



Modelos de Linguagem de Programação

Unidade 4 – Implementação de Linguagens de Programação



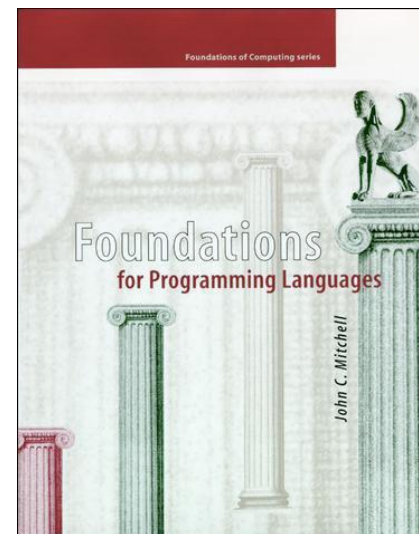
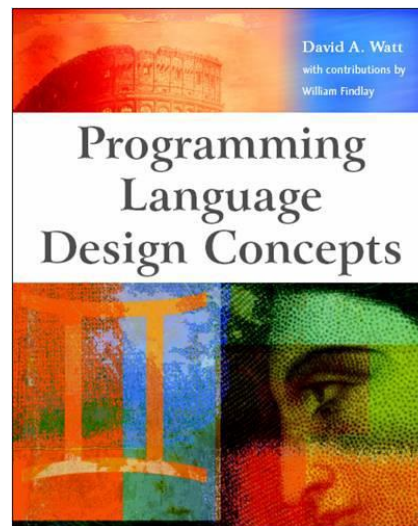
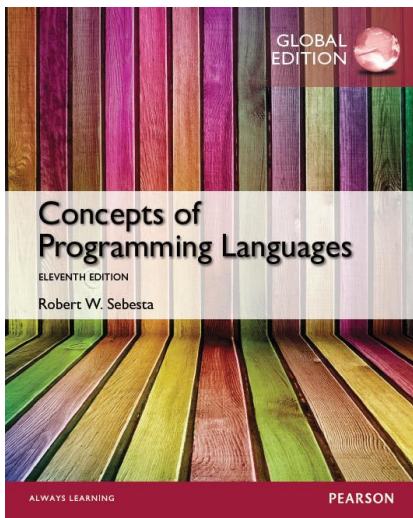
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com





Bibliografia

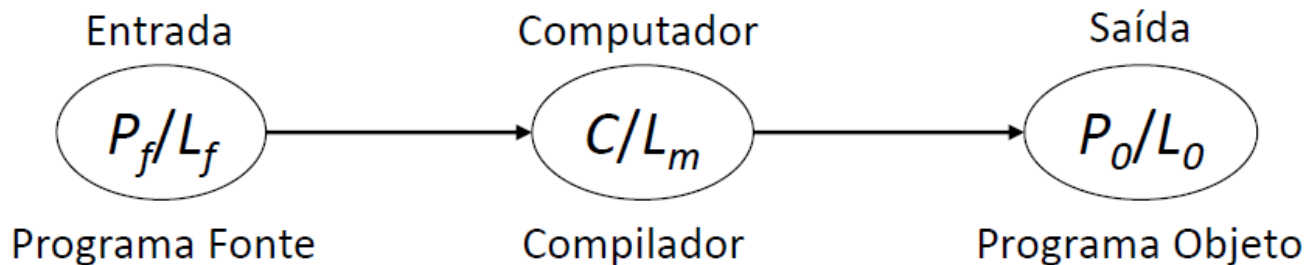
- Sebesta, Robert W. Concepts of Programming Languages – Eleventh Edition
- Watt, D. Programming Language Design Concepts. John Wiley and Sons, 2004.
- Mitchell, J. Foundations for Programming Languages, MIT Press, 1996.





Processo de Compilação

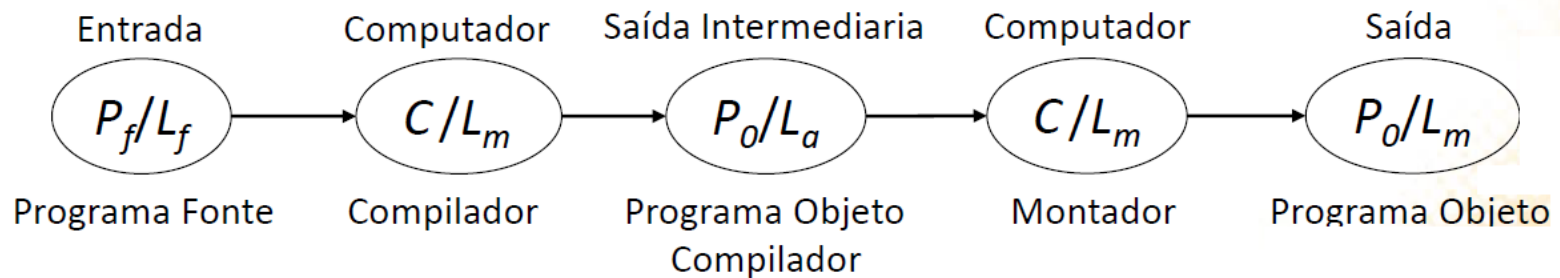
- Um Compilador C é um programa que tem a finalidade de traduzir ou converter um programa P_f (fonte) escrito numa linguagem L_f (linguagem fonte) para um programa P_o (objeto) escrito numa outra linguagem L_o (linguagem objeto).

