Autômatos e Linguagens

Eduardo Ferreira dos Santos

Ciência da Computação
Centro Universitário de Brasília – UniCEUB

Agosto, 2016

Sumário

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

4 A estrutura de um compilador

1. Compiladores
2. Linguagens de programação
3. Ciência dos compiladores
4. A estrutura de um compilador

Linguagens

- Teoria de linguagens formais [UNICER, 2001]:
  - Estudo das características, propriedades e aplicações da linguagem formal;
  - Representação da estrutura: sintaxe;
  - Determinação do significado: semântica.
- É necessário estudar as linguagens formais no domínio da matemática.
  - *uma linguagem é uma forma de comunicação, usada por sujeitos de uma determinada comunidade;*
  - *uma linguagem é o conjunto de SÍMBOLOS e REGRAS para combinar esses símbolos em sentenças sintaticamente corretas;*
  - *uma linguagem é formal quando pode ser representada através de um sistema com sustentação matemática.*

Símbolo

- Símbolo: entidade abstrata sem definição formal;
- Ex.: letras, dígitos, etc.
- Ordenação lexicofráfica [UNICER, 2001]: igualdade ou precedência;
- Usados como elementos atômicos em definições de sintaxe.

Alfabeto

- Definição: sequência finita de símbolos;
- Exemplos:
  - $\beta = \{0, 1\}$;
  - $\Gamma = \{a, b, c, d, e, f\}$.
- Uma palavra sobre um alfabeto β é uma sequência finita de símbolos de β.
- Ex.: (1, 1, 0, 0, 1) (tupla);
- Representamos apenas como 11001.

Alfabetos e palavras

- $|\rho|$ denota o número de símbolos da palavra $|\rho|$.
- Ex.: $|11001| = 5$
- Uma linguagem sobre um alfabeto β é um conjunto de palavras sobre β.
- Ex.: $L = \{1^p \mid \text{p é primo}\} = \{11, 111, 11111, 1111111, \ldots\}$

Computador formal

Figura 1.1: Exemplo de computador formal [Pinto, 2016]

Processadores de linguagem

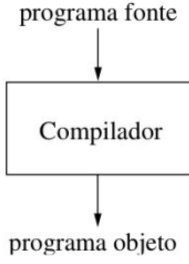

Figura 1.2: Um compilador [Aho et al., 2007]

Figura 1.3: Executando o programa objeto [Aho et al., 2007]

Interpretador

- Ao invés de produzir linguagem de máquina, o interpretador executa diretamente as operações especificadas no programa fonte sobre as entradas do usuário;
- Normalmente o programa objeto é mais rápido;
- O interpretador oferece melhor diagnóstico de erros.

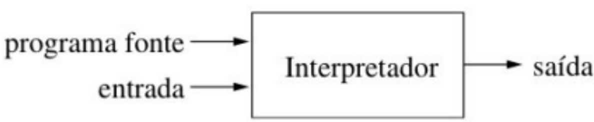

Figura 1.4: Um interpretador [Aho et al., 2007]

Compiladores Java

- O programa Java primeiro gera código intermediário: *bytecode*;
- Os *bytecodes* são interpretados por uma máquina virtual;
- Conceito de compilação universal.

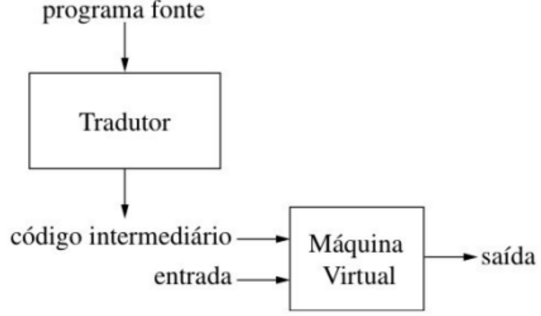

Figura 1.5: Um compilador híbrido [Aho et al., 2007]

Pré-processamento

- Alguns outros programas podem ser necessários para a geração do executável;
- O pré-processador é responsável por coletar o programa fonte e, possivelmente, expandir macros em comandos na linguagem fonte.

