Universidade de Santa Cruz do Sul – UNISC
Departamento de informática

COMPILADORES

Introdução

Geovane Griesang
geovanegriesang@unisc.br

Processadores de linguagem

Linguagens de programação são notações para se descrever computações para pessoas e máquinas.

Então, todos os *softwares* executados em todos os computadores foram escritos em alguma linguagem de programação.

Mas, antes que possa rodar, um programa primeiro precisa ser traduzido p/ um formato que lhe permita ser executado em um computador.

Os compiladores fazem essa tradução.

Processadores de linguagem

"Compilador é um programa de computador que lê um programa escrito em uma linguagem (linguagem fonte) e a traduz em um programa equivalente em outra linguagem (linguagem objeto)".
Aho, Sethi, Ullman.

Função: relatar quaisquer erros no programa fonte detectados durante este processo de tradução.

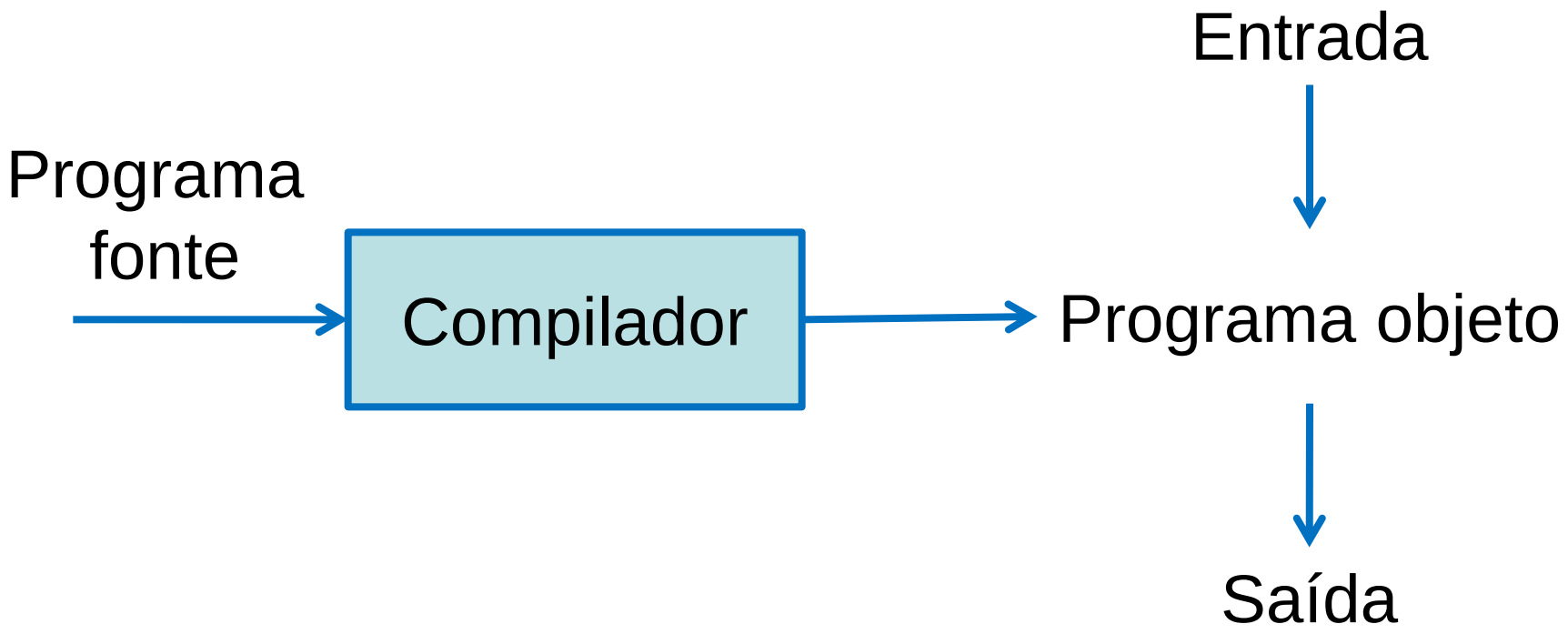
Histórico

Nos anos 50, os compiladores eram programas de difícil escrita, tanto os próprios compiladores quanto os programas fonte.

Aumentou a demanda por linguagens de mais alto nível que linguagens de máquina e *Assember*.