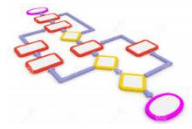




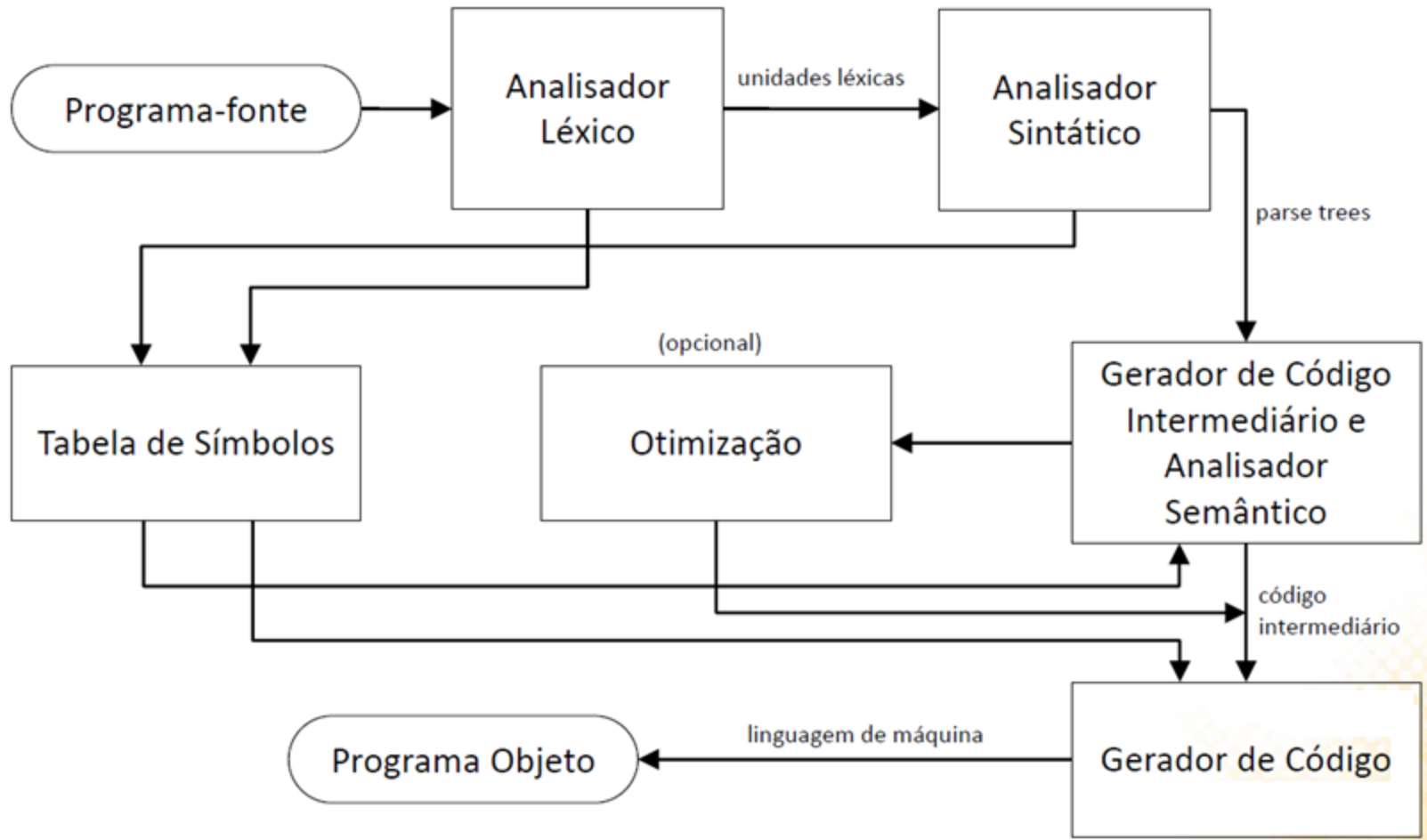
Processo de Compilação

Em geral, L_f é uma linguagem de alto nível como C, COBOL, PASCAL, etc. L_o não é necessariamente uma linguagem de máquina. Por exemplo, L_o pode ser uma Linguagem de montagem ("Assembly") L_a . Nesse caso, é necessário ter-se mais uma fase de tradução de L_a para a linguagem de máquina L_m do computador a ser utilizado para processar o programa objeto.





