COMPILADORES

CONCEITOS GERAIS

LINGUAGENS DE PROGRAMAÇÃO

LINGUAGENS DE PROGRAMAÇÃO

Definição: Meio de Comunicação entre o programador e o computador

Fornece ligação entre o pensamento humano é a precisão requerida pela máquina;

Linguagens de Alto Nível (LP)
Linguagens de Baixo Nível / Linguagem de Máquina (LM)

LP →COMPILADOR → LM

EVOLUÇÃO DAS LIGUAGENS

LINGUAGEM DE MÁQUINA

- Usa 0 e 1
- Problemas: difícil de ler, escrever, modificar e fortemente sujeito a erros;

LINGUAGEM DE MONTAGEM

- Oferece uma notação simbólica, usando nomes ao invés de código binário

b = a + 2; (linguagem assemble)

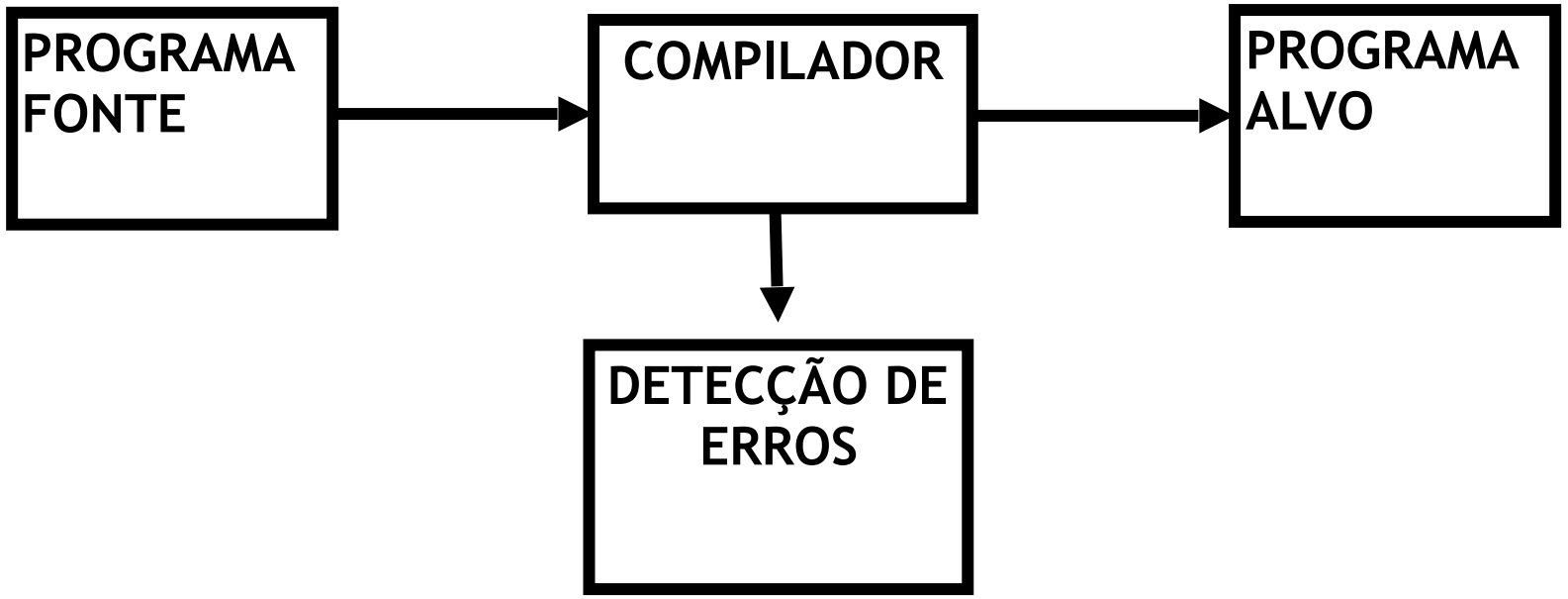
```
MOV    a, R1
ADD    #2, R1
MOV    R1, b
```

LINGUAGEM DE ALTO NÍVEL

O COMPILADOR

COMPILADOR

Def.: Programa que lê um programa escrito em uma linguagem fonte e traduz para uma linguagem alvo;



HISTÓRICO

Os primeiros compiladores começaram a surgir na década de 50. É difícil fornecer uma data exata para o primeiro compilador.

Ao logo dos anos 50 os compiladores foram considerados programas difíceis de escrever.

