Compiladores

Eduardo Ferreira dos Santos

Ciência da Computação
Centro Universitário de Brasília – UniCEUB

Fevereiro, 2017

Sumário

1. Compiladores
2. Linguagens de programação
3. Ciência dos compiladores

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

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.: $L = \{1^p \mid \text{p é primo}\} = \{11, 111, 11111, 1111111, ...\}$

Computador formal

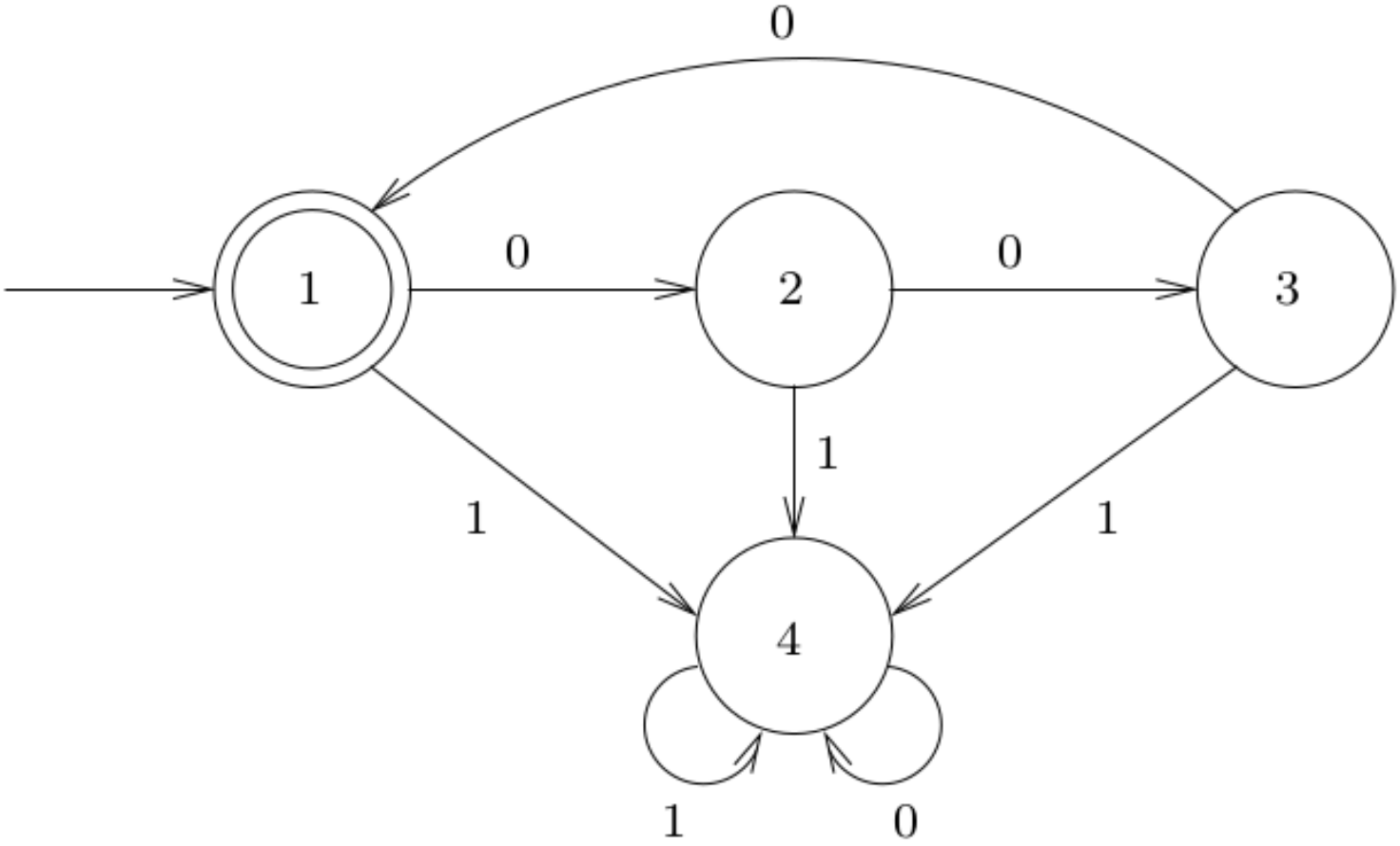

Figura 1.1: Exemplo de computador formal [Pinto, 2016]