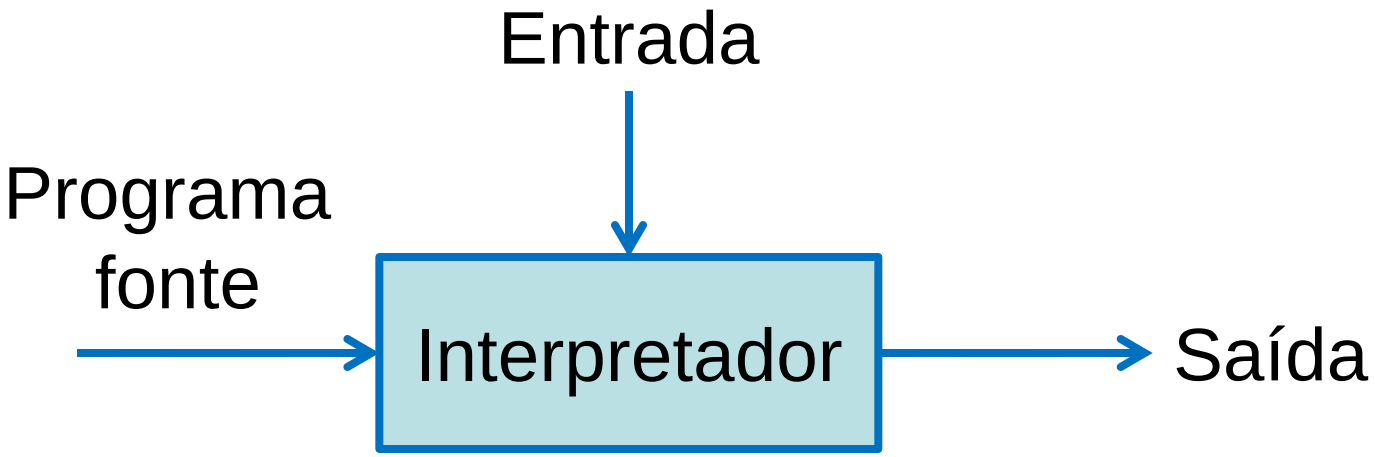
Portanto, com o passar dos anos, o avanço teórico, de técnicas e ferramentas de implementação tornou possível o desenvolvimento compiladores com muito mais facilidade.

Compilador



- ✓ Programa objeto é um programa de máquina executável.
- ✓ Esse programa objeto pode ser chamado pelo próprio usuário para processar entradas e produzir saída.

Interpretador



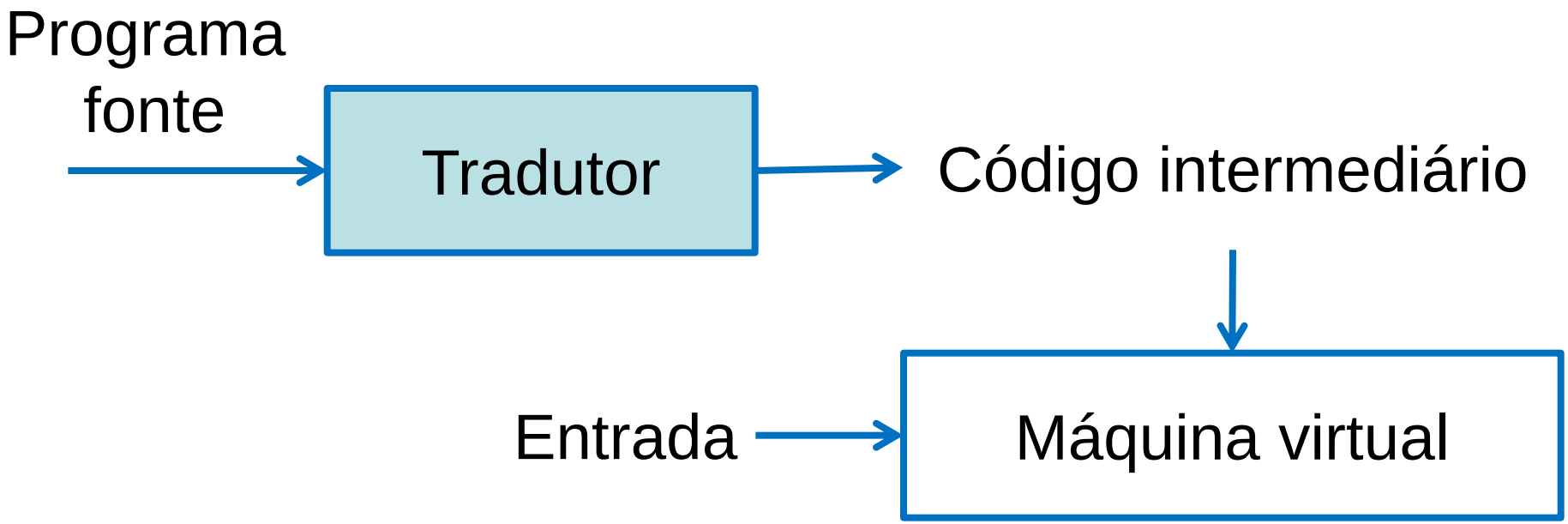
- ✓ Interpretador: tipo comum de processador de linguagem.
- ✓ Em vez de produzir um programa objeto como resultado da tradução, esse interpretador executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário.