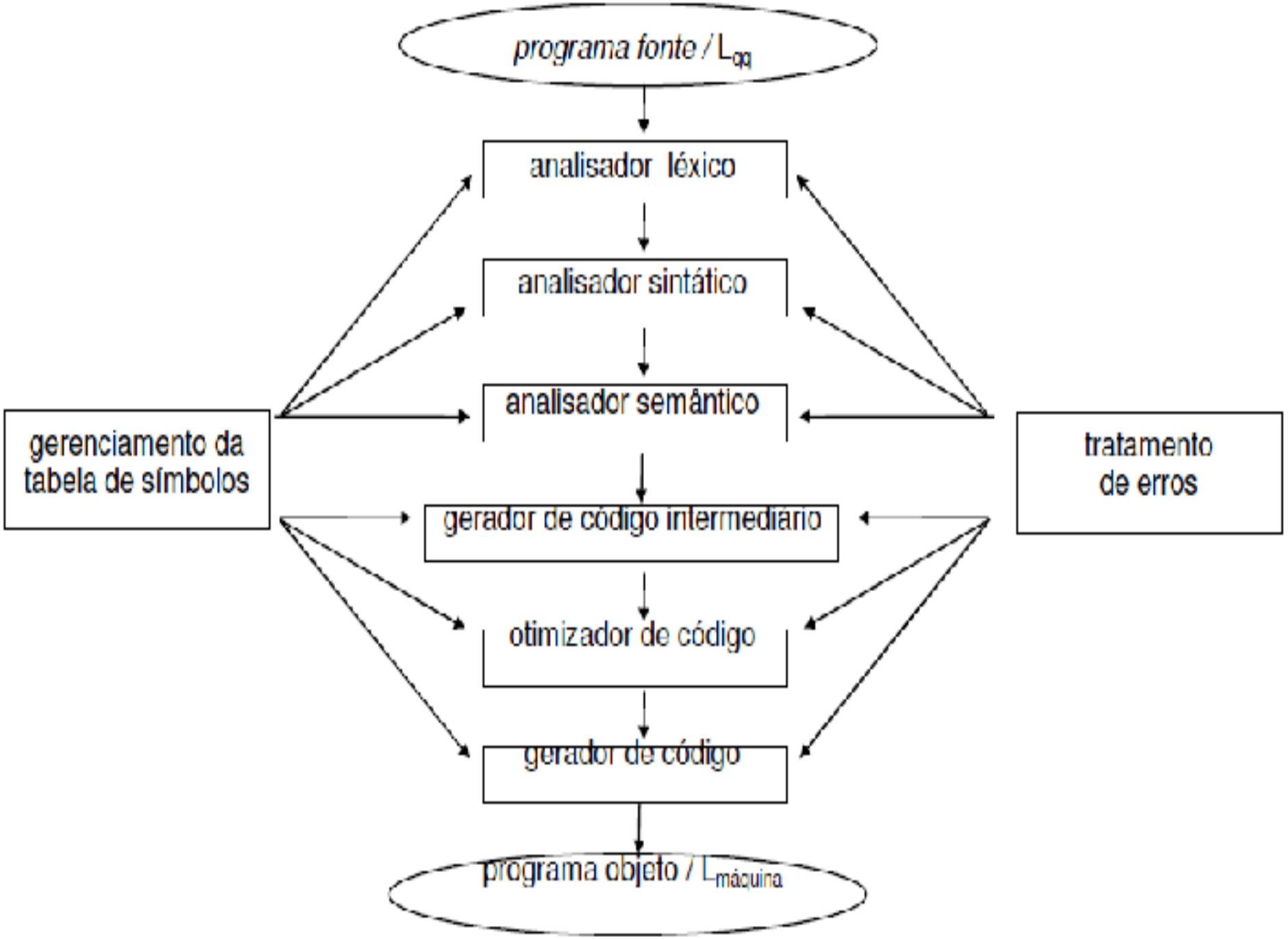
O primeiro compilador do Fortran, por exemplo, consumiu 18 homens-ano para implementar.