Processo de Compilação



Processo de Compilação

- Programa-fonte:**

```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

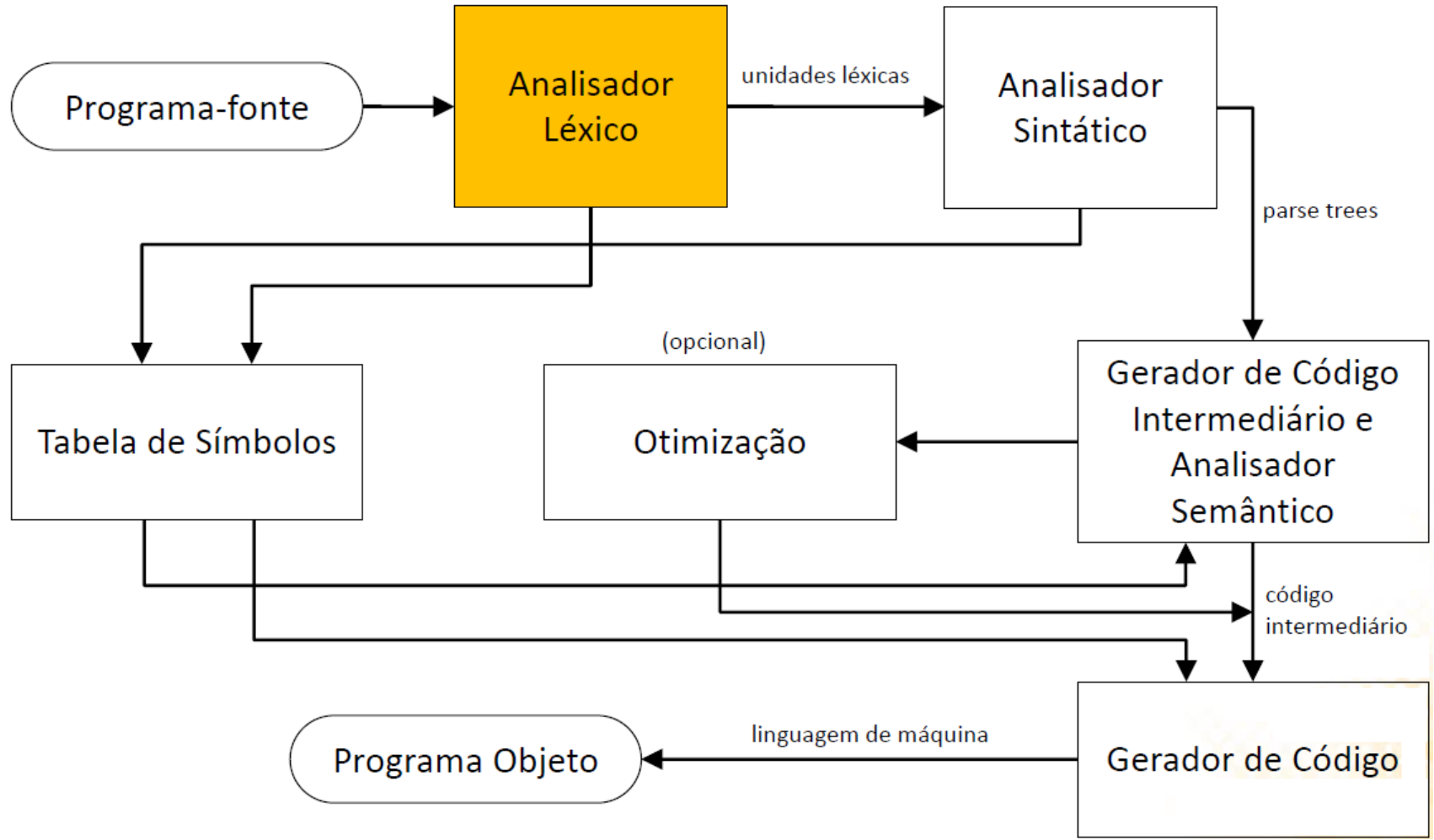
- O compilador vê o código como uma sequência de caracteres:**