Compilador x Interpretador

O programa objeto em uma linguagem de máquina produzido por um compilador normalmente é muito mais rápido no mapeamento de entrada para saídas do que um interpretador.

Porém, um interpretador frequentemente oferece um melhor diagnóstico de erro do que um compilador, pois executa o programa fonte instrução por instrução.

Compilador híbrido



Compilador híbrido

Sendo assim, os processadores da linguagem Java combinam compilação e interpretação.

Um programa em Java pode ser primeiro compilado p/ um forma intermediária, chamada *bytecodes*.

Os *bytecodes* (ou códigos de *bytes*) são interpretados por uma máquina virtual.

Como um benefício dessa combinação, os *bytecodes* compilados em uma máquina podem ser interpretados em outra máquina, talvez por meio de uma rede.

Compilador híbrido

Para conseguir um processamento mais rápido, das entradas para saídas, alguns compiladores Java (*just-in-time)*, traduzem os *bytecodes* para uma linguagem de máquina, imediatamente antes de executarem o programa intermediário para processar a entrada.

- ✓ *Just-in-time*: é um tradutor que converte, em tempo de execução, instruções de um formato para outro, como por exemplo, de *bytecode* para código de máquina.
- ✓ Esta técnica é normalmente utilizada para incrementar o desempenho de programas "executados" - na verdade, interpretados - em máquinas virtuais.