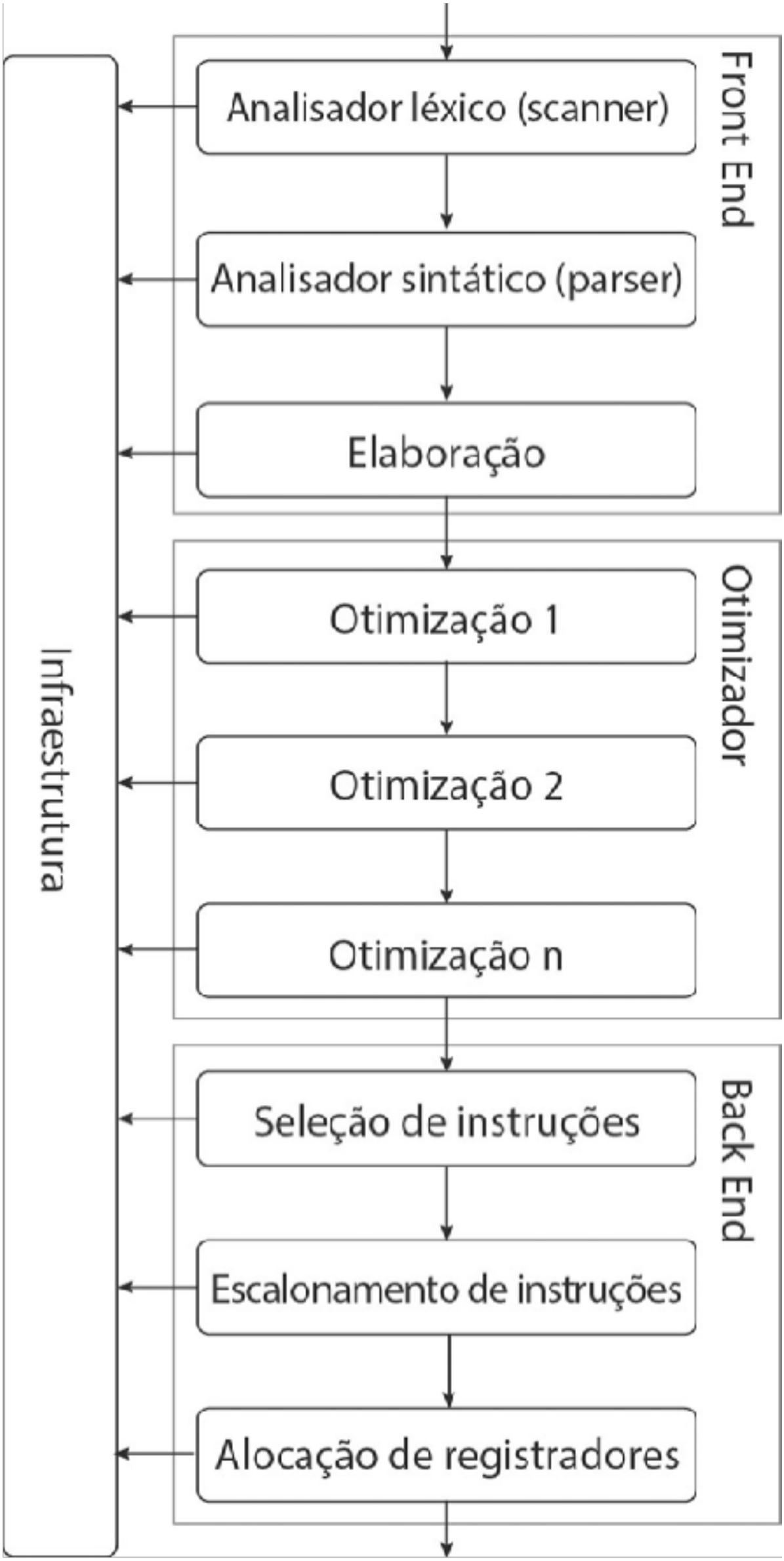
FASES DO COMPILADOR

Conceitualmente, o compilador opera em fases.

Cada uma das quais transforma o programa fonte de uma representação para outra.

Na prática, algumas fases podem ser agrupadas e a representação intermediária entre as mesmas não precisa ser explicitamente construída.


programa fonte / L_{pg}
analisador léxico
analisador sintático
analisador semântico
gerenciamento da tabela de símbolos
tratamento de erros
gerador de código intermediário
otimizador de código
gerador de código
programa objeto / $L_{máquina}$


Front End
Analisador léxico (scanner)
Analisador sintático (parser)
Elaboração
Otimizador
Otimização 1
Otimização 2
Otimização n
Back End
Seleção de instruções
Escalonamento de instruções
Alocação de registradores
Infraestrutura

FASES DO COMPILADOR ANÁLISE E SINTESE

***ANÁLISE (front-end)**: divide o programa fonte em partes constituintes e cria uma representação intermediária do mesmo;*

***SINTESE (back-end)**: constrói o programa alvo a partir da representação intermediária.*

Princípios fundamentais da compilação

- **O primeiro é inviolável:**
  - *O compilador deve preservar o significado do programa a ser compilado.*

- **O segundo princípio a ser observado é prático:**
  - *O compilador deve melhorar o programa de entrada de alguma forma perceptível.*

PRIMOS DO COMPILADOR

INTERPRETAÇÃO PURA

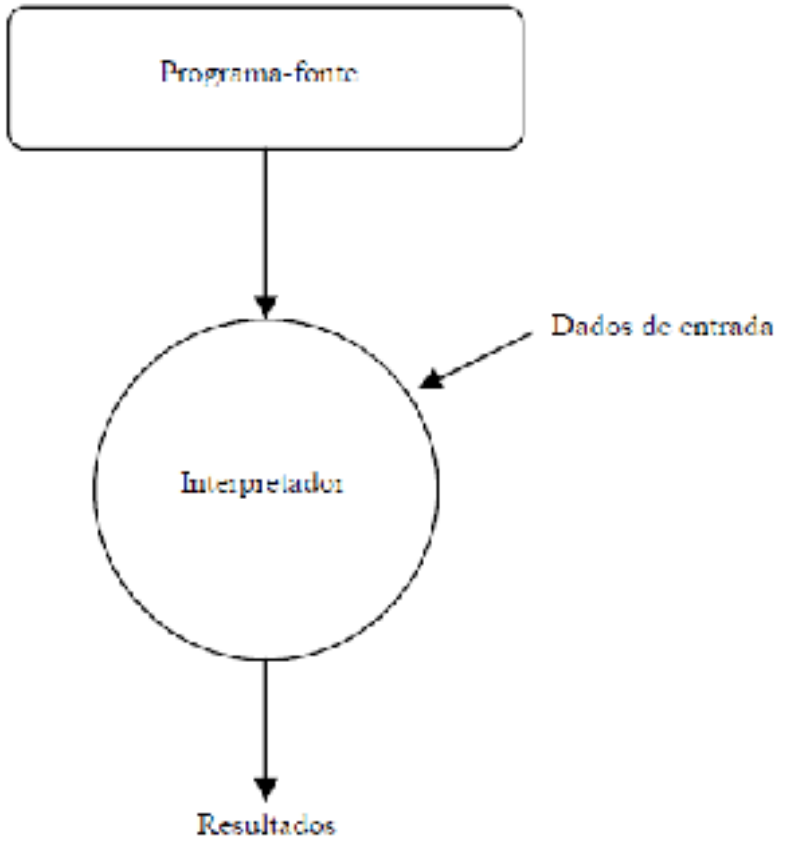
- *Os programas podem ser traduzidos por outro programa chamado interpretador, sem nenhuma conversão.*