Processadores de linguagem

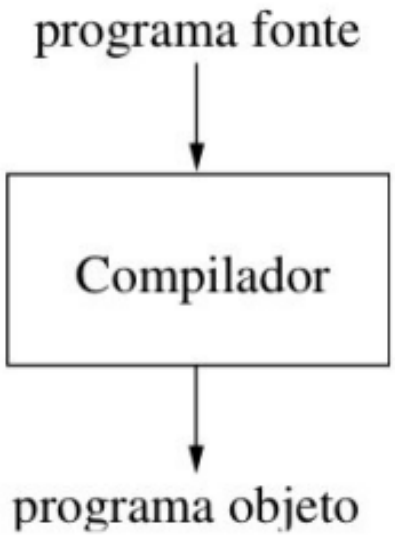

Figura 1.2: Um compilador [Aho et al., 2007]

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.

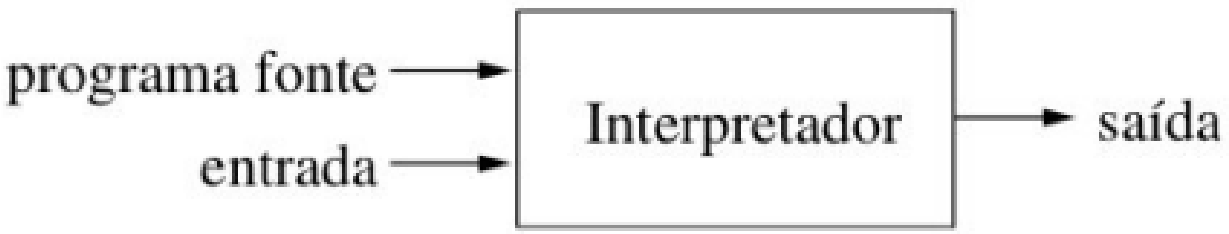

Figura 1.4: Um interpretador [Aho et al., 2007]