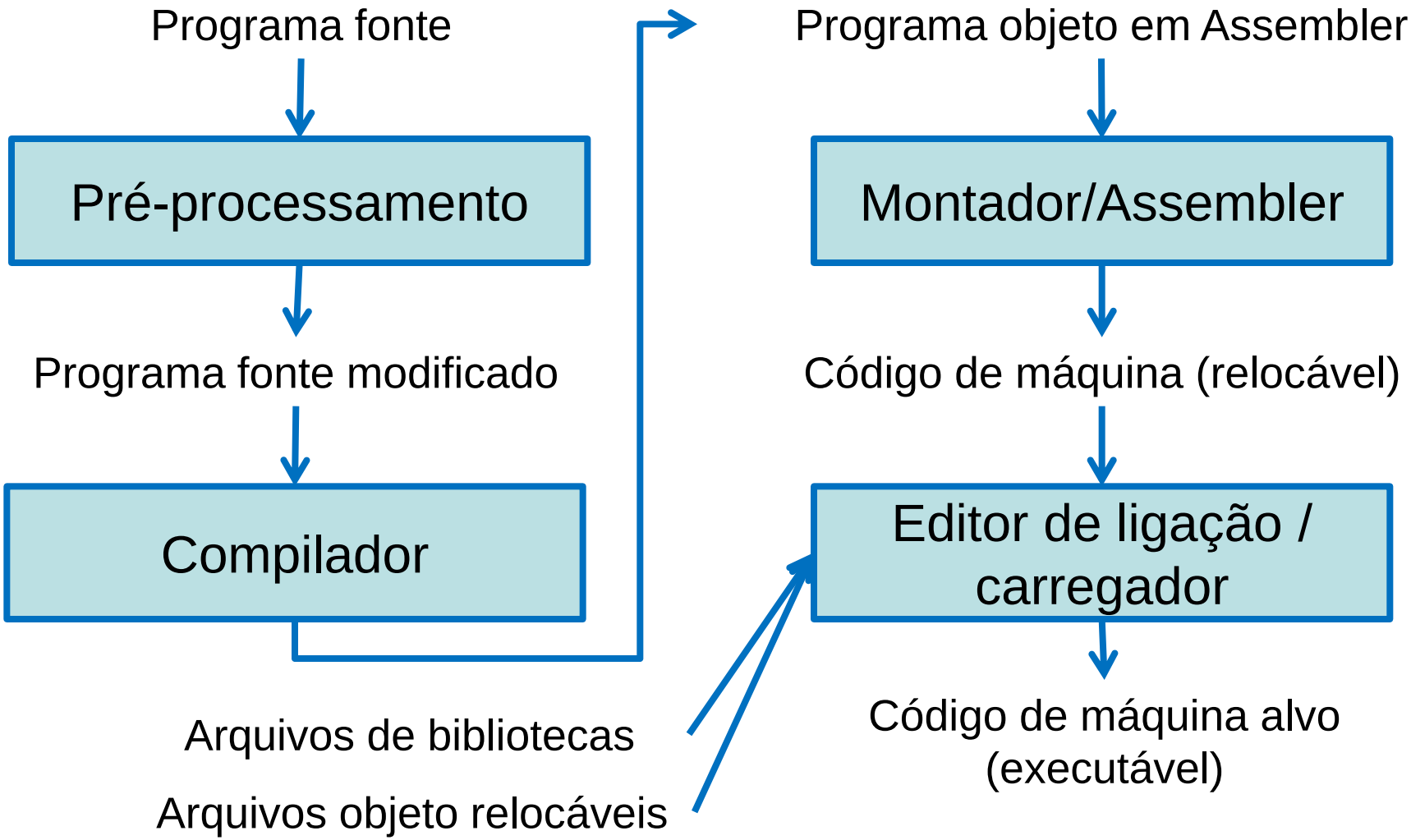
Processo de compilação

Além do compilador, vários outros programas podem ser necessários p/ a criação de um programa objeto executável.

Um programa fonte pode ser subdivido em módulos armazenados em arquivos separados.

Pode-se confiar a tarefa de coletar o programa fonte a um programa separado, chamado pré-processador, que por sua vez, pode expandir *macros* em comandos na linguagem fonte.

Processo de compilação



Processo de compilação

Então, o compilador recebe na entrada o programa fone modificado e pode produzir como saída um programa em uma linguagem simbólica, conhecida como *assembly*, considerada mais fácil de ser gerada como saída e mais fácil de depurar.

Essa linguagem simbólica é então processada por um programa chamado montador (*assembler*), que produz o código de máquina realocável como sua saída.