```
i  n  t  s p  g  c  d  (  i  n  t  s p  a  ,  s p  i
n  t  s p  b  )  n l  {  n l  s p  s p  w  h  i  l  e  s p
(  a  s p  !  =  s p  b  )  s p  {  n l  s p  s p  s p  s p  i
f  s p  (  a  s p  >  s p  b  )  s p  a  s p  -  =  s p  b
;  n l  s p  s p  s p  s p  e  l  s  e  s p  b  s p  -  =  s p
a  ;  n l  s p  s p  }  n l  s p  s p  r  e  t  u  r  n  s p
a  ;  n l  }  n l
```





Processo de Compilação





Processo de Compilação

- O **Analizador Léxico** reúne os caracteres do programa-fonte em unidades léxicas (*tokens*).
 - Identificadores, palavras especiais, operadores, símbolos;
 - Comentários são ignorados;

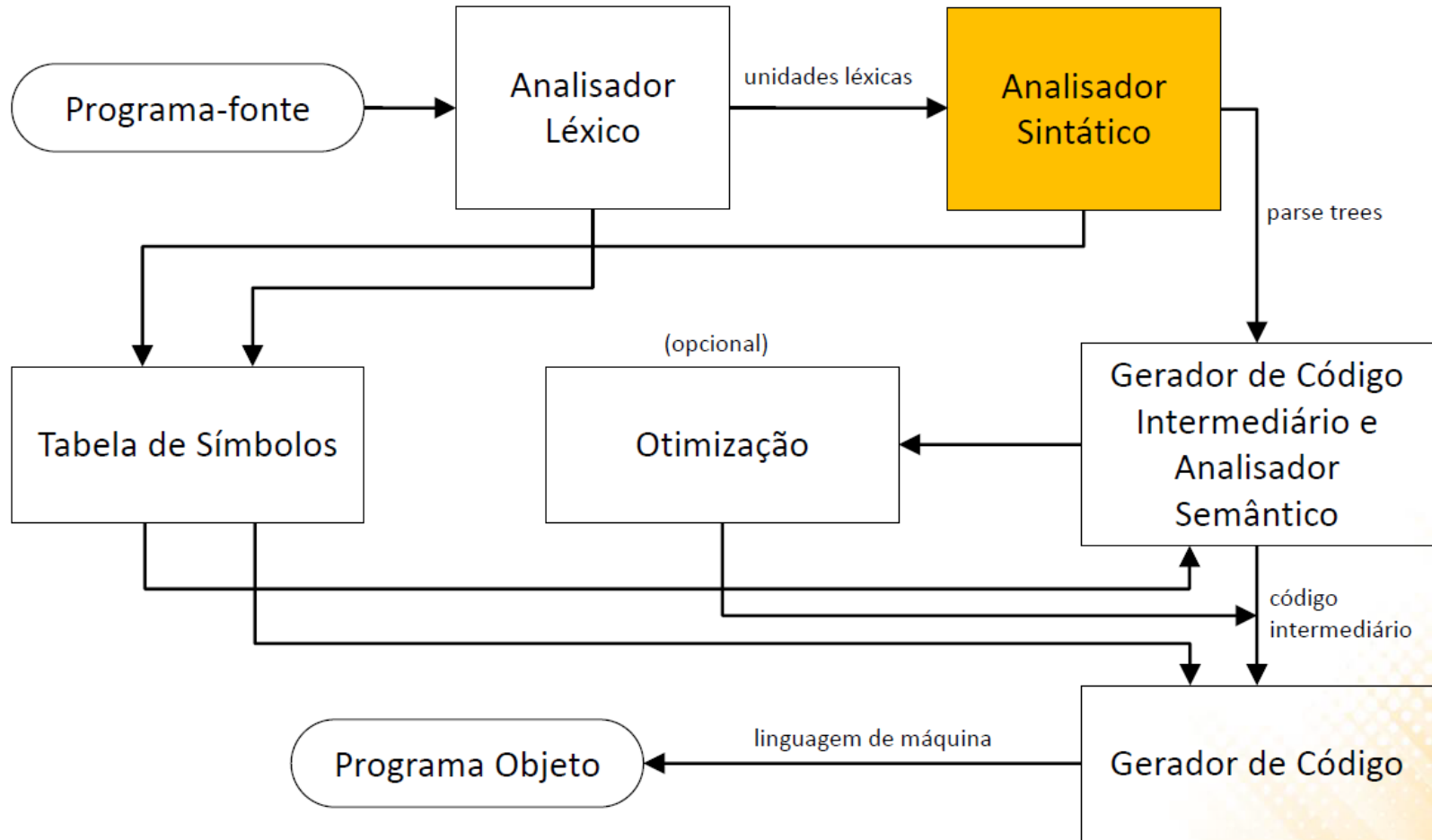
```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

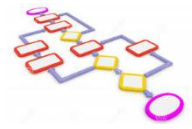
int	gcd	(int	a	,	int	b)	{	while	(a		
!=	b)	{	if	(a	>	b)	a	-=	b	;	else
b	-=	a	;	}	return	a	;	}						