Compiladores Java

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

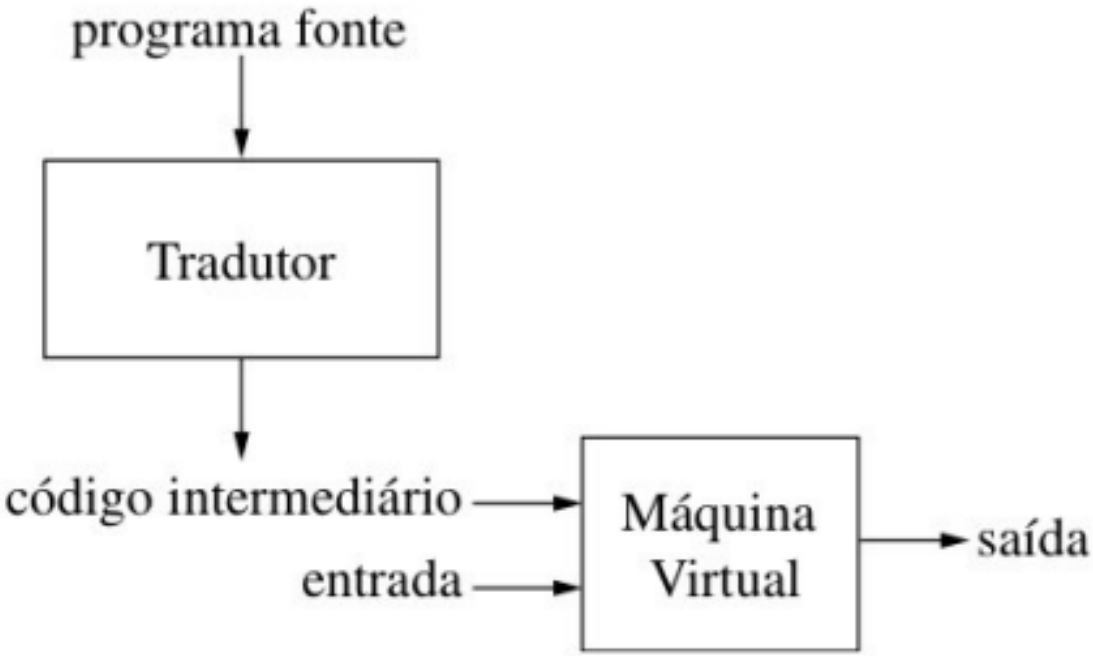

Figura 1.5: Um compilador híbrido [Aho et al., 2007]

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.

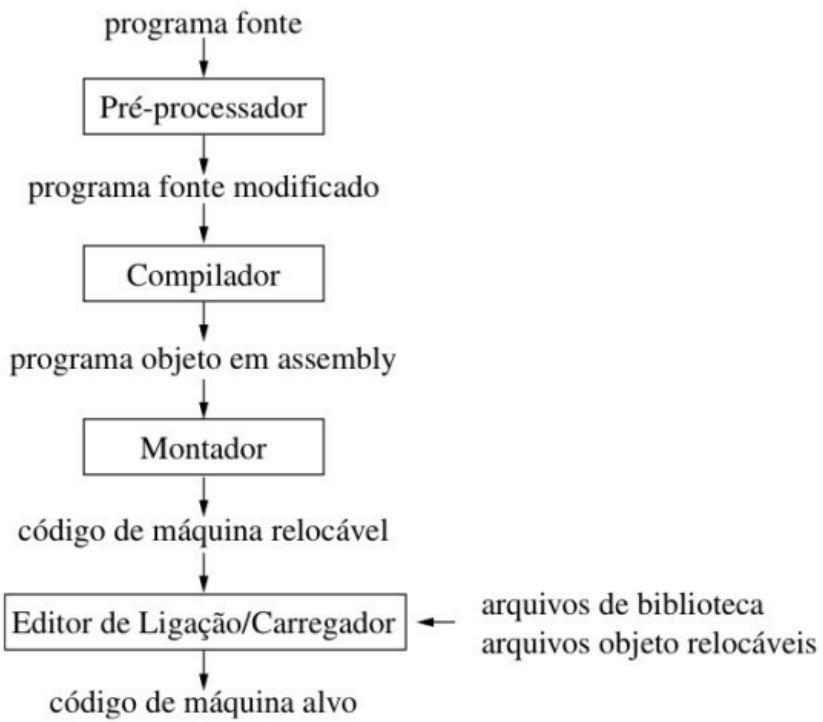

Figura 1.6: Um sistema de processamento de linguagem [Aho et al., 2007]