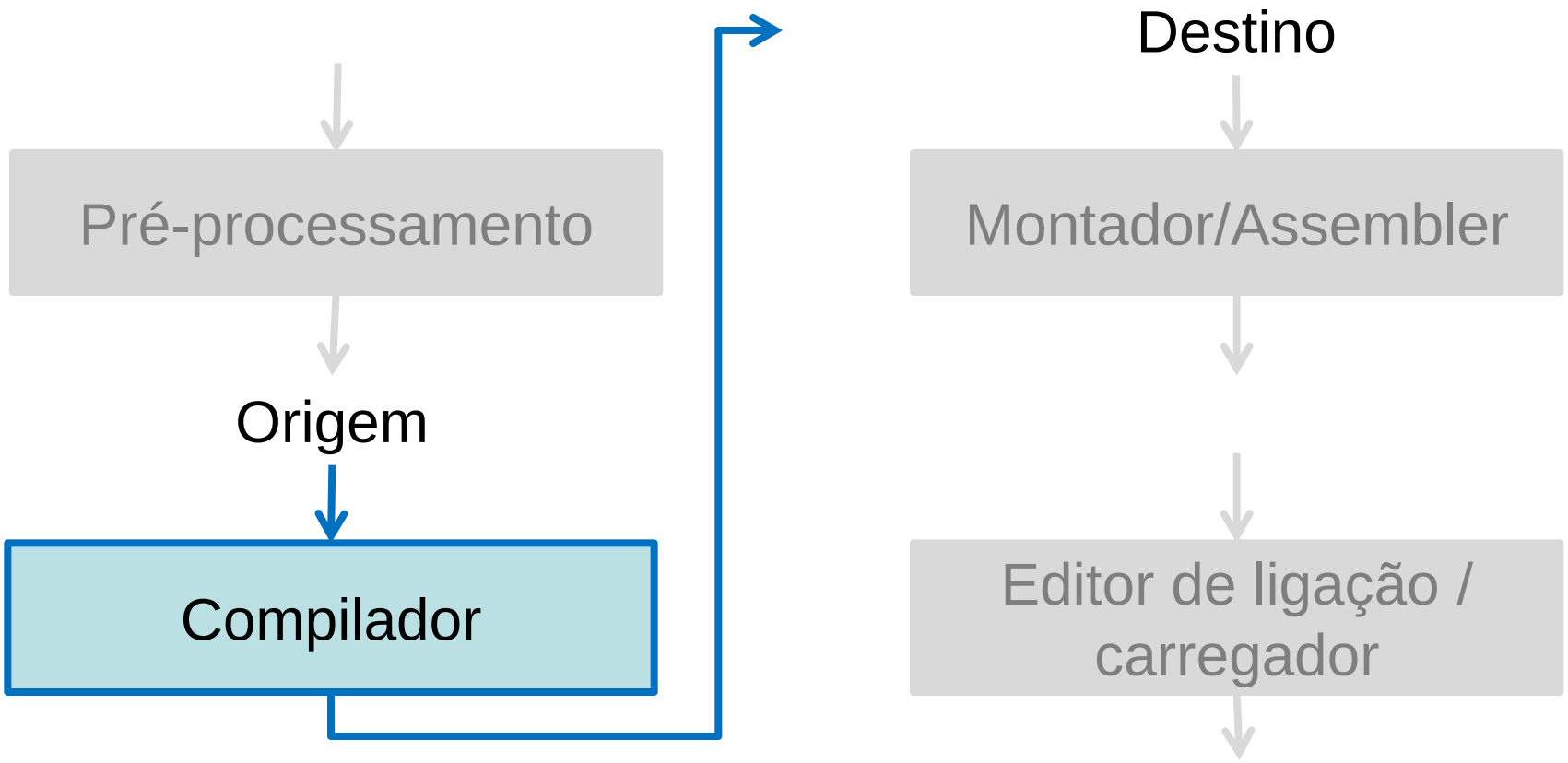
Processo de compilação

Os programas grandes normalmente são compilados em parte, de modo que o código de máquina realocável pode ter de ser ligado a outros arquivo objetos realocáveis e a arquivos de biblioteca para formar o código que realmente é executado na máquina.

Editor de ligação (*linker*) resolve os endereços de máquina externos, onde o código em um arquivo pode referir-se a uma localização em outro arquivo.

Carregador (*loader*) reúne então todos os arquivos objeto executáveis na memória para execução.

Processo de compilação



Compilador - tarefas

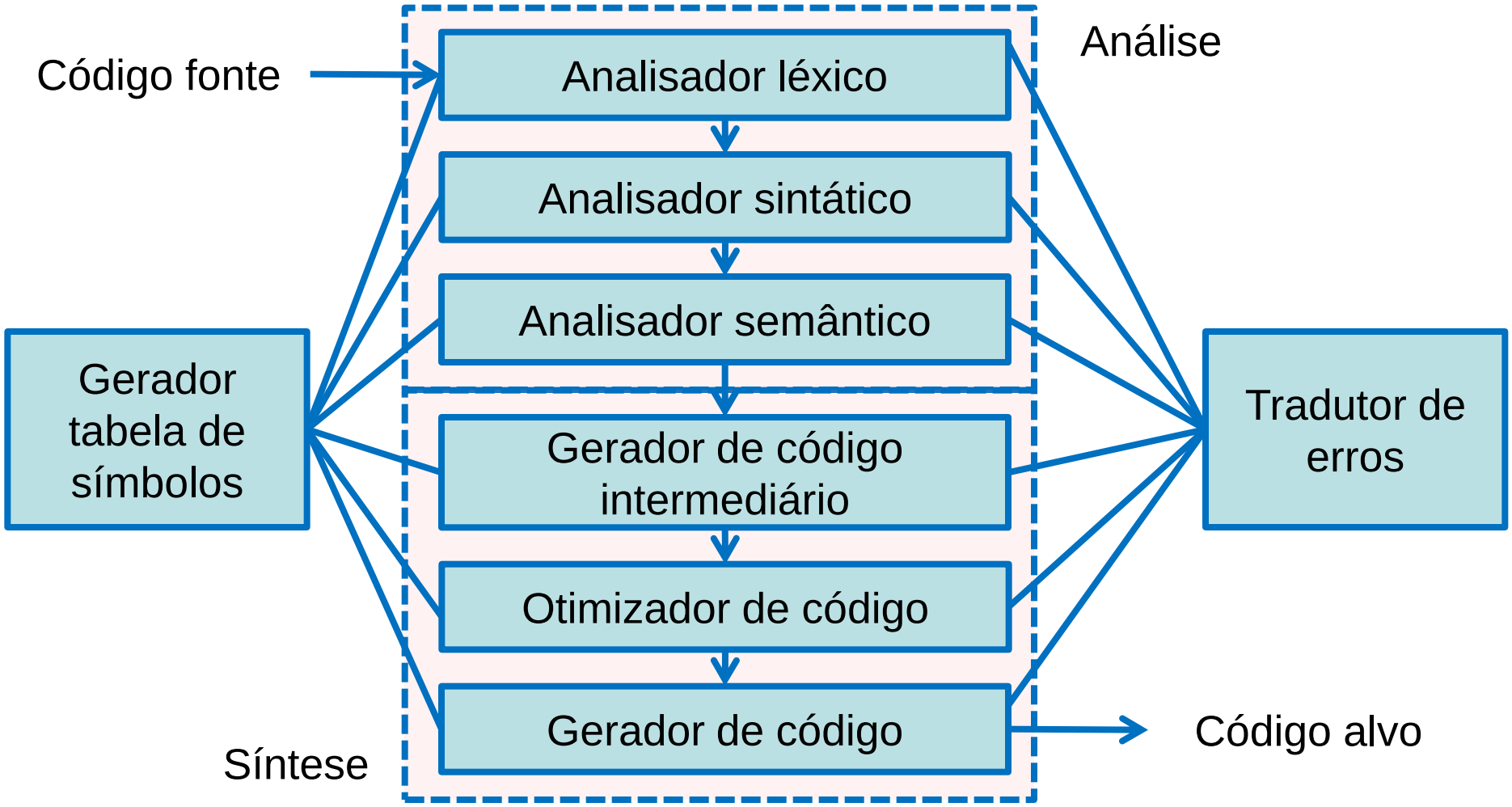
Nesse processo de tradução, há 2 tarefas básicas a serem executadas por um compilador:

