Aula 1 - Introdução

Daniel A. G. de Oliveira

8 de março de 2025

Compiladores traduzem o código fonte (por isso, algumas vezes é chamado de tradutor) para algum outro código, como um executável; depois o código pode receber os dados de entrada e executá-lo sem a necessidade de compilar novamente. Um interpretador recebe o código fonte e os dados de entrada, e já os executa (sendo necessário compilar o código em cada execução). O principal livro-texto para a disciplina será o chamado livro do dragão, por ter um dragão na capa [1].

1 Breve história dos compiladores

Em 1954 IBM desenvolveu uma máquina chamada 704, primeira máquina de sucesso a ser comercializada. Porém, os clientes perceberam que os custo de software eram maiores que o custo de hardware (que não era baixo). Portanto, muitos começaram a pensar em como escrever código de forma mais eficiente.

Em 1953 John W. Backus desenvolveu Speedcoding para o IBM 701. Speedcoding era um interpretador que tinha o foco em facilidade de uso em troca de recursos do sistema. Assim, era rápido desenvolver código em Speedcoding, porém era 10 a 20x mais lento que código escrito a mão. O interpretador de Speedcoding consumia 300 bytes, o que era 30% da memória do IBM 704. Por isso, não se tornou popular, mas Backus achava que o projeto era interessante.

Backus achou que o problema do Speedcoding era que as fórmulas executadas eram interpretadas. Por isso, ele achou que seria melhor se as fórmulas fossem traduzidas para código de máquina. Com isso, surgiu FORTRAN (FORmulas TRANslated), um projeto que se pensava que iria durar 1 ano, mas levou de 1954 até 1957 (3 anos), o custo do projeto foi de 18 homens-ano [2]. Foi um projeto bem sucedido, em 1958 50% dos programas eram escritos em FORTRAN, melhorando a produtividade.

FORTRAN produziu a um corpo teórico gigante, unindo teoria e prática, tendo um grande impacto em ciência da computação. Compiladores modernos preservam a estrutura (outline) de FORTRAN I.

1.1 Estrutura do FORTRAN I

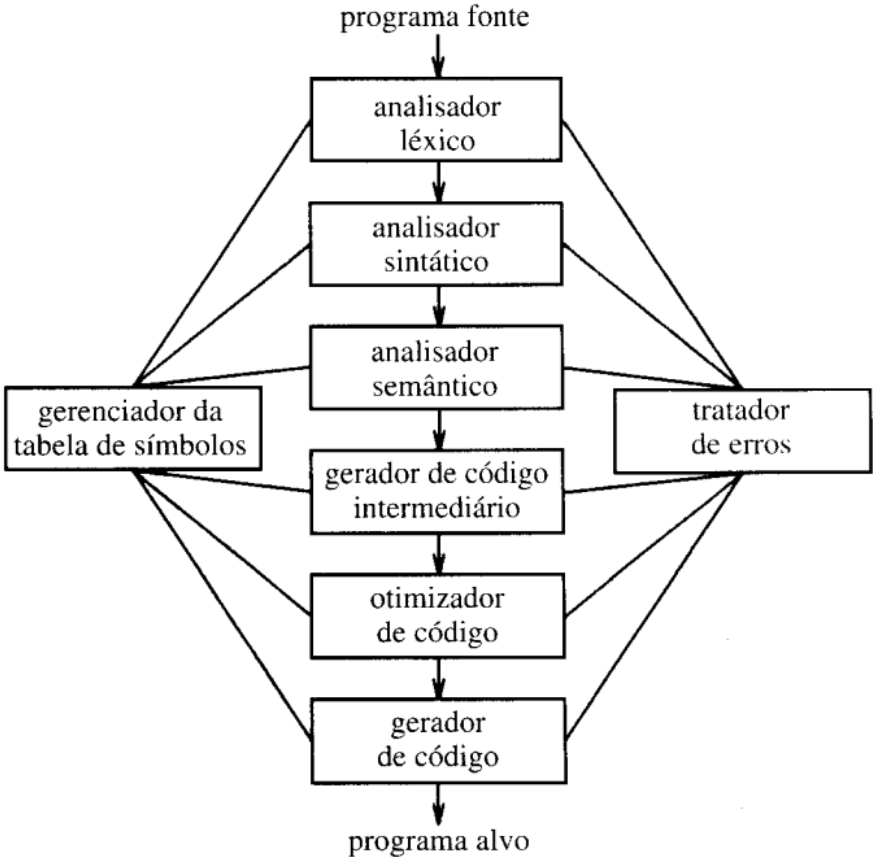
1. Análise léxica
2. Análise sintática
3. Análise Semântica
4. Otimização
5. Geração de código (assembly, bytecode, outra linguagem de alto nível)

2 As fases de um compilador

Tipicamente um compilador opera no que chamamos de *fases*, como na estrutura do FORTRAN I. Cada fase transforma o código fonte de uma representação para outra, embora fases podem ser agrupadas e uma representação explícita entre as fases não precisa ser construída.

A Figura 1 mostra uma decomposição típica das fases de um compilador. Note que duas atividades, muitas vezes chamadas também de fases, permeiam todo o processo. Essas atividades são o gerenciamento da tabela de símbolos e o tratamento de erros.

Figura 1: Fases de um compilador, fonte: [1]



2.1 Gerenciamento da tabela de símbolos

A tabela de símbolos será uma estrutura de dados para armazenar informações sobre os símbolos do código fonte. Esses símbolos, ou identificadores, podem ser, por exemplo, variáveis e nomes de procedimento/funções. A tabela deverá guardar informações como escopo, tipo, tamanho de memória alocada, endereço, parâmetros de entrada entre outros.