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

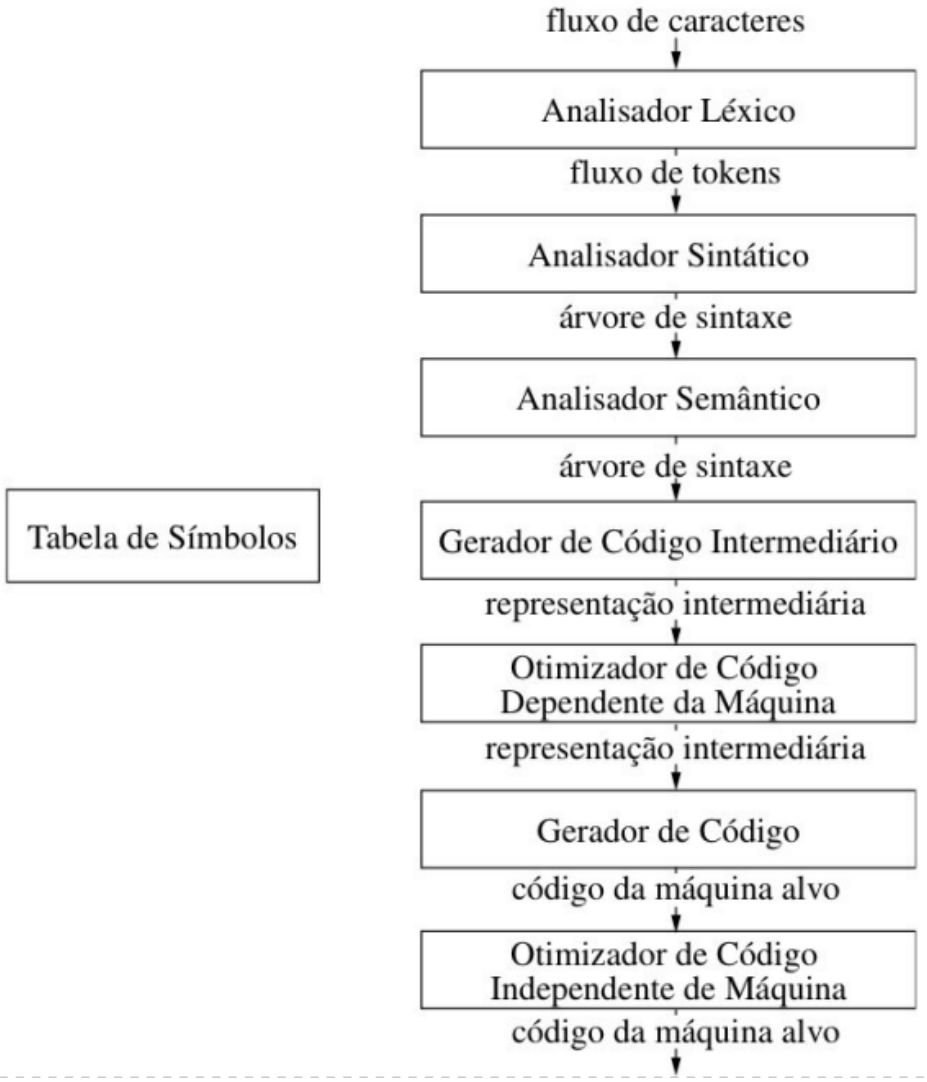

Figura 1.7: Fases do compilador [Aho et al., 2007]

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

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].
- Linguagens de programação: no início eram traduções das instruções de máquina;
- Introdução às linguagens orientados ao cálculo numérico: FORTRAN, LISP e Cobol.

Histórico

- Em 1954 a IBM desenvolve o computador 704;
- O custo do software excede o custo do hardware;
- Toda a programação é feita em Assembly.

Figura 2.1: Computador IBM 704 [1]

[1] Fonte: http://www.computer-history.info/Page4.dir/pages/IBM.704.dir/

Solução

- *Speedcoding*;
- Interpretadores: rodam de 10-20 vezes mais lentos que Assembly [Schwarz et al., 2016];
- FORTRAN I: John Backus;
- Principais ideias:
  - Traduzir código de alto nível para Assembly;
  - Muitos pensavam que era impossível;
  - Já tinha falhado em outros projetos.

FORTRAN I

- 1954-7: Projeto FORTRAN I
- 1958: 50% de todo o software é desenvolvido em FORTRAN;
- Tempo de desenvolvimento cai pela metade;
- O primeiro compilador: teve enorme impacto na Ciência da Computação;
- Deixou uma vasta quantidade de material teórico;
- Os compiladores modernos preservam as raízes do FORTRAN I.

Linguagens de programação

- 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*.

Implementação

- Como as linguagens são implementadas?
  - Compiladores;
  - Interpretadores.
- Os sistemas de compilação em lote (*batch*) dominam. Ex.: gcc
- Algumas linguagens utilizam primeiro interpretadores. Ex.: Java bytecode
- Outras linguagens utilizam abordagem híbrida. Ex.: Lisp
  - Interpretador para o desenvolvimento;
  - Compilador para a produção.

Impactos

Como os mudanças nas linguagens de programa afetam os compiladores?