Processo de Compilação

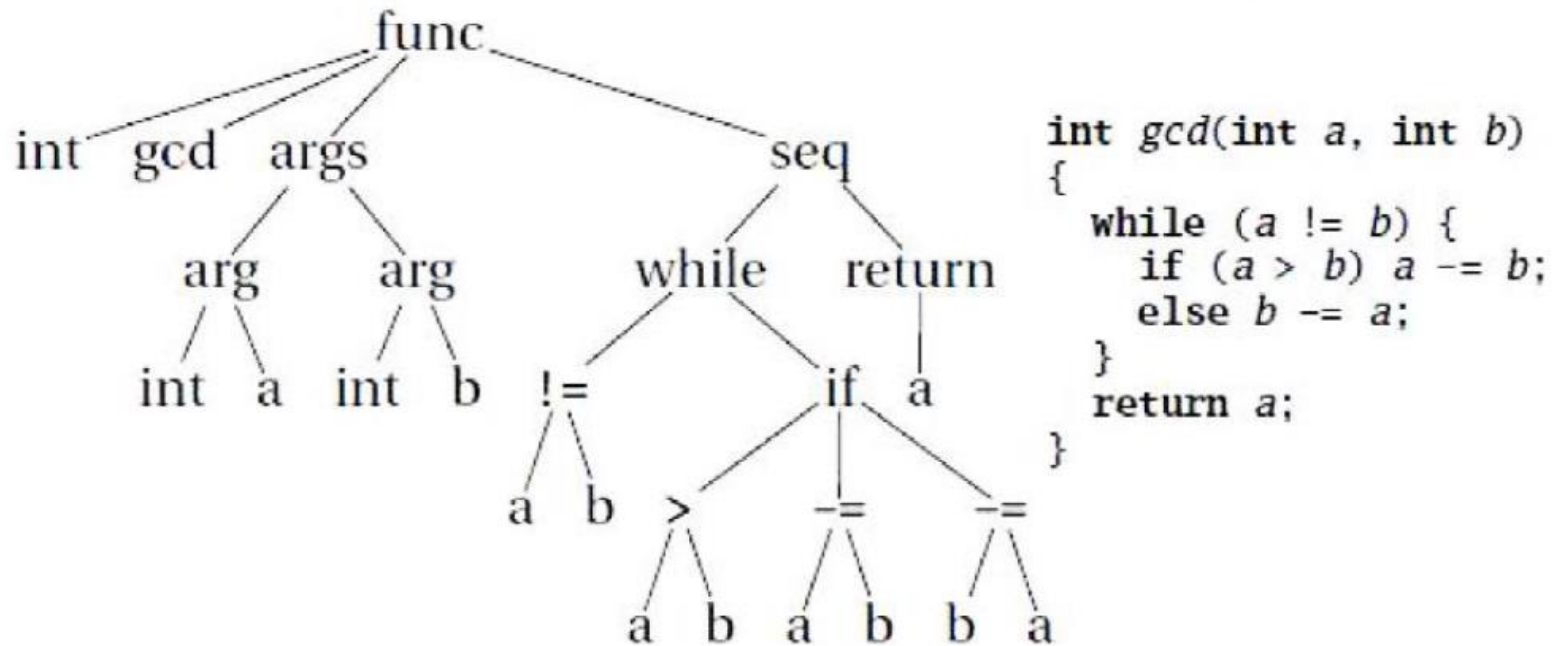




Processo de Compilação

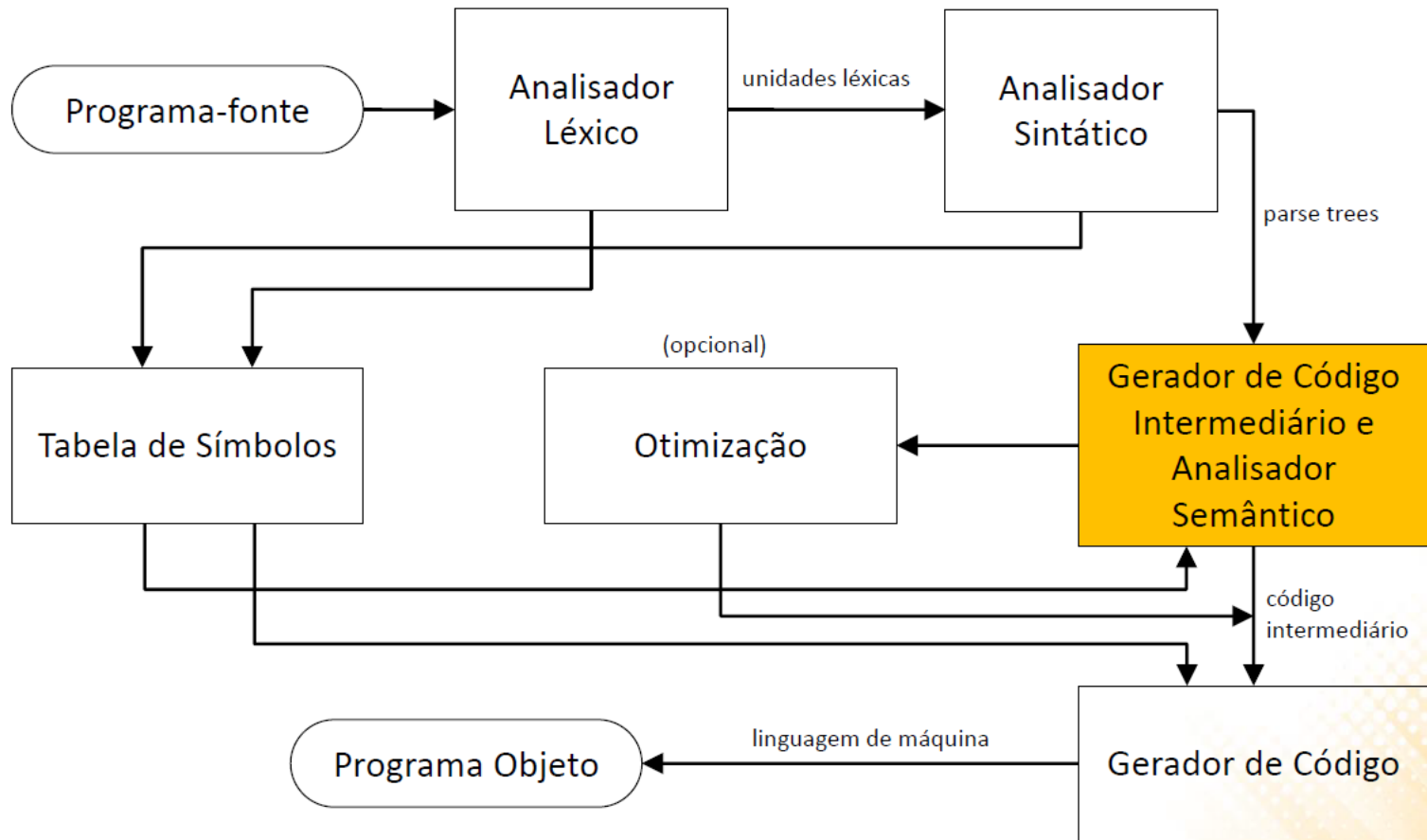
O **Analizador Sintático** utiliza as unidades léxicas (*tokens*) para construir estruturas hierárquicas chamadas de *parse trees*.

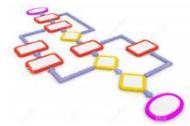
- A árvore é construída a partir das regras da **gramática**;
- Representam a estrutura sintática do programa.