Análise (*front-end*), em que o texto de entrada (na ling. fonte) é examinado, verificado e compreendido.

- Análise léxica, sintática e semântica.

Síntese (*back-end*), ou geração de código, em que o texto de saída (na linguagem objeto) é gerado, de forma a corresponder ao texto de entrada.

Compilador - fases



Etapa de análise de um compilador

- Cria representações intermediárias do programa
- Verifica presença de certos tipos de erros

Análise léxica (*scanner ou scanning*): organiza os caracteres e os agrupa em símbolos (*tokens*).

Entrada: Fluxo de caracteres.
Saída: Fluxo de símbolos.
Símbolos: Palavras reservadas, identificadores de variáveis/procedimentos, operadores, pontuação, ...

Análise léxica

Exemplo: montante := saldo + taxa_de_juros * 30;

São identificados os seguintes tokens:

- Identificador montante
- Símbolo de atribuição :=
- Identificador saldo
- Símbolo de adição +
- Identificador taxa_de_juros
- Símbolo de multiplicação *
- Número 30

Análise léxica

Exemplo: montante := saldo + taxa_de_juros * 30;

<identificador, 1>, <:=>,
<identificador, 2>, <+>,
<identificador, 3>, <*>, <número, 30>

Tabela de símbolos

	Nome	Tipo	
1	montante	-	...
2	Saldo	-	
3	Taxa_de_juros	-	
...			

Análise léxica

A tabela de símbolos é atualizada em várias etapas da compilação.

- Estrutura de dados para guardar identificadores e informações sobre eles:
  - Tipo do identificador.
  - Escopo: onde o identificador é válido no programa.
  - se for um procedimento ou função: número e tipo dos argumentos, forma de passagem dos parâmetros e tipo do resultado.

Análise léxica

Eliminação de alguns elementos, como espaços em branco, marcas de formatação e comentários.

O projetista do compilador caracteriza o analisador léxico através de Expressões Regulares (ER), que são usadas no reconhecimento.

Foco: Gramaticas Livres de Contexto (GLC).

A geração do analisador léxico é automática a partir da definição das ERs. Ferramentas FLEX e Lex para a geração automática de *scanners*.

Análise sintática (*Parser*)

Agrupa símbolos em unidades sintáticas.

Exemplo: Os 3 símbolos A+B podem ser agrupados em uma estrutura chamada de expressão.

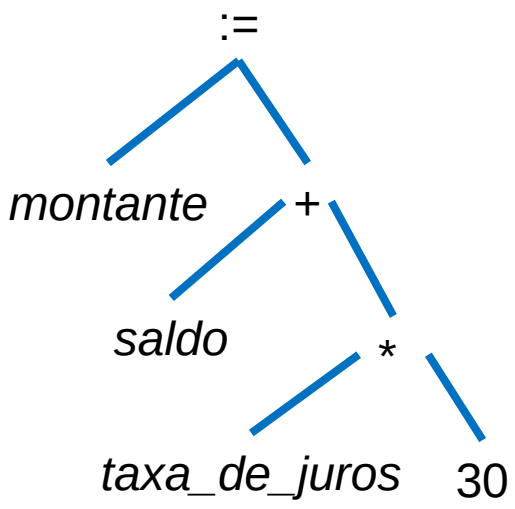
Estas expressões podem ser agrupados para formar comandos ou outras unidades.

A partir dos *tokens*, cria-se uma estrutura em árvore (árvore sintática) que, por sua vez, representa a estrutura gramatical do programa.