1 Compiladores

2 Linguagens de programação

3 Ciência dos compiladores

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.

Autômatos finitos

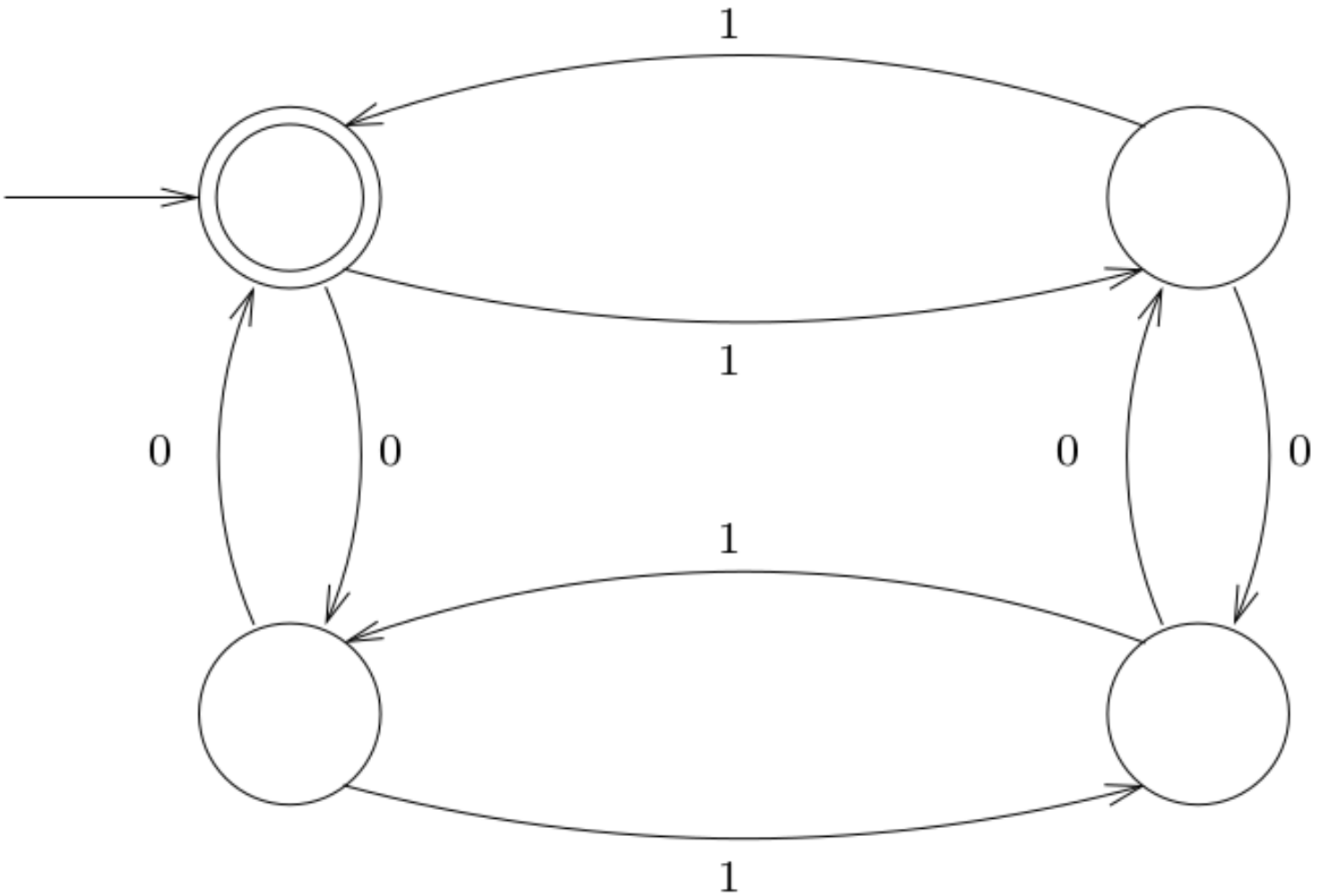

Figura 3.1: Um autômato [Pinto, 2016]