Processo de Compilação

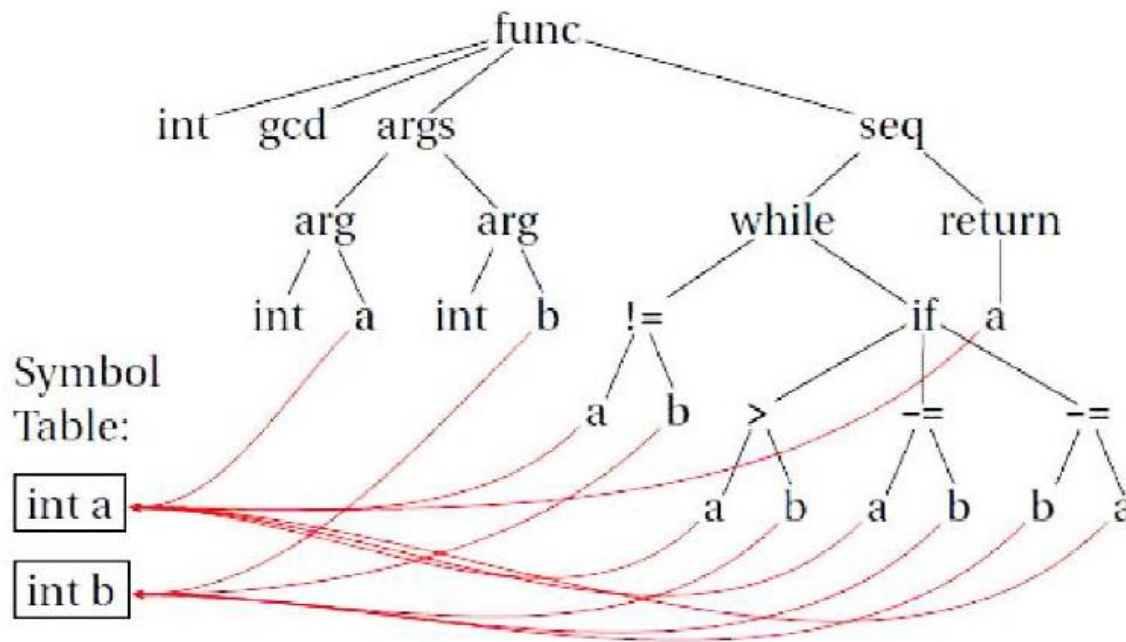


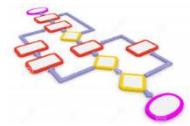


Processo de Compilação

O **Analizador Semântico** faz parte do Gerador de Código Intermediário e realiza a verificação de erros que não são verificados durante a análise sintática, como erros de tipo.

- O processo de verificação de tipos é realizado consultando a **Tabela de Símbolos**.





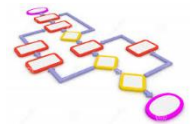
Processo de Compilação

O **Gerador de Código Intermediário** produz um programa em uma linguagem intermediária entre o programa-fonte e saída final do compilador.

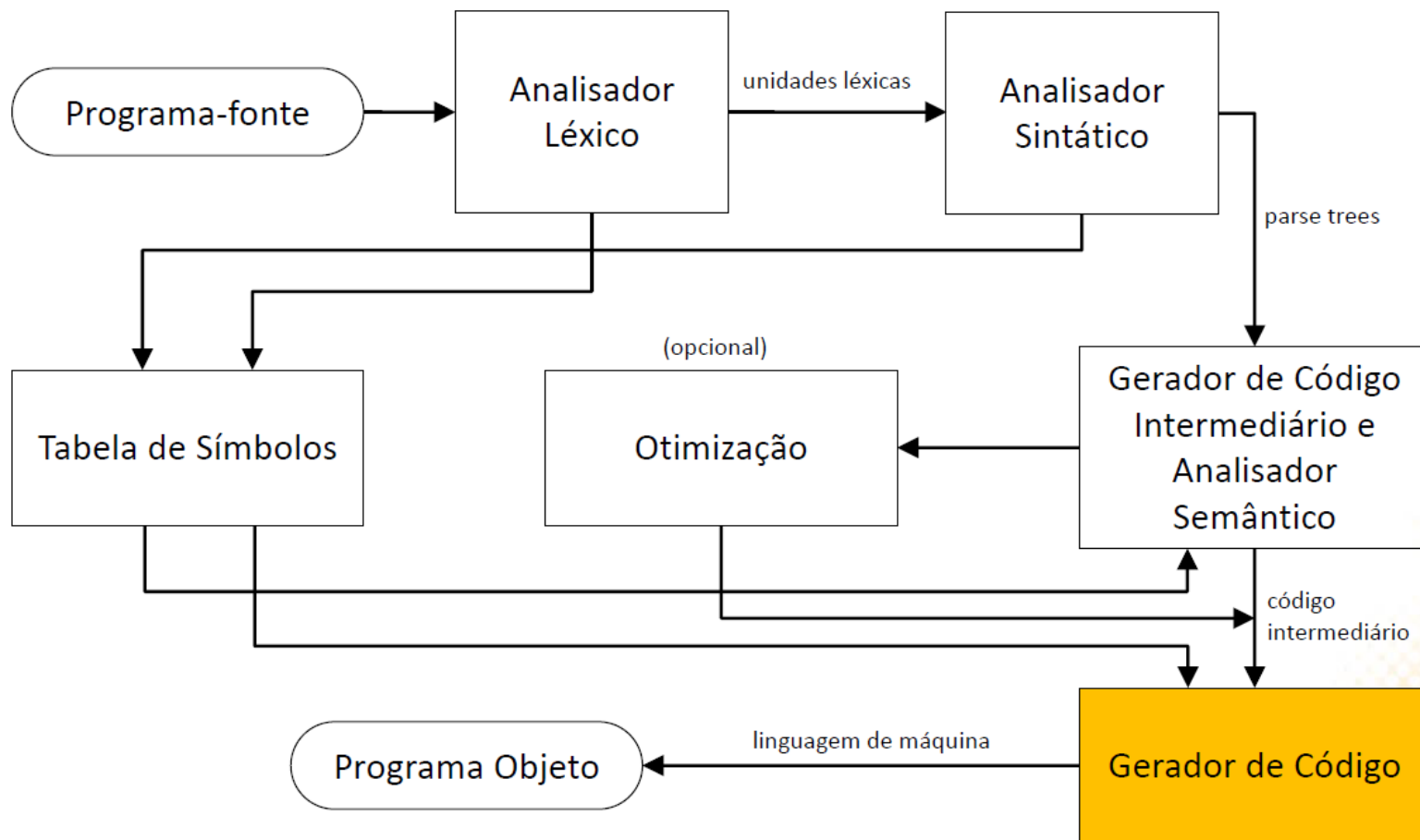
- As linguagens intermediárias se parecem muito com linguagens de montagem (e muitas vezes são linguagens de montagem (assembly)).