2.2 Tratamento de erros

Essa fase, geralmente mais ativa durante a análise sintática e semântica, precisa lidar com os erros encontrados e reportá-los ao usuário. Normalmente é necessário que a compilação continue após a identificação do primeiro erro, detectando assim a maior quantidade de erros possíveis em uma única tentativa de compilação.

2.3 Análise léxica

Primeiro passo, tanto para humanos como máquinas, é o de reconhecer as palavras. O objetivo da análise léxica é dividir o código fonte em "palavras", que são chamadas de *tokens*.

Por exemplo: if x == y then z = 0; else z = 2;

Uma dificuldade, em nível de máquina, é como diferenciar o token de comparação '==' do token de atribuição '='. Algo que só pode ser feito ao olhar caracteres a frente (*look ahead*).

2.4 Análise sintática

Cada token pertence a uma classe diferente, como no português onde cada palavra pode ser um substantivo, adjetivo, artigo e etc. No caso de linguagens de programação, podemos ter as classes de palavra-chave,

variáveis, constantes entre outras.

A análise sintática envolve em entender a estrutura sintática de uma sentença. Por exemplo, em português podemos ter uma sentença válida que consiste em ter um sujeito seguido de um predicado. No caso do if-then-else, a estrutura válida seria algo como:

- palavra-chave: if
- predicado: x == y
- palavra-chave: then
- declaração then: z = 0;
- palavra-chave: else
- declaração else: z = 2;

2.5 Análise semântica

O próximo passo, após identificar a estrutura sintática, seria entender o significado. Algo que é muito difícil para compiladores, o que é feito de forma bem limitada. Até hoje não sabemos como exatamente isso é feito por humanos.

Por exemplo, como identificar o significado exato da frase: João disse que João perdeu a sua bicicleta. Como saber a quem a palavra 'sua' se refere, ao primeiro ou ao segundo João? Ou se cada João se refere a uma pessoa diferente ou a mesma?

Algo próximo em linguagens de programação seria a questão de escopo:

```
{
    Joao = 1;
    {
        Joao = 2;
        cout << Joao;
    }
}
```

Um compilador não consegue lidar, de forma consistente, com ambiguidades. O que compiladores faz é identificar algumas inconsistências, como uma operação que mistura valores em inteiro com variáveis do tipo booleano. Algo como erro de tipo. Algo similar em português seria: João perdeu a bicicleta dela. Sabemos que dela se refere a uma mulher e não pode ser João, portanto a frase está sintaticamente correta mas a semântica está errada.

2.6 Gerador de código intermediário

Alguns compiladores, após as fases de análise, geram uma representação intermediária do código. Essa representação funciona como um código para uma máquina abstrata. Essa representação deve ter duas propriedades importantes: Fácil de produzir, de ser gerada; e fácil de traduzir para o código alvo, o código final. Um exemplo disso é o *código de três endereços*, onde cada instrução possui no máximo três operandos.

```
r1 := id1 * id2
r2 := r1 + id3
id1 := r2
```

2.7 Otimização

Otimização, em português, poderia ser algo como economizar palavras. Por exemplo, a frase: "João disse que o que ele quer é arroz e feijão"poderia ser otimizada para "João quer arroz e feijão".

Otimizações podem ter alvos específicos, como executar em um menor tempo, usar menos memória, economizar consumo energético ou usar menos a rede de dados. Algo que nem sempre é trivial. Por exemplo, o seguinte operação "X = Y * 0"pode ser substituída por "X = 0", algo que é válido para inteiros, mas invalido para ponto flutuante uma vez que NaN (Not a Number) multiplicado por zero ainda é NaN (NaN * 0 = NaN).

2.8 Geração de código

Geralmente a última fase do compilador é produzir código assembly. Mas poderia traduzir o código fonte para qualquer outra linguagem, mesmo que seja uma de alto nível. O equivalente em nível humano, de traduzir de um idioma para outro, como do português para o inglês.

2.9 Exemplo de uma tradução

A imagem a seguir mostra a representação de um enunciado após cada uma das fases de um compilador.

3 Ferramentas para a construção de compiladores

Backus afirma que a vida dele foi motivada pelo princípio da preguiça, ele criou um interpretador de fórmulas e posteriormente o FORTRAN para evitar a escrita de programas para cálculo de trajetórias balísticas em assembly. No mesmo princípio, após a escrita dos primeiros compiladores, surgiram ferramentas para auxiliar nesse processo, ao ponto que um aluno pode escrever um compilador simples em apenas um semestre. Essas ferramentas são frequentemente chamadas de *compiladores de compiladores*, ou *geradores de compiladores*.

Durante a disciplina usaremos duas ferramentas [3], uma para auxiliar na análise léxica (FLEX) e outra para análise gramatical (BISON). Na análise gramatical será incluída tanto a análise sintática como semântica, onde pode-se pensar que essas fases serão unidas.

Referências

[1] A. V. AHO, M. S. LAM, R. SETHI, and J. D. ULLMAN, "Compiladores: princípios, técnicas e ferramentas," *Guanaban Koogan*, vol. 1995, 1986.

[2] J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan, H. Stern, *et al.*, "The fortran automatic coding system," in *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability*, pp. 188–198, 1957.

[3] J. Levine, *Flex & Bison: Text Processing Tools*. "O'Reilly Media, Inc.", 2009.

Figura 2: Tradução de um enunciado, fonte: [1]