- *O programa 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

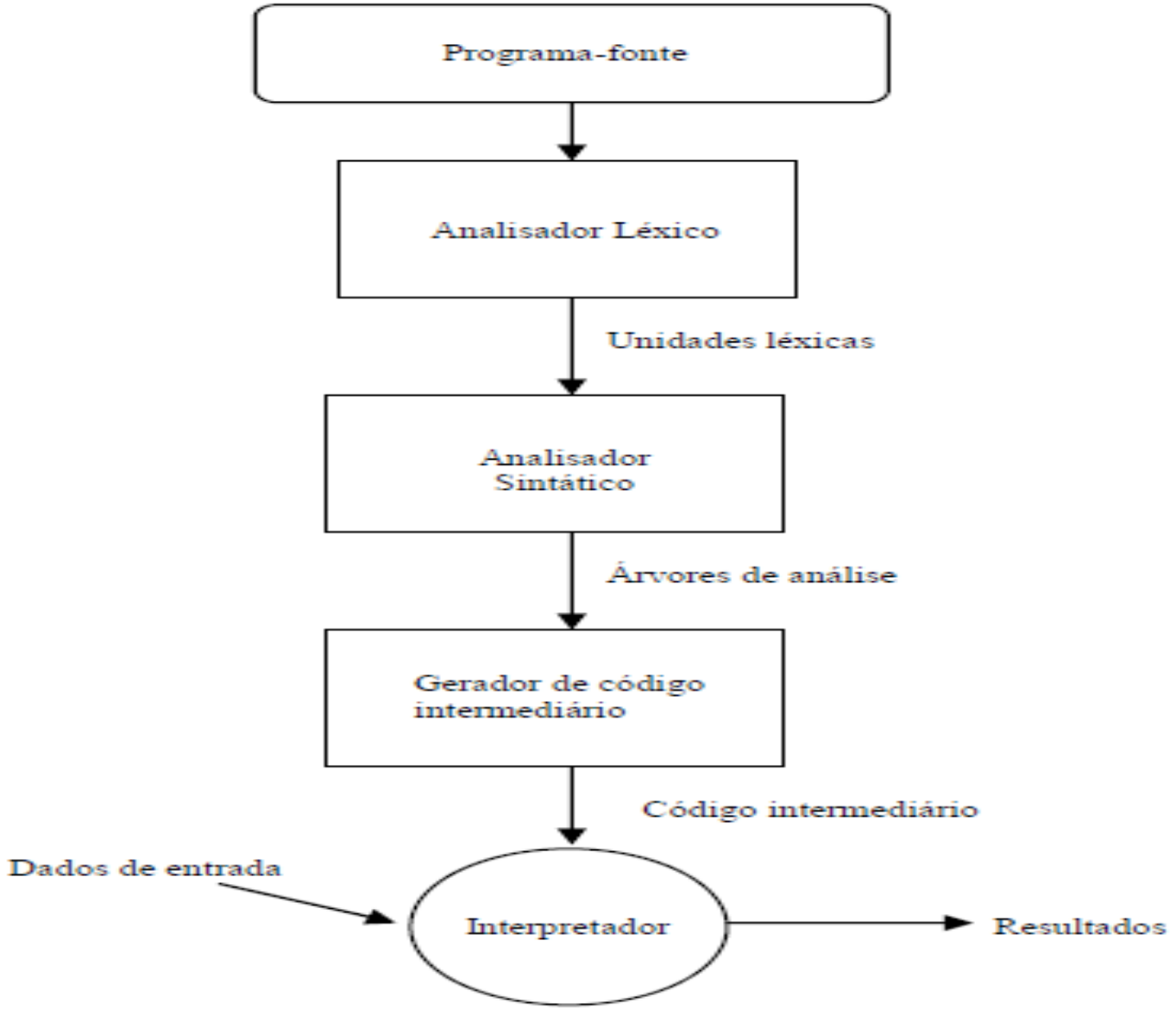
VANTAGENS
- Fácil implementação
- Fácil depuração

DESVANTAGEM
- Execução 10 a 100 vezes mais lenta
- Exige mais espaço em memória



SISTEMAS DE INTERPRETAÇÃO HÍBRIDOS

- 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.
- *Essa simulação de software, evidentemente, fornece uma máquina virtual para a linguagem.*


Programa-fonte
Analisador Léxico
Unidades léxicas
Analisador Sintático
Árvores de análise
Gerador de código intermediário
Código intermediário
Dados de entrada
Interpretador
Resultados

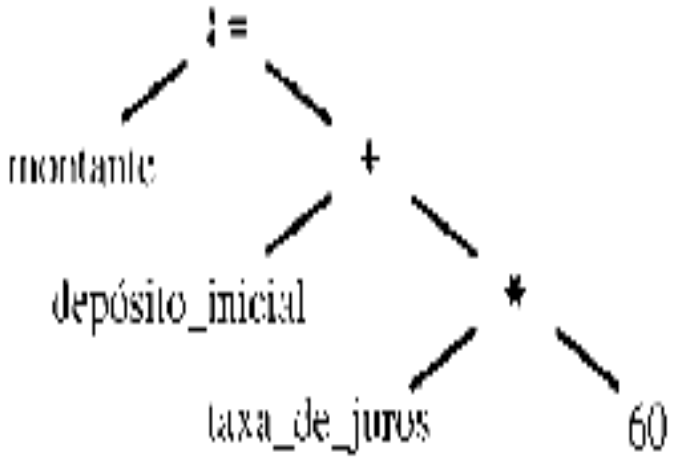
TRADUTORES

- *Muitos “Compiladores” de pesquisa produzem programa C como saída;*
- *Como existem compiladores para C na maioria dos computadores, isso torna o programa-alvo executável em todos estes sistemas ao custo de uma compilação extra.*
- Compiladores que visam uma linguagem de programação são chamados de tradutores fonte-a-fonte;

ANÁLISE

As operações são determinadas e registradas em uma estrutura hierárquica chamada **árvore sintática.**

Cada nó representa uma operação e cada filho argumentos da operação.