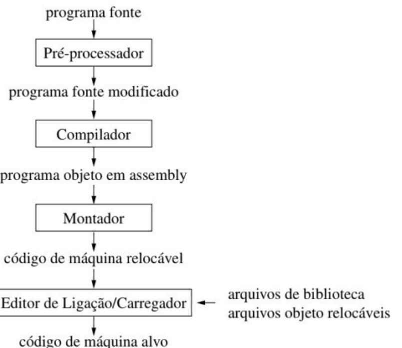

Figura 1.6: Um sistema de processamento de linguagem [Aho et al., 2007]

Modelo de análise e síntese

- A análise impõe um modelo gramatical para o código;
- Caso esteja sintaticamente mal formado ou semanticamente incorreto, deve informar qual é o erro;
- Gera também a tabela de símbolos, passada para a próxima etapa junto com a apresentação intermediária;
- A parte de síntese constrói o programa objeto a partir da tabela de símbolos e da representação intermediária;
- A compilação é organizada em fases, onde cada etapa transforma a representação anterior para a próxima camada.

Fases

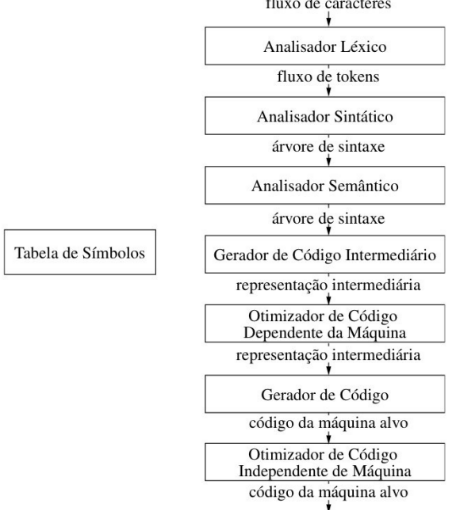

Figura 1.7: Fases do compilador [Aho et al., 2007]

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

4 A estrutura de um compilador

Computadores e linguagens

- Um computador formal tem o objetivo principal de transformar linguagem fonte em linguagem objeto;
- A linguagem fonte é uma abstração de alto nível implementada no programa de computador;
- A linguagem objeto é o conjunto de símbolos que serão posteriormente lidos pelo processador;
- O programa objeto pode então ser chamado pelo usuário para processar entradas e produzir saídas [Aho et al., 2007].

UniCEUB
Centro Universitário de Brasília

Linguagens de programação

- No início eram traduções das instruções de máquina;
- Introdução às linguagens orientados ao cálculo numérico: Fortran, LISP e Cobol;
- Linguagens de primeira geração: linguagens de máquina;
- Linguagens de segunda geração: linguagens simbólicas ou de montagem, como Assembly;
- Linguagens de terceira geração: procedurais de alto nível, como Fortran, LISP, Cobol, etc;
- Linguagens de quarta geração: aplicações específicas, como NOMAD para relatórios;
- Linguagens de quinta geração: lógica com restrição, tipo Prolog e OPS5.

Classificações

- Linguagens imperativas;
- Linguagens declarativas;
- Linguagens de Von Neumann (estruturadas);
- Linguagens orientadas a objeto;
- Linguagens de *scripting*.

Impactos

Como os mudanças nas linguagens de programa afetam os compiladores?

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

4 A estrutura de um compilador

Modelagem

- Ferramentas para descrever os algoritmos e as unidades léxicas utilizadas pelos compiladores:
  - **Autômatos Finitos** Representação gráfica;
  - **Expressões Regulares** Representação matemática.
- Ferramentas utilizadas para descrever a estrutura sintática:
  - **Gramáticas livres de contexto** Representação matemática;
  - **Árvores** Representação gráfica.