Análise sintática (*Parser*)

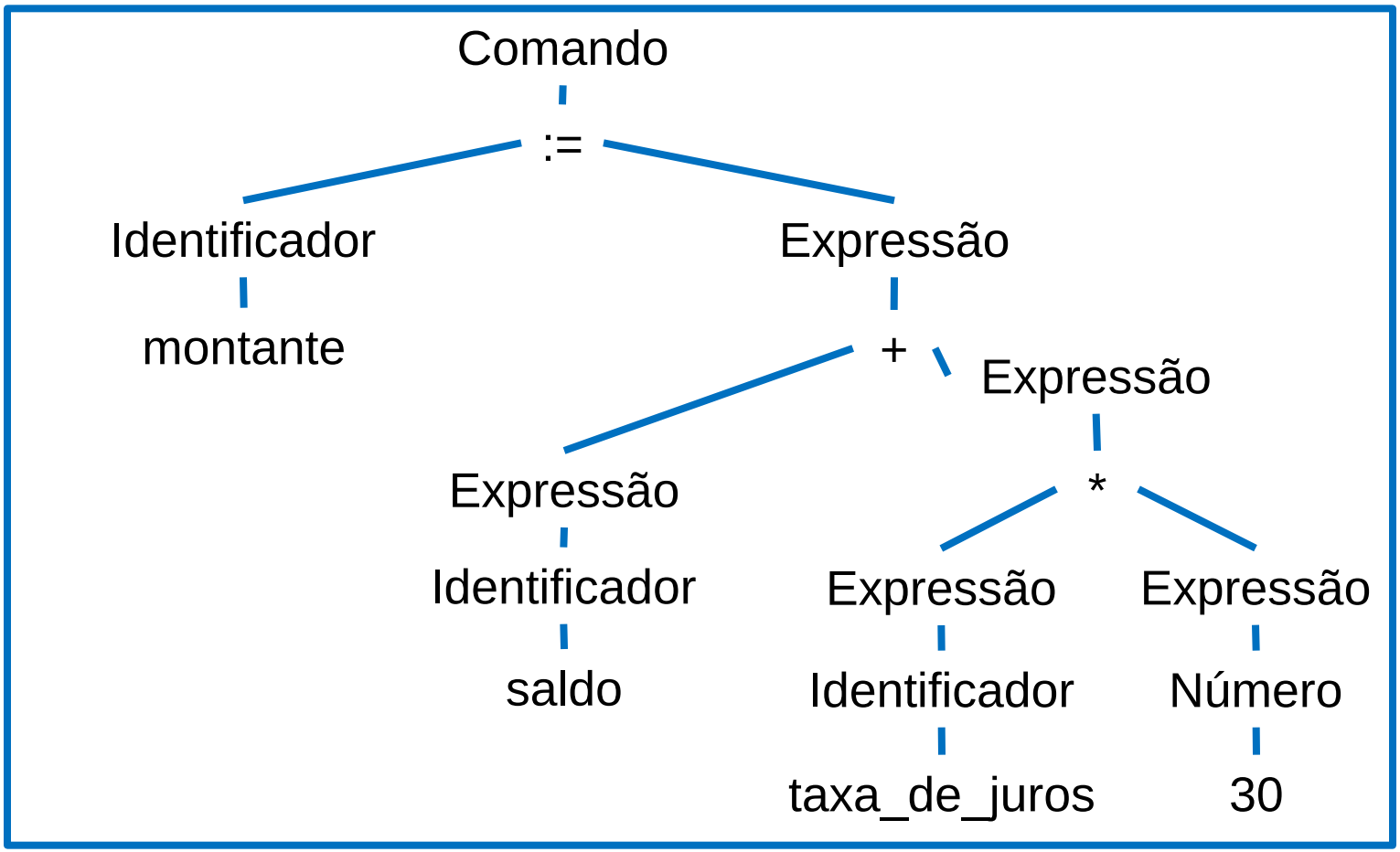
Gramatica Livre de Contexto (GLC) é usada para definir a estrutura do programa reconhecida por um *parser*.

Exemplo: montante := saldo + taxa_de_juros * 30;



Análise sintática (*Parser*)

Exemplo: montante := saldo + taxa_de_juros * 30;



Análise semântica

Procura possíveis erros semânticos e armazena informações contextuais adicionais.

Verifica se as estruturas sintáticas, embora corretas sintaticamente, possui um significado admissível na linguagem.

Exemplo: Não é possível representar em gramática livre de contexto uma regra como “todo identificador deve ser declarado antes de ser usado”.

Análise semântica

Um importante componente é a checagem de tipos.