Árvore sintática para **montante := depósito_inicial + taxa_de_juros * 60**;

ANÁLISE DO PROGRAMA FONTE

Na compilação a análise consiste em 3 fases:

- **ANÁLISE LINEAR (LÉXICA)** – O programa e lido (escaneado) e agrupado em tokens, que são sequência de caracteres com significado coletivo;
- **ANÁLISE HIERÁRQUICA (SINTÁTICA) -** agrupa os tokens fornecidos pelo analisador léxico em estruturas sintáticas (baseado na gramática da linguagem);
- **ANÁLISE SEMÂNTICA –** certas verificações são realizadas para garantir que os componentes de um programa se combinem de forma significativa;

ANÁLISE LÉXICA

ANÁLISE LÉXICA

- *É também chamada de análise linear ou esquadrilhamento (scanning);*
- *O **analisador léxico** separa a seqüência de caracteres que representa o programa fonte em entidades ou tokens, símbolos básicos da linguagem.*
- *Durante a análise léxica, os tokens são classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (inteiro real, literal, etc.), entre outras categorias.*

EXEMPLO:

Em **montante := depósito_inicial + taxa_de_juros * 60,** poderiam ser agrupados os seguintes tokens:

- **O identificador** montante
- **O símbolo de atribuição** :=
- **O identificador** depósito_inicial
- **O sinal de adição** +
- **O identificador** taxa_de_juros
- **O sinal de multiplicação** *
- **O número** 60

EXEMPLO: Em SOMA := SOMA + 35, os tokens podem ser agrupados, pelo analisador léxico, em 5 entidades:

(VALOR)	(CLASSE)
SOMA	identificador
:=	Comando de atribuição
SOMA	Identificador
+	operador aritmético de adição
35	constante numérica inteira

ANÁLISE LÉXICA

Um *token* consiste de um par ordenado (valor, classe). A *classe* indica a natureza da informação contida em *valor*.

Outras funções atribuídas ao analisador léxico são: ignorar espaços em branco e comentários, e detectar erros léxicos.

ANÁLISE SINTÁTICA

ANÁLISE SINTÁTICA

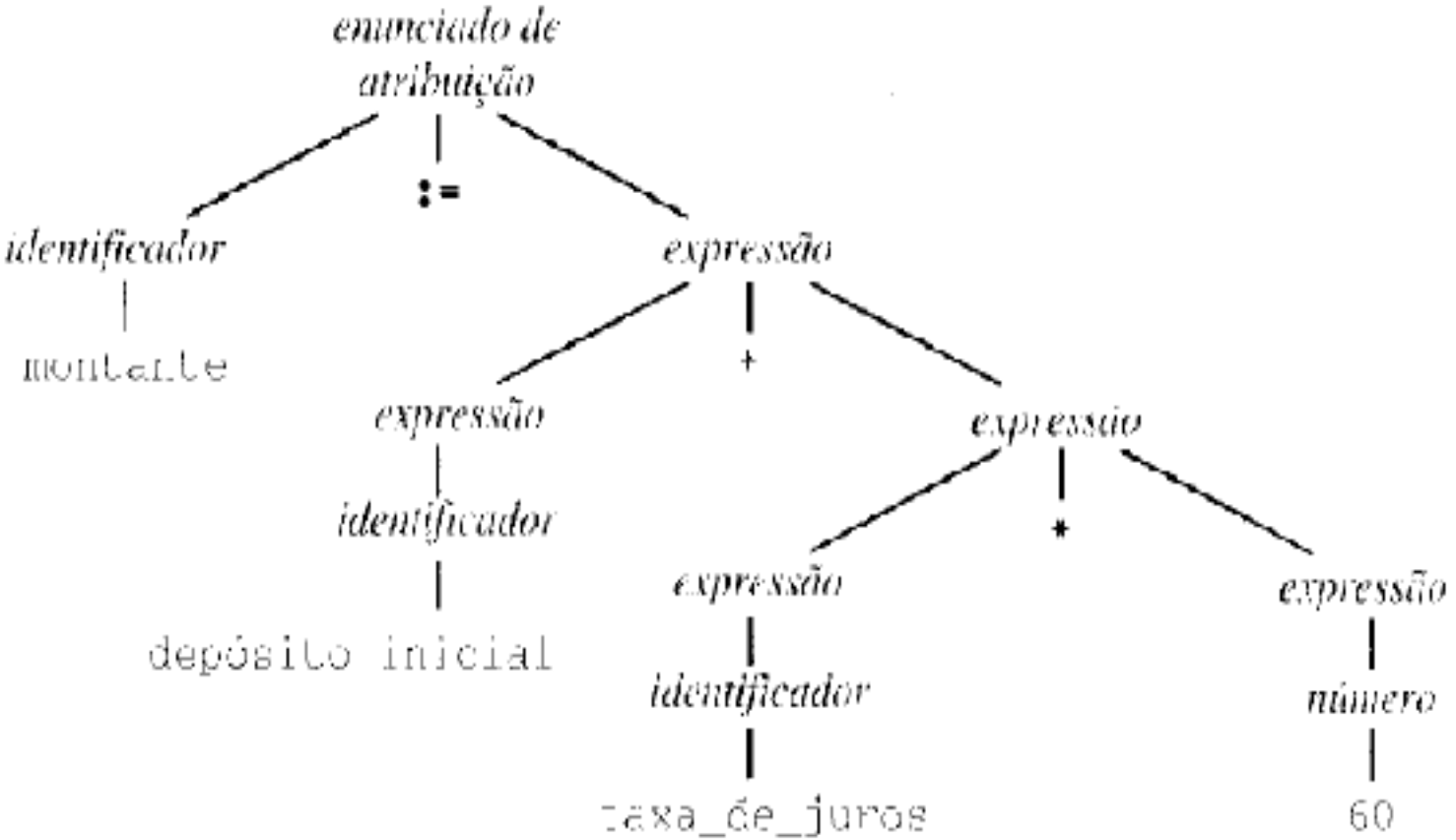
- *Também chamado de análise hierárquica ou análise gramatical.*
- *O **analisador sintático** agrupa os tokens fornecidos pelo analisador léxico em estruturas sintáticas, **construindo a árvore sintática** correspondente.*
- *Para isso, utiliza uma série de regras de sintaxe, que constituem a gramática da linguagem fonte.*
- *É a gramática da linguagem que define a estrutura sintática do programa fonte.*
  - Matematicamente, a linguagem de origem é um conjunto, normalmente infinito, de strings definido por algum conjunto finito de regras, chamado *gramática*;