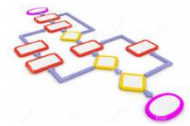
```
gcd:  pushl %ebp                % Save FP
      movl %esp,%ebp
      movl 8(%ebp),%eax        % Load a from stack
      movl 12(%ebp),%edx       % Load b from stack
.L8:  cmpl %edx,%eax
      je   .L3                % while (a != b)
      jle  .L5                % if (a < b)
      subl %edx,%eax          % a -= b
      jmp  .L8
.L5:  subl %eax,%edx          % b -= a
      jmp  .L8
.L3:  leave                   % Restore SP, BP
      ret
```





Processo de Compilação





Processo de Compilação

O **Gerador de Código** converte a versão do código intermediário do programa para um programa em linguagem de máquina equivalente.

```
gcd:  pushl %ebp
      movl %esp,%ebp
      movl 8(%ebp),%eax
      movl 12(%ebp),%edx
.L8:  cmpl %edx,%eax
      je   .L3
      jle  .L5
      subl %edx,%eax
      jmp  .L8
.L5:  subl %eax,%edx
      jmp  .L8
.L3:  leave
      ret
```

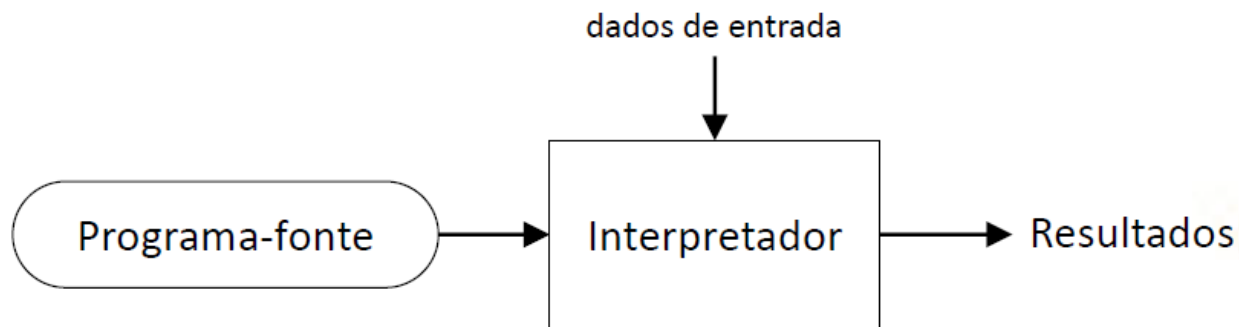
Assembly Language	Machine Code
add \$t1, \$t2, \$t3	04CB: 0000 0100 1100 1011
addi \$t2, \$t3, 60	16BC: 0001 0110 1011 1100
and \$t3, \$t1, \$t2	0299: 0000 0010 1001 1001
andi \$t3, \$t1, 5	22C5: 0010 0010 1100 0101
beq \$t1, \$t2, 4	3444: 0011 0100 0100 0100
bne \$t1, \$t2, 4	4444: 0100 0100 0100 0100
j 0x50	F032: 1111 0000 0011 0010
lw \$t1, 16(\$s1)	5A50: 0101 1010 0101 0000
nop	0005: 0000 0000 0000 0101
nor \$t3, \$t1, \$t2	029E: 0000 0010 1001 1110
or \$t3, \$t1, \$t2	029A: 0000 0010 1001 1010
ori \$t3, \$t1, 10	62CA: 0110 0010 1100 1010
ssl \$t2, \$t1, 2	0455: 0000 0100 0101 0101
srl \$t2, \$t1, 1	0457: 0000 0100 0101 0111
sw \$t1, 16(\$t0)	7050: 0111 0000 0101 0000
sub \$t2, \$t1, \$t0	0214: 0000 0010 0001 0100





Interpretação Pura

- Na extremidade oposta dos métodos de implementação, os programas podem ser interpretados por outro programa chamado **Interpretador**.
- O interpretador age como uma **simulação de software de uma máquina** cujo ciclo buscar-executar lida com instruções de programa em linguagem de alto nível em vez de instruções de máquina.