UniCEUB Centro Universitário de Brasília

Autômatos finitos

Figura 3.1: Um autômato [Pinto, 2016]

Expressões regulares

São expressões (sequências de símbolos), definidas recursivamente, que representam linguagens sobre um alfabeto Σ

Expressões Regulares

Expressão regular	representa a linguagem
$\emptyset$	vazia
ε	$\{\varepsilon\}$
$\boldsymbol{a}$	$\{a\}$, para cada $a \in \Sigma$
$(\boldsymbol{r} + \boldsymbol{s})$	$R \cup S$
$(\boldsymbol{r}\boldsymbol{s})$	RS
$(\boldsymbol{r}^*)$	R^*

onde $\boldsymbol{r}$ e $\boldsymbol{s}$ são expressões regulares representando as linguagens R e S

Figura 3.2: Algumas expressões regulares [Pinto, 2016]

UniCEUB Centro Universitário de Brasília

Gramáticas livres de contexto

Mecanismo de formação de palavras (sentenças) a partir de substituição de variáveis [Pinto, 2016].

$$E \rightarrow \epsilon \tag{1}$$

$$E \rightarrow 0E1 \tag{2}$$

Exemplos de gramática [Pinto, 2016]

Árvores

Figura 3.3: Árvore de parsing [Amarasinghe and Rinard, 2010]

Otimização

- A otimização do código depende da capacidade de controlar a saída para todas as possíveis saídas;
- Caso a afirmação seja válida, dizemos que a saída é ótima;
- Requisitos da otimização [Aho et al., 2007]:
  - *A otimização precisa ser correta, ou seja, preservar a semântica do programa compilado;*
  - *A otimização precisa melhorar o desempenho de muitos programas;*
  - *O tempo de compilação precisa continuar razoável;*
  - *O esforço de engenharia empregado precisa ser administrável.*
- Importância da corretude;
- Conceito de eficiência;
- Novo paradigma: consumo de energia!

Execução eficiente

- Do alto para o baixo nível [Amarasinghe and Rinard, 2010]:
  - Mapeamento puro e simples do programa para o *assembly* normalmente gera uma execução ineficiente;
  - Quanto maior o nível de abstração, mais ineficiente.
- As abstrações para alto nível só são úteis se forem eficientes;
- Um bom compilador fornece a possibilidade de escrever em alto nível de abstração com a performance de instruções de baixo nível.

Exemplo de otimização

Listing 1: [Amarasinghe and Rinard, 2010]

```
int sumcalc(int a, int b, int N) {
  int i;
  int x, y;
  x = 0;
  y = 0;
  for (i=0; i <= N; i++) {
    x = x+4*a/b*i+(i+1)+(i+1);
    x = x + b*y;
  }

  return x;
}
```

Como otimizar esse código?

UniCEUB Centro Universitário de Brasília

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

4 A estrutura de um compilador

Execução

- A fase de análise divide o programa e impõe uma estrutura gramatical;
- Se houver algum erro, deve fornecer mensagens esclarecedoras;
- A fase de síntese constrói o programa objeto usando:
  - Representação intermediária;
  - Tabela de símbolos.
- Normalmente a compilação é dividida em fases, como descrito na Figura 14.

UniCEUB
Centro Universitário de Brasília

Análise léxica

- Normalmente a compilação se inicia pela análise léxica;
- Analisador léxico: lê o fluxo de caracteres e agrupa em sequências significativas;
- As sequências significativas são chamadas lexemas;
- Para cada lexema, o analisador léxico produz como saída um token;

$$\langle nome_token, valor_atributo \rangle \tag{3}$$

- O token é enviado para a etapa seguinte, a análise sintática.

Exemplo

$$position = initial + rate * 60 \tag{4}$$

position
: Mapeado para o token $\langle id, 1 \rangle$:
  - id: Símbolo abstrato que significa identificador;
  - 1: Entrada na tabela de símbolos onde está *position*.

=
: Mapeado para o token $\langle = \rangle$. Por não exigir um valor de atributo, omitimos o segundo componente;

initial
: Mapeado para o token $\langle id, 2 \rangle$;

+
: Mapeado para o token $\langle + \rangle$;

rate
: Mapeado para o token $\langle id, 3 \rangle$;

*
: Mapeado para o token $\langle * \rangle$;

60
: Mapeado para o token $\langle 60 \rangle$[1].