ANÁLISE SINTÁTICA

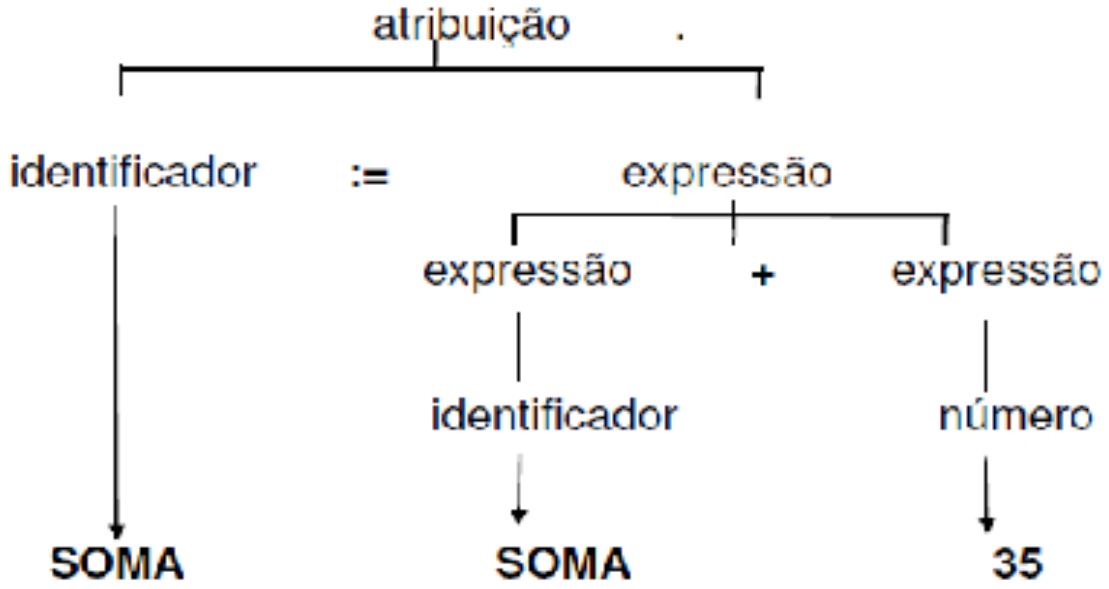
- *O analisador sintático tem também por tarefa o reconhecimento de erros sintáticos.*

- *Esses erros são construções do programa fonte que não estão de acordo com as regras de formação de estruturas sintáticas como especificado pela gramática.*

Exemplo de árvores gramaticais:



Exemplo de árvores gramaticais:



ANÁLISE SINTÁTICA

A divisão entre análise léxica e a sintática é um tanto arbitrária.

Um fator determinante na divisão é o de uma construção da linguagem fonte ser inerentemente recursiva ou não.

As construções léxicas não requerem recursão, enquanto as sintáticas frequentemente as exigem.

ANÁLISE SEMÂNTICA

ANÁLISE SEMÂNTICA

A fase de análise semântica **verifica os erros semânticos** no programa fonte e **captura as informações de tipo** para fase subseqüente de geração de código.

Um importante componente da análise semântica é a **verificação de tipos.**

O analisador semântico utiliza a árvore sintática determinada pelo analisador sintático para:

- **Identificar operadores e operandos das expressões;**
- **Reconhecer erros semânticos;**
- **Fazer verificações de compatibilidade de tipo;**
- **Analisar o escopo das variáveis, etc.**

ANÁLISE SEMÂNTICA

Por exemplo, para o comando de atribuição SOMA:= SOMA + 35, é necessário fazer a seguinte análise:

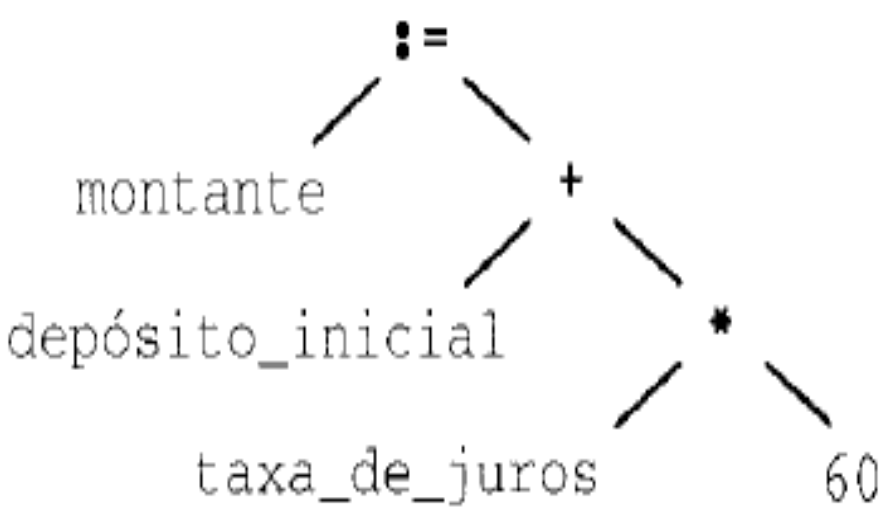
- o identificador SOMA foi declarado? em caso negativo, erro semântico.
- o identificador SOMA é uma variável? em caso negativo, erro semântico.
- qual o escopo da declaração da variável SOMA: local ou global?
- qual o tipo da variável SOMA? o valor atribuído no lado direito do comando de atribuição é compatível?

