Projeto de Compiladores

FIR – Faculdade Integrada do Recife

João Ferreira

12 e 13 de fevereiro de 2007

Questionário

1. Em quais linguagens de programação você já programou?
2. O que você sabe sobre compiladores?
3. Qual a diferença entre compiladores de uma mesma linguagem?
4. O que você espera da disciplina “Projeto de Compiladores”?

Agenda da Aula

- Motivação, Objetivo
- Programa da Disciplina
- Bibliografia
- Linguagem de Programação
- Tradutores
- Compilador
- Modelo Análise-Síntese

Motivação

- Conhecimento das estruturas, técnicas e algoritmos usados na implementação de linguagens: noções sobre uso de memória, eficiência, etc.
- Aplicabilidade freqüente na solução de problemas que exigem alguma forma de tradução entre linguagens ou notações
- Implementação de linguagens para um domínio específico

Motivação

- Possui integração com outras disciplinas:
  - Programação
  - Algoritmos e Estruturas de Dados
  - Infra-estrutura de Software e Hardware
- Conhecer o funcionamento interno de um compilador permite ao programador desenvolver um código melhor
- Conceitos podem ser aplicados em uma grande quantidade de problemas

Bibliografia

- AHO, Alfred V; SETHI, Ravi; ULLMAN, Jeffrey D; Compiladores - princípios, técnicas e ferramentas. Tradução Daniel de Ariosto Pinto. Rio de Janeiro: LTC, 1995.
  - S.I 005.453 A286e
- AHO, Alfred V; SETHI, Ravi; ULLMAN, Jeffrey D; Compilers - principles, techniques and tools. Massachusetts: Addison-Wesley, 1986.
  - S.I 005.453 A286c
- PRICE, Ana Maria de Alencar (.); Implementação de linguagens de programação - compiladores. Porto Alegre: Sagra-Luzzatto, 2001.
  - S.I 005.453 P946i

Linguagens de Programação

- Linguagem de programação
  - Meio de comunicação entre o programador e o computador
  - Notação formal para expressar algoritmos
- Computador -> Linguagem de baixo nível (bits)
- Humanos -> Linguagem de alto nível baseada em instruções (comandos)

Linguagem de Máquina

- Seqüência de instruções primitivas expressas por uma seqüência de bits, que é interpretada para executar uma determinada operação (primitiva): carregar dados, somar registradores, desvios, etc.
- Originalmente se escrevia diretamente em linguagem de máquina:
  0000 0001 0011 0010
- Dificuldade em ler, escrever, editar; controle explícito dos endereços de memória para dados e para o próprio programa

Linguagem de Montagem

- *Assembly language*
  - *Tradução para linguagem de máquina = montagem*
- Uso de um programa montador (*assembler*)
- Instruções ainda muito próximas da linguagem de máquina (relação de 1 para 1)
- Notação simbólica para facilitar a escrita, leitura e edição:

```
LOAD x
ADD R1 R2
JUMPZ h
```

Linguagem de Alto Nível

- Maior nível de abstração

  let s = (a+b+c)/2

- Ao invés de:

```
LOAD R1 a;
ADD R1 b;
ADD R1 c;
DIV R1 #2;
```

Linguagem de Alto Nível

- Precisa permitir:
  - uso de *expressões*, usando notação semelhante à matemática
  - *tipos de dados* primitivos e compostos
  - *estruturas de controle* (if-then-else, while, for, ...)
  - *declarações* de variáveis, tipos, funções, procedimentos etc.
  - *abstração* o que é feito X como é feito
  - *encapsulamento*: classes, pacotes, módulos

Processadores de Linguagens de Programação

- Sistemas que manipulam programas expressos em alguma linguagem de programação: editores, tradutores, compiladores, interpretadores.
- Linguagem de programação x processadores integrados (**I**DEs: ***I**ntegrated **D**evelopment **E**nvironments*)
  - JDK x Eclipse
  - C# x Visual Studio
  - Obj. Pascal x Delphi

Processadores de Linguagens de Programação

- Algumas poucas técnicas de projeto de compiladores podem ser utilizadas na construção de:
  - tradutores
  - formatadores e processadores de texto
  - interpretadores (de programas ou *queries*)
  - máquinas de estado
  - além de compiladores

Especificação de Linguagens de Programação

- Projetista (*designer*): projeta/especifica linguagens de programação
- Implementador: implementa uma linguagem
- Programador: é usuário da linguagem

- Todos devem ter o mesmo entendimento da linguagem: ter como referência a especificação da linguagem

Especificação de Linguagens de Programação

- A sintaxe define a forma do programa: palavras reservadas, organização das frases
- Restrições contextuais (semântica estática) como: regras de escopo e regras de tipo
- Semântica é o significado do programa

Especificação de Linguagens de Programação

- Especificação informal: texto em linguagem natural (inglês ou outra). Riscos: especificação imprecisa, incompleta ou ambígua
- Especificação formal: consistente, completa, não ambígua. Porém mais difícil de escrever e difícil de ser entendida por pessoas que não conhecem a notação utilizada.