Interpretação Pura

- O método de interpretação pura tem a séria desvantagem de que a execução é de **10 a 100 vezes mais lenta** que em sistemas compilados.
 - **Problema:** a decodificação das instruções de alto nível é bem mais complexa do que as instruções em linguagem de máquina.
- A interpretação é um processo complexo em programas escritos em linguagens complexas, pois o significado de cada instrução deve ser determinado diretamente do programa-fonte em tempo de execução.
 - Linguagens mais simples podem ser interpretadas mais facilmente. Exemplos: Lisp, Lua, JavaScript...





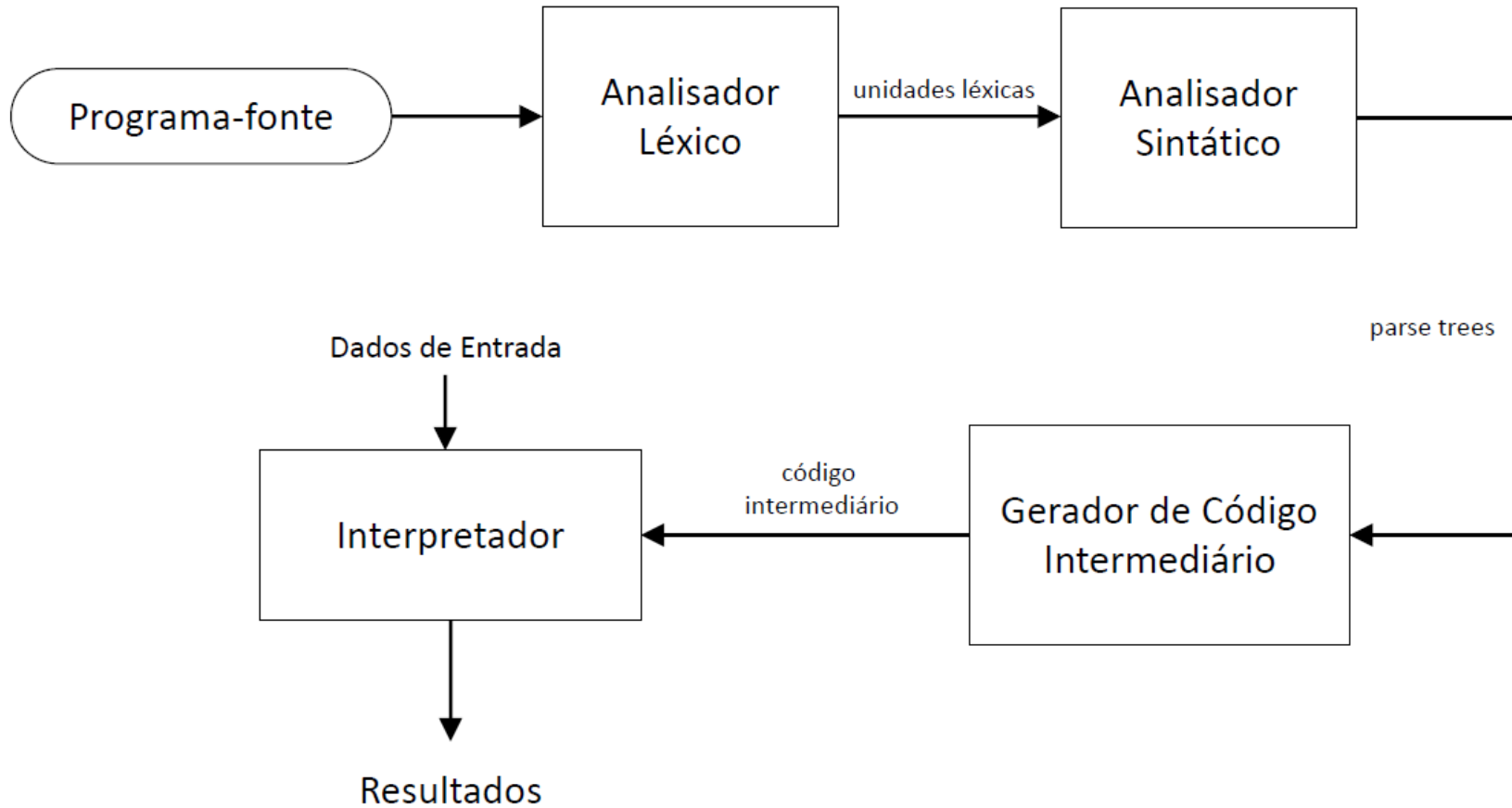
Métodos Híbridos

- Alguns métodos de implementação são um meio-termo entre os compiladores e os interpretadores puros.
- Eles traduzem programas em linguagem de alto nível para uma **linguagem intermediária** projetada para permitir fácil interpretação.
 - Esse método é mais rápido do que a interpretação pura porque as instruções da linguagem fonte são decodificadas somente uma vez.
- Essas implementações são chamadas de **sistemas de implementação híbridos**.





Métodos Híbridos





Métodos Híbridos

Exemplo: JAVA