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]

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

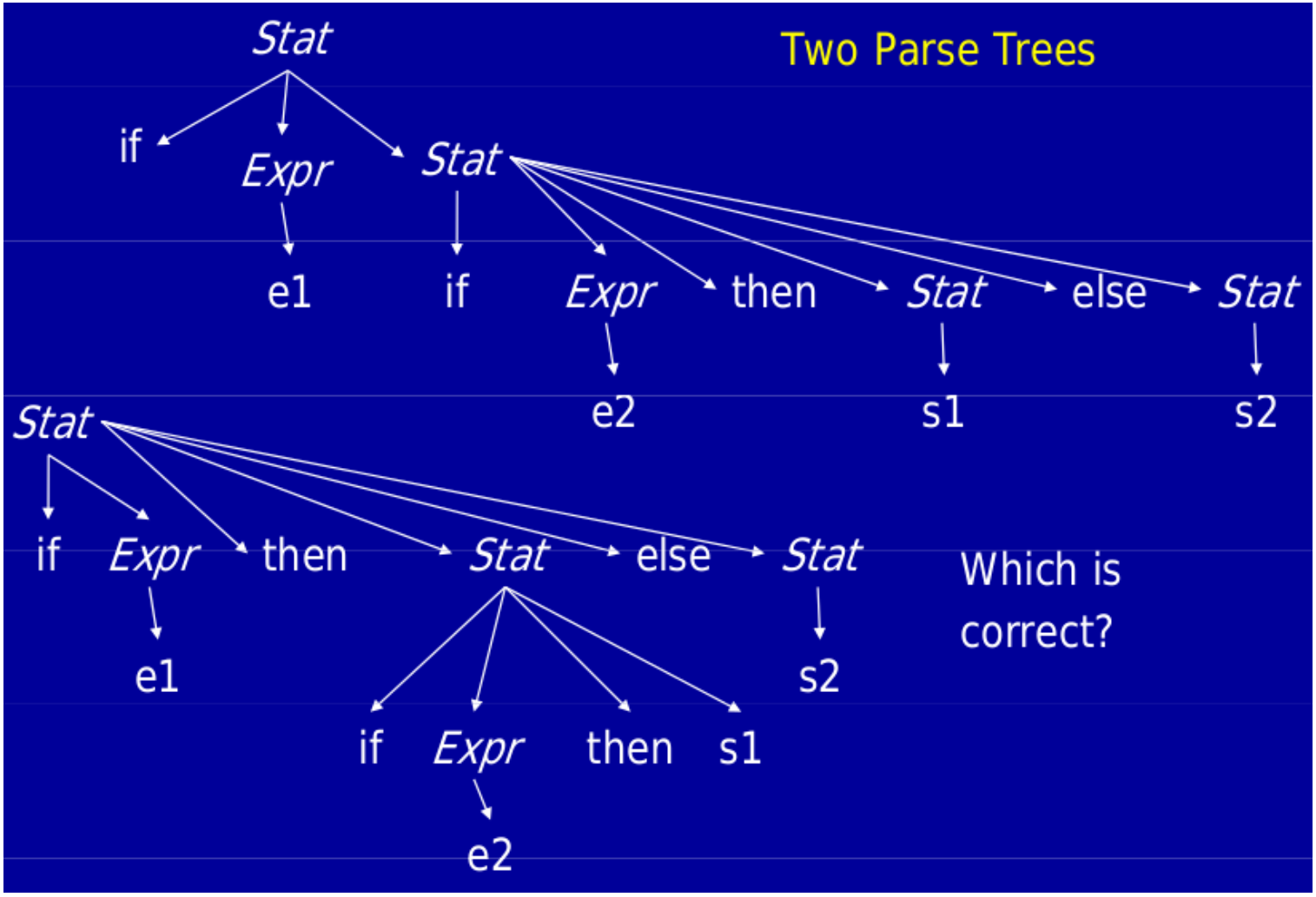

Figura 3.3: Árvore de parsing [Amarasinghe and Rinard, 2010]

Análise Léxica

- Primeiros passos: reconhecer palavras

This is a sentence.