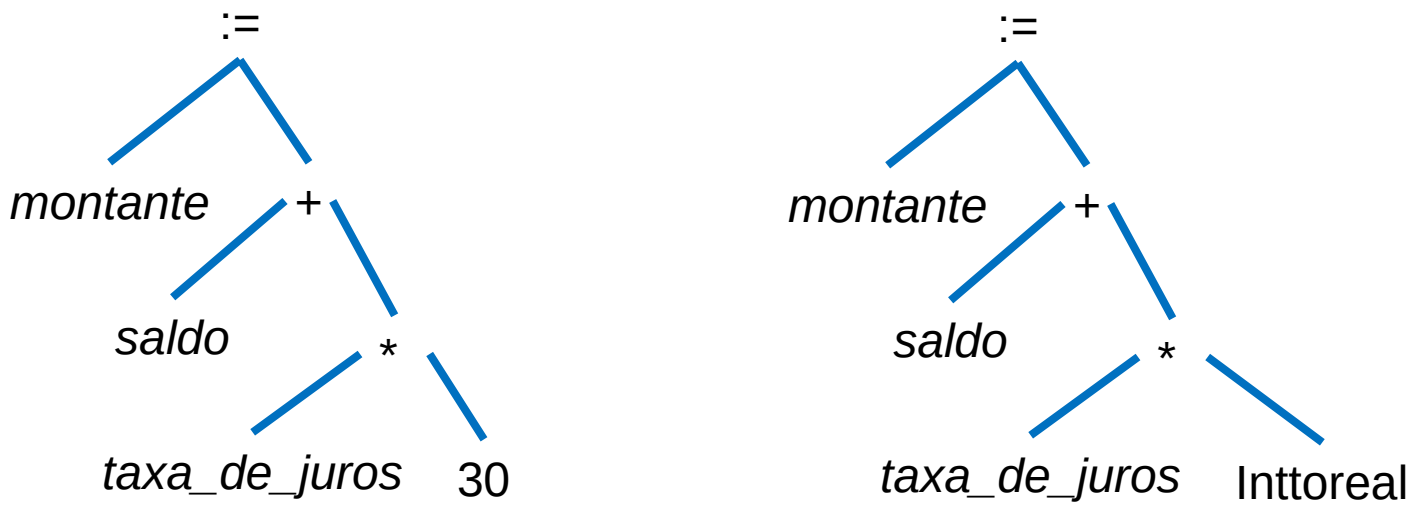
Utiliza informações coletadas anteriormente e as armazena na tabela de símbolos.

Exemplo: x := x + 3.0;

O exemplo está sintaticamente correto, mas pode estar semanticamente errada, dependendo do tipo de x.

Análise semântica

Saída: Árvore de *Parse* anotada.



Conversão de inteiro para real inserida pela análise semântica.

Etapa de síntese de um compilador

- Síntese: constrói o programa destino a partir de representações intermediárias.

- Gerador de código intermediário: Usa estruturas produzidas pelo analisador sintático e verificadas pelo analisador semântico p/ criar uma sequencia de instruções simples.

- Está entre a linguagem de alto nível e a de baixo nível.

Gerador de código intermediário

Uma popular linguagem intermediária é conhecida como código de três endereços.

Considera-se apenas um acumulador.

Exemplo: x := a + b * c;

Traduzir em:

```
t1 := b * c;
t2 := a + t1;
x := t2;
```

Gerador de código intermediário

Toda operação aritmética (binária) gera 3 instruções.
Para t1 := b*c

Carga do primeiro operando no acumulador:

load b

Executa a operação correspondente com o segundo operando, deixando o resultado no acumulador:

mult c

Armazena o resultado em uma nova variável temporária:

store t1

Gerador de código intermediário

Para t2 := a + t1:

load a; add t1; store t2;

Um comando de atribuição gera sempre duas instruções. Para x:= t2

Carrega o valor da expressão no acumulador:

load t2

Armazena o resultado na variável:

store x

Gerador de código intermediário

P/ o comando de atribuição x := a + b * c, é gerado o código intermediário:

1. Load b	{ t1 := b * c }
2. Mult c	
3. Store t1	
4. Load a	{ t2 := a + t1 }
5. Add t1	
6. Store t2	
7. Load t2	
8. Store x	{ x := t2 }

Otimizador de código

Independente de máquina.

Melhora o código intermediário de modo que o programa objeto seja menor (ocupe menos espaço de memória) e/ou mais rápido (tenha tempo de execução menor).

A saída é um novo código intermediário:

load b;
mult c;
add a; *store x;*

Gerador de código

Produz o código objeto final (a partir do código intermediário), ou seja, é última fase da compilação.

Toma decisões com relação à:

- Alocação de espaço p/ os dados do programa;
- Seleção da forma de acessá-los;
- Definição de quais registradores serão usados.

Projetar um gerador código que produza programas objeto eficientes é uma das tarefas mais difíceis no projeto de um compilador.

COMPILADORES

Obrigado!!

Geovane Griesang
geovanegriesang@unisc.br