ANÁLISE SEMÂNTICA

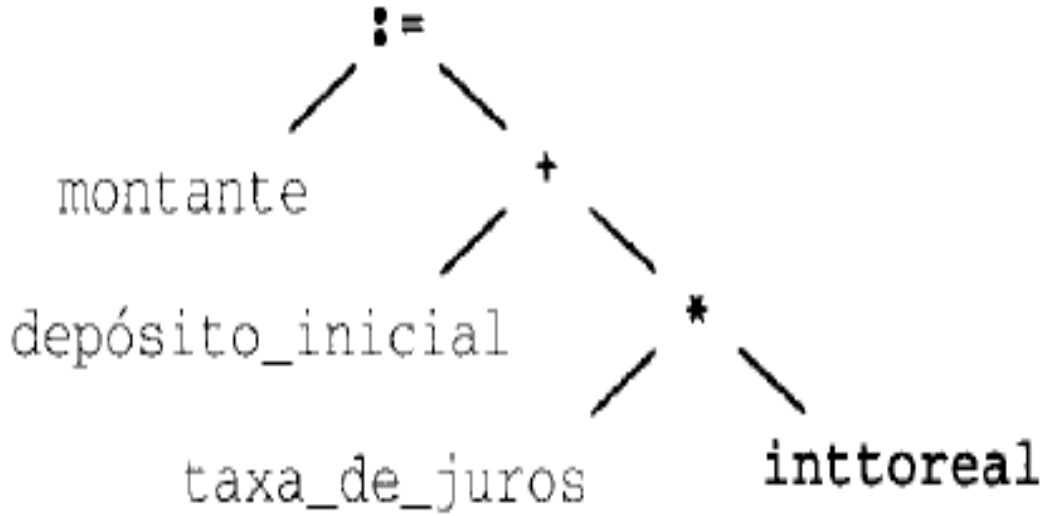
Fundamentalmente, a análise semântica trata os aspectos sensíveis ao contexto da sintaxe das linguagens de programação.

Por exemplo, não é possível representar em uma gramática livre de contexto uma regra como "Todo identificador deve ser declarado antes de ser usado.", e a verificação de que essa regra foi aplicada cabe à análise semântica.

Suponha por exemplo que todos os identificadores da fig. (a) tenham sido declarados com reais e o número 60, por si só, seja um inteiro. O analisador semântico pode fazer correções de tipo.



(a)



(b)

TABELA DE SÍMBOLOS

GERENCIAMENDO DA TABELA DE SÍMBOLOS

- Uma função essencial em um compilador é registrar os identificadores usados no programa fonte e coletar as informações sobre os seus diversos atributos.

- Atributos:
  - tipo,
  - escopo,
  - memória reservada,
  - em caso de procedimento, número e tipo de argumentos, tipo retornado, etc.

GERENCIAMENDO DA TABELA DE SÍMBOLOS

- *Uma tabela de símbolos é uma estrutura de dados contendo um registro para cada identificador, com os campos contendo os atributos do identificador.*

- *A estrutura de dados nos permite encontrar rapidamente cada registro e, igualmente, armazenar ou recuperar dados do mesmo.*

GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

Após as análises sintática e semântica, alguns compiladores geram uma representação intermediária explicita do programa fonte.

*Uma representação intermediária é um código para uma máquina abstrata e deve ser **fácil de produzir e traduzir** no programa objeto.*

*Por exemplo, pode ser usada como forma intermediária o **código de três endereços** (AHO et. al., 1995). O código de três endereços consiste em uma seqüência de instruções, cada uma possuindo no máximo três operandos.*

GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

montante := depósito_inicial + taxa_de_juros *60, tem se:

temp1 := inttoreal (60)

temp2 := id3 * temp1

temp3 := id2 + temp2

id1 := temp3

GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

SOMA:= SOMA + 35, tem-se:

temp1 := 35

temp2 := id1 + temp1

id1:= temp2

OTIMIZAÇÃO DE CÓDIGO

OTMIZAÇÃO DE CÓDIGO

O processo de otimização de código consiste em melhorar o código intermediário de tal forma que o programa objeto resultante seja mais rápido em tempo de execução.

Por exemplo, um algoritmo para geração do código intermediário gera uma instrução para cada operador na árvore sintática, mesmo que exista uma maneira mais otimizada de realizar o mesmo

Otimização - Eficiência

Eficiência pode ter muitos significados.