[1] Para o lexema 60 deveria haver um token como $\langle numero, 4 \rangle$

Atribuição

- Após a análise léxica, executa-se o comando de atribuição;
- O comando gera as substituições apontadas no exemplo 4;
- Após a substituição, obtemos o resultado 5:

$$\langle id, 1\rangle\langle=\rangle\langle id, 2\rangle\langle+\rangle\langle id, 3\rangle\langle*\rangle\langle 60\rangle \tag{5}$$

- É possível perceber que alguns tokens são símbolos abstratos;
- Os símbolos representam operadores.

Análise sintática

- Analisador sintático: utiliza os primeiros componentes dos tokens para gerar uma representação de árvore;
- A árvore representa a estrutura gramatical dos tokens;
- Representação da árvore de sintaxe:
  - Cada nó interior representa uma operação;
  - Filhos dos nós representam os argumentos da operação.
- As próximas fases do compilador utilizam a estrutura gramatical;
- Gerar o programa fonte;
- Gerar o programa objeto.

Árvore de sintaxe

position = initial + rate * 60

Analisador Léxico

⟨**id**, 1⟩⟨=⟩⟨**id**, 2⟩⟨+⟩⟨**id**, 3⟩⟨+⟩⟨60⟩

Analisador Sintático

=
⟨**id**, 1⟩ +
⟨**id**, 2⟩ *
⟨**id**, 3⟩ 60

Analisador Semântico

=
⟨**id**, 1⟩ +
⟨**id**, 2⟩ *
⟨**id**, 3⟩ **inttofloat**
60

Gerador de Código Intermediário

position	...
initial	...
rate	...

TABELA DE SÍMBOLOS

Figura 4.1: Tradução de uma instrução para o exemplo 4 [Amarasinghe and Rinard, 2010]

Análise semântica

- Utiliza a árvore de sintaxe e a tabela de símbolos para verificar a consistência semântica;
- Relaciona a semântica do programa fonte com a definição da linguagem;
- Reúne informações sobre tipos e salva na árvore de sintaxe;
- Realiza a verificação de tipos;
- Permite algumas conversões de tipo chamada coerções:
  - Aplicar um operador aritmético binário a um par de inteiros;
  - Ao aplicar um operador a um ponto flutuante e a um inteiro, esse último pode ser convertido para ponto flutuante;
  - Ex.: Na figura 35 aparece o operador inttofloat.

Código intermediário

- O compilador pode produzir uma ou mais representações intermediárias;
- Após as análises sintática e semântica, o compilador normalmente gera uma representação mais próxima da linguagem de máquina;
- Objetivos da representação de baixo nível:
  1. Ser facilmente produzida;
  2. Ser facilmente traduzida para a máquina alvo.

Tradução e atribuição

position	...
initial	...
rate	...

TABELA DE SÍMBOLOS

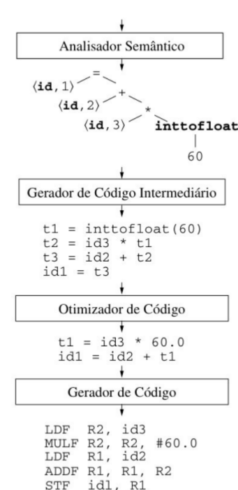

Figura 4.2: Tradução e atribuição para o exemplo 4 [Amarasinghe and Rinard, 2010]

Três endereços

- Representação em código de três endereços:

$$\begin{aligned} t1 &= inttofloat(60) \\ t2 &= id3 * t1 \\ t3 &= id2 + t2 \\ id1 &= t3 \end{aligned}$$

1. Cada instrução de atribuição tem no máximo um operador do lado direito;
    - Determinam a ordem de realização das operações.
2. O compilador precisa guardar o valor computador pela instrução;
    - Geração de nome intermediário.
3. Algumas instruções podem ter menos de três operandos.

OBRIGADO!!!
PERGUNTAS???

Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007).
Compiladores–Princípios Técnicas e Ferramentas.
Pearson, 2a. edition.

Amarasinghe, S. and Rinard, M. (2010).
Computer language engineering.
Disponível em http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-035-computer-language-engineering-spring-2010/ Acessado em 02/08/2016.

Pinto, G. (2016).
Notas de aula do Prof. Guilherme Pinto.

UNICER (2001).
Apostila de compiladores.