- Análise léxica não é trivial.

ist his a se nte nce

- A análise léxica divide o programa em palavras ou tokens

if x == y then z = 1; else z = 2;

Parsing

- Uma vez que as palavras sejam entendidas, o próxima passo é entender a estrutura;
- *Parsing* = Diagramar sentenças
- A diagramação é organizada através de uma árvore.

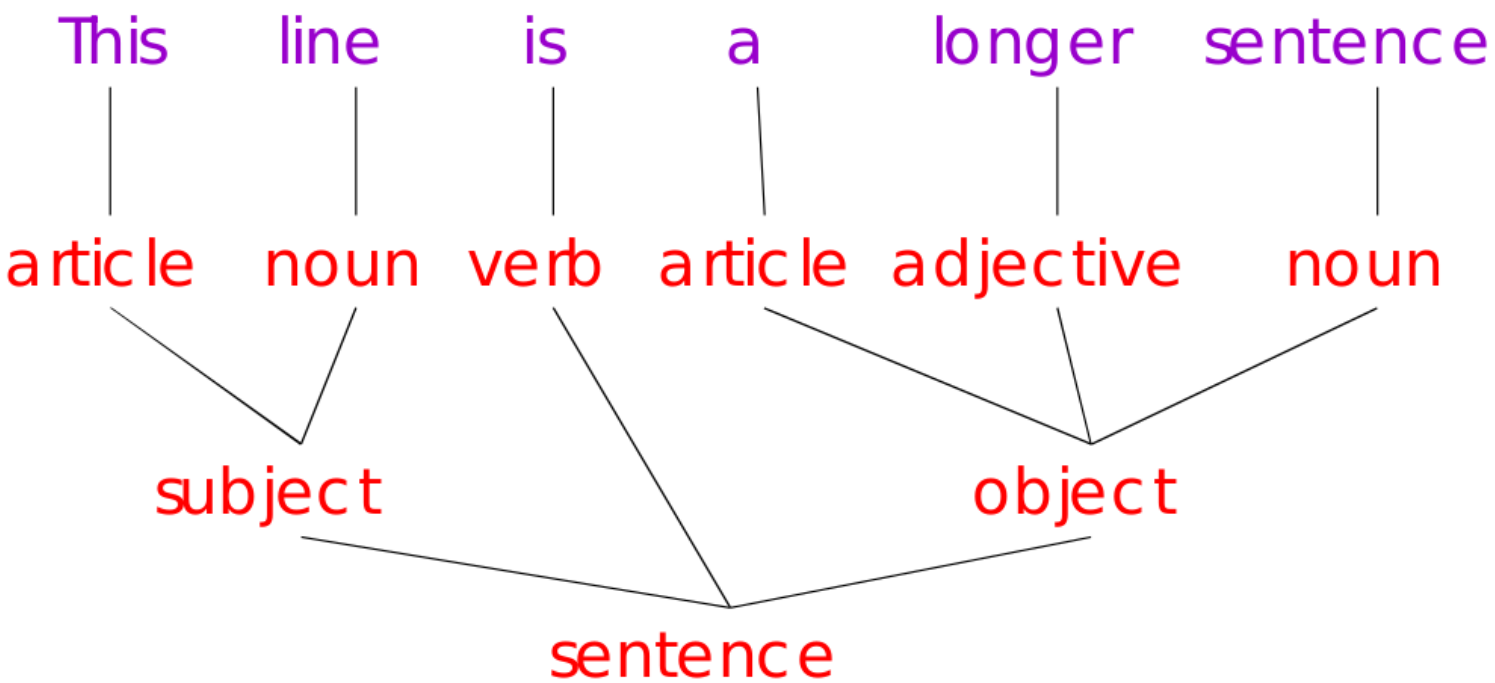

Figura 3.4: Diagramação através de árvore [Schwarz et al., 2016]

