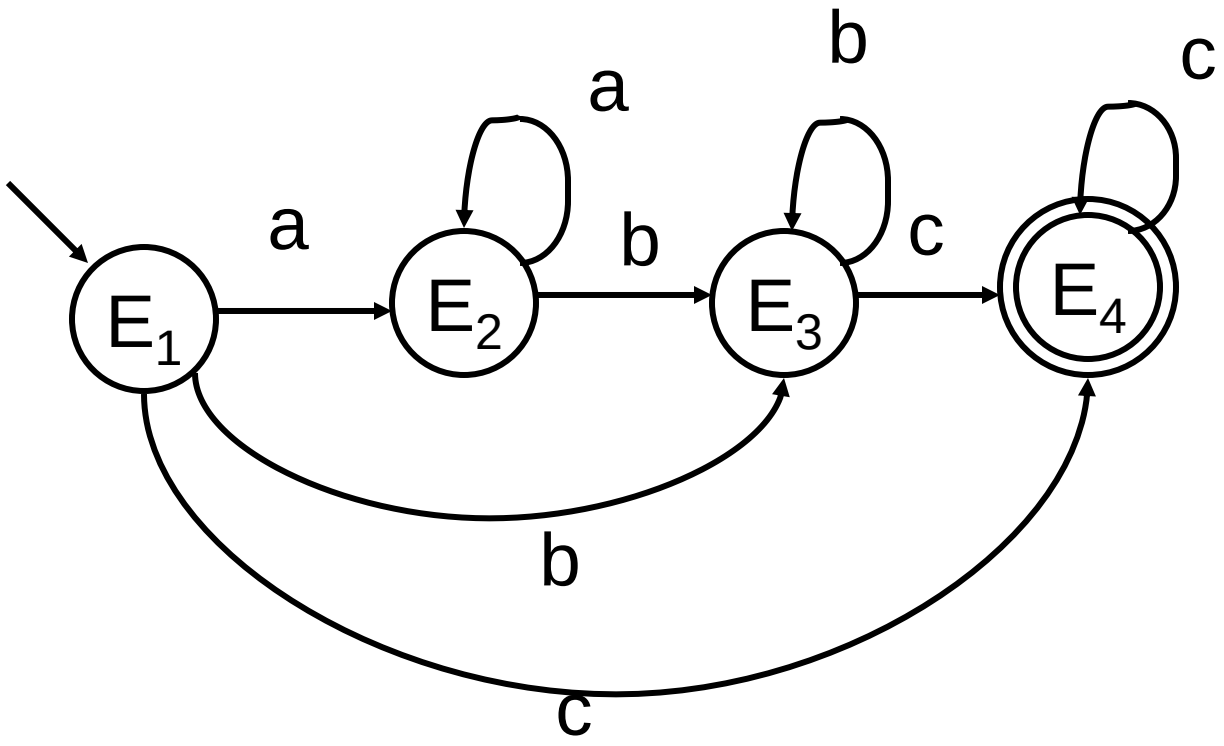
M(E2,a) = E2 M(E2,b) = E3 M(E2,c) = E4

M(E3,b) = E3 M(E3,c) = E4

M(E4,c) = E4

Um exemplo II

Representação gráfica:



Linguagem

Uma sentença $w = x_1x_2 .. x_k$ é aceita por um autômato de estados finitos se existe uma sequência de transições da forma:

$$M(E_1,x_1) = E_2, \;\; M(E_2,x_2) = E_3, \; ... \;\; M(E_k,x_k) = E \in F.$$

A linguagem aceita por um autômato de estados finitos é uma linguagem regular.

Autômato e Análise Léxica

O conjunto de símbolos léxicos de uma linguagem de programação são descritos por uma linguagem regular, que pode ser reconhecida por um autômato de estados finitos.

Implementação I

Dirigido por Tabela: Uma tabela M[E,x] descreve a função de transição e um programa genérico é dirigido pela mesma:

```
E= EstadoInicial;
repita
{
  ch=PróximoCaracter; E = M[E,ch];
  se (E == EstadoIndefinido
        {
           erro("caracter inválido")
        }
} até FimDeEntrada;
```

Implementação II

Programa Dedicado

O analisador léxico dirigido por tabela tem a vantagem de ser genérico e a desvantagem de ser difícil de se contruir manualmente.

Na prática, o tratamento léxico da maioria das linguagens de programação pode ser feita através de um programa dedicado.

Normalmente o “analisador léxico” é implementado como uma função que devolve o código do próximo símbolo da entrada. Essa função é chamada pelo analisador sintático cada vez que este necessita de um símbolo.

Um exemplo

Reconhecimento de inteiros e identificadores:

```
função proximo_símbolo () {
  enquanto (caracter == ´ ´ ) leia caracter;
  caso caracter seja{
     ´a´..´z´:  enquanto (´a´ ≤ caracter ≤ ´z´)
                   leia caracter;
                retorne ident;
     ´0´..´9´:  enquanto(´0´ ≤ caracter ≤ ´0´)
                   leia caracter;
                retorne numero;
       outro:    erro( "caracter inválido");
  }
}
```