Regras de Tipos

- Normalmente, valores são classificados em tipos.
- Cada operação na linguagem tem uma regra de tipos, que define os tipos esperados para os operandos, caso haja, o tipo do resultado.
- Qualquer operação que utilize um valor com tipo errado gera um erro de tipo.
- Regra de tipos para o operador '**>**':
  se os dois operandos são do tipo *int*, então o resultado é do tipo *boolean*;
- Regra de tipos para 'while E do C':
  E deve ser do tipo *boolean*;

Classificação em Relação a Tipos

- Estaticamente tipada
  - Todos os erros de tipos podem ser detectados estaticamente, sem executar o programa
- Dinamicamente tipada
  - Erros de tipos só podem ser detectados durante a execução do programa
  - Uma variável pode assumir diversos valores, de tipos diferentes, durante a execução do programa
- "Não-tipadas"
  - ex: PERL

Como traduzir o código de alto para baixo nível?

Tradutor

- Recebe como entrada um programa em uma linguagem e produz resultado em outra
  - Assemblers (montadores)
    - Mapeamento de linguagem assembly para linguagem de máquina (1 para 1)
  - Macro-assemblers
    - Um comando macro é traduzido para vários comandos simbólicos (1 para N)
  - Compiladores
    - Traduzem código de alto nível para linguagem simbólica ou de máquina

Tradutor

- Pré-compiladores, pré-processadores, filtros
  - Mapeamento de linguagem estendida para linguagem de alto nível
- Interpretadores
  - Em tempo de execução interpreta o código-fonte sem gerar código objeto

Interpretador X Compilador

- Interpretadores não geram código executável
- Compiladores tem performance superior
- Interpretadores são menores
- As técnicas mais modernas permitem que o código fonte seja convertido para uma linguagem intermediária que, em seguida, é interpretada (ex: Java e C# ?)
- Interpretadores facilitam a portabilidade do código

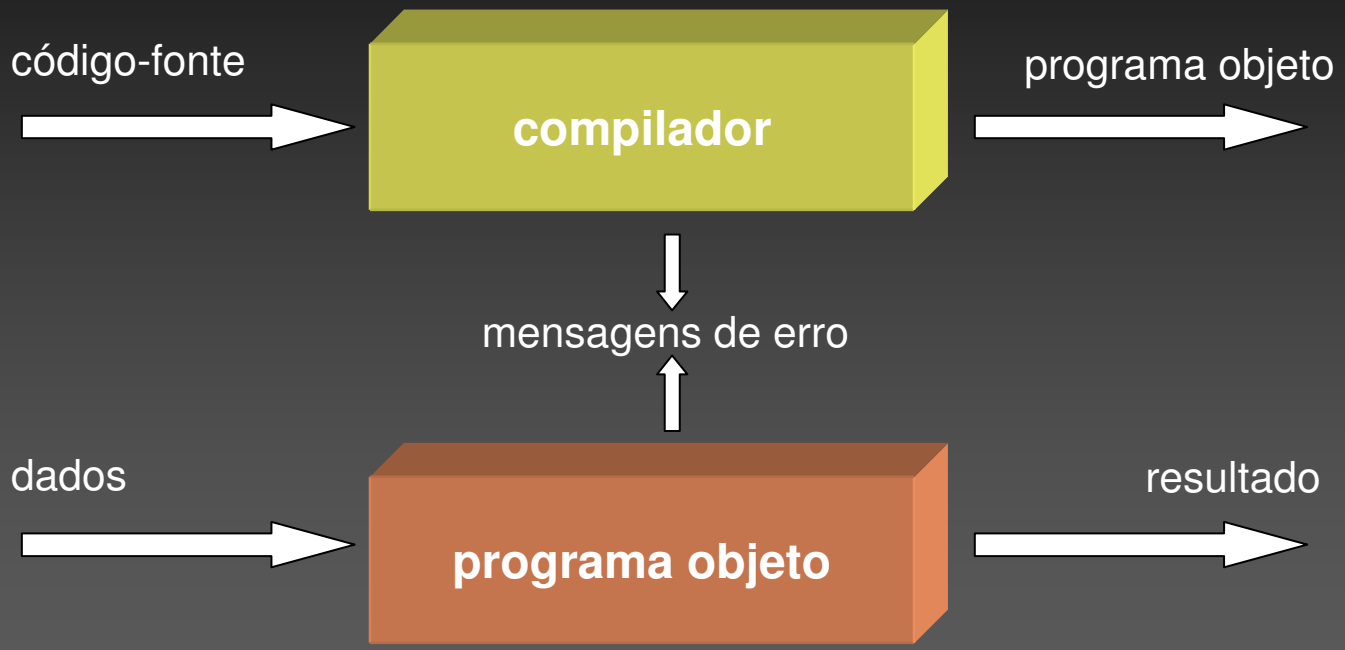
Compilador

- Compilador (sentido geral): aquele que compila, isto é, copia ou transcreve, podendo ou não traduzir aquilo que é transcrito
- Compilador (*software*): programa que lê um texto escrito numa linguagem (fonte) traduzindo-o num texto equivalente escrito em outra linguagem (alvo)
- Nos anos 50, construir um compilador era um processo bastante complicado
- Avanço teórico e de técnicas e ferramentas de implementação tornaram possível implementar compiladores muito mais facilmente