Notação clássica:

- Reduzir o tempo de execução;

Outros Contextos:

- Diminuir uso de memória;
- Reduzir tamanho do código;
- Reduzir a energia que o processador consome para avaliar um código;

OTMIZAÇÃO DE CÓDIGO

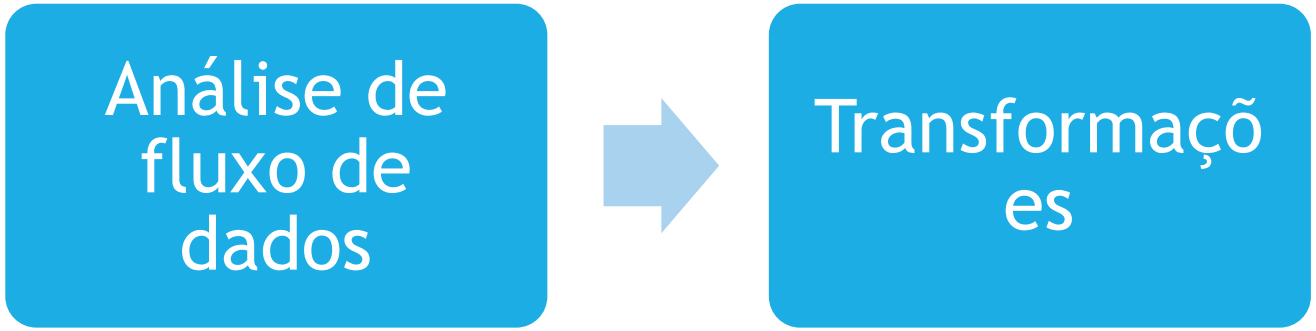
Códigos anteriores otimizados ficam, respectivamente:

```
temp1:= id3 * 60.0
id1 := id2 + temp1
```

```
id1 := id1 + 35
```

OTIMIZAÇÃO

A maioria das otimizações consiste em uma análise e uma transformação.



GERAÇÃO DE CÓDIGO

GERAÇÃO DO CÓDIGO

A fase final do compilador é a geração do código para o programa objeto, consistindo normalmente de código de montagem ou de código em linguagem de máquina.

```
MOVF    id3, R2
MULF    #60.0, R2
MOVF    id2, R1
ADDF    R2, R1
MOV     R1, id1
```

```
MOV     id1, R1
ADD     35, R1
MOV     R1, id1
```

GERAÇÃO DO CÓDIGO ALVO

1. Seleção de Instruções: *reescreve as operações da IR em operações da máquina-alvo;*

2. Alocação de registradores: *durante o processo de seleção o compilador ignora o fato de a máquina-alvo possuir um conjunto finito de registradores. O alocador de registradores precisa mapear esses registradores virtuais para registradores reais;*

3. Escalonador de instruções: *reordena as operações no código, e tenta minimizar o número de ciclos desperdiçados aguardando pelos operandos;*

CONSTRUÇÃO DE COMPILADORES

CONSTRUÇÃO DE COMPILADORES

- A construção de compiladores é um exercício de projeto de engenharia.

- O construtor de compiladores precisa escolher um caminho através de um espaço de projeto que é repleto de alternativas diversas, cada uma com diferentes custos, vantagens e complexidade.

- A qualidade do produto final depende de decisões bem feitas em cada etapa ao longo do caminho.

CONSTRUÇÃO DO COMPILADOR

- Assim, não existe uma única resposta certa para muitas das decisões de projeto em um compilador.

- Até mesmo dentro de problemas “bem entendidos” e “resolvidos” nuances no projeto e implementação têm impacto sobre o comportamento do compilador e a qualidade do código que produz.

- Muitas considerações devem ser feitas em cada decisão.

Um bom compilador contém um microcosmo da ciência da computação.

Faz uso:

- Prático de algoritmos gulosos (alocação de registradores);
- Técnicas de busca heurística (agendamento de lista);
- Algoritmos de grafos (eliminação de código morto);
- Programação dinâmica (seleção de instruções);
- Autômatos finitos e autômatos de pilha (análises léxica e sintática);
- Algoritmos de ponto fixo (análise de fluxo de dados).
- Lida com problemas, como alocação dinâmica, sincronização, nomeação, localidade gerenciamento da hierarquia de memória e escalonamento de pipeline.

CONSTRUÇÃO DE COMPILADORES

A construção de compiladores engloba várias áreas desde teoria de linguagens de programação até engenharia de software, passando por arquitetura de máquina, sistemas operacionais e algoritmos.

Algumas técnicas básicas de construção de compiladores podem ser usadas na construção de ferramentas variadas para o processamento de linguagens

- *Formatadores de texto;*
- *Interpretadores de Queries (consultas a banco de dados);*
- *Pesquisa de texto;*
- *Filtragem de website;*
- *Interpretadores de linguagem de comandos.*

PERSPECTIVAS

Alguns problemas que surgem na construção de compiladores são problemas abertos — isto é, as melhores soluções atuais ainda têm espaço para melhorias.

Tentativas de projeto de representações de alto nível, universais e intermediárias esbarram na complexidade.

POR EXEMPLO: O método dominante para escalonamento de instruções é um algoritmo guloso com várias camadas de heurística de desempate.

TRABALHO DESAFIO

Desenvolva um código que:

- recebe uma expressão simples;
- faça a análise léxica;
- faça a análise sintática;
- e de forma dirigida pela sintaxe transforme em um código ILOC.

Usar:

Flex: gerador de analisador léxico;

Bison: gerador de analisador sintático;

- **ILOC** – pseudocódigo de máquina criado pelo MIT

Apêndice A - Livro Construindo Compiladores.

- **Simplificações:**

1. O compilador deve processar apenas expressões do tipo inteiro;

2. As informações cabem nos registradores:

- Infinitos registradores;
- Com tamanho suficiente;

3. Algumas instruções ILOC podem ser simplificadas;

TRABALHO DESAFIO

REGRAS ADICIONAIS:

- Equipe com no **MÁXIMO** 3 componentes;

- O trabalho será apresentado em 3 etapas após as AP1, AP2 e AP3 respectivamente. Os componentes deverão, obrigatoriamente, reversa na apresentação das etapas. Apenas um componente apresentará por vez.

- Na apresentação final:

1. O código deve ser explicado de maneira detalhada;
2. A aplicação deve compilar e realizar a tradução perfeitamente;