Parsing de programas

- Considere o programa:

$$\text{if } x == y \text{ then } z = 1; \text{ else } z = 2;$$

- Realizando o *parsing*:

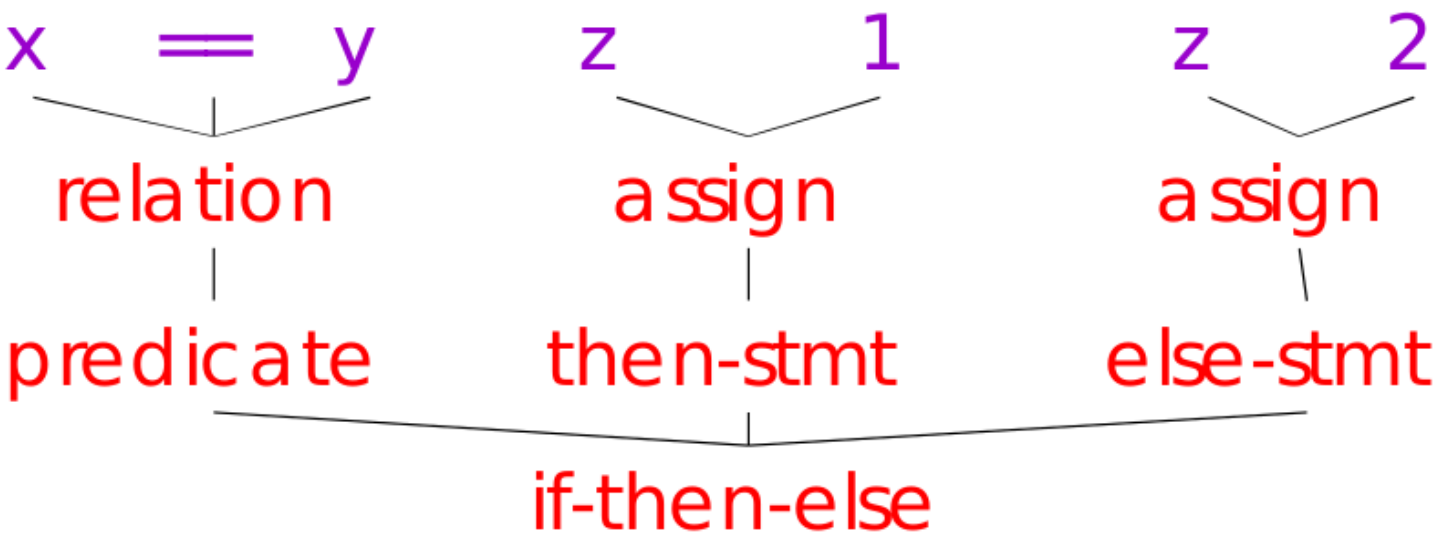

Figura 3.5: *Parsing* de um programa [Schwarz et al., 2016]

Análise semântica

- Uma vez que a sentença esteja construída, podemos entender seu significado;
- Os compiladores realizam uma análise limitada para detectar inconsistências;
- Ex.: O Alexandre disse que o Arthur fez sua tarefa:
  - Qual é o significado do conector sua?
  - A tarefa se refere a Alexandre ou Arthur?
- Complicando: O Alexandre disse que o Alexandre deixou sua tarefa em casa?
  - Quantos Alexandres existem?
  - Qual deles esqueceu a tarefa?

Ambiguidade

- As linguagens de programação definem regras para evitar ambiguidades:

Listing 1: [Schwarz et al., 2016]

```
{
  int Alexandre = 3;
  {
    int Alexandre = 4;
    cout << Alexandre;
  }
}
```

- O código imprime como resultado 4;
- A definição mais de dentro é utilizada.

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 eficência;
- 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 2: [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?

OBRIGADO!!!
PERGUNTAS???


Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007).
Compiladores–Princıpios 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.


Schwarz, K., Papadakis, H., and Mittal, R. (2016).
Compilers.
Disponível em `http://web.stanford.edu/class/cs143/` Acessado em 30/09/2016.


UNICER (2001).

Apostila de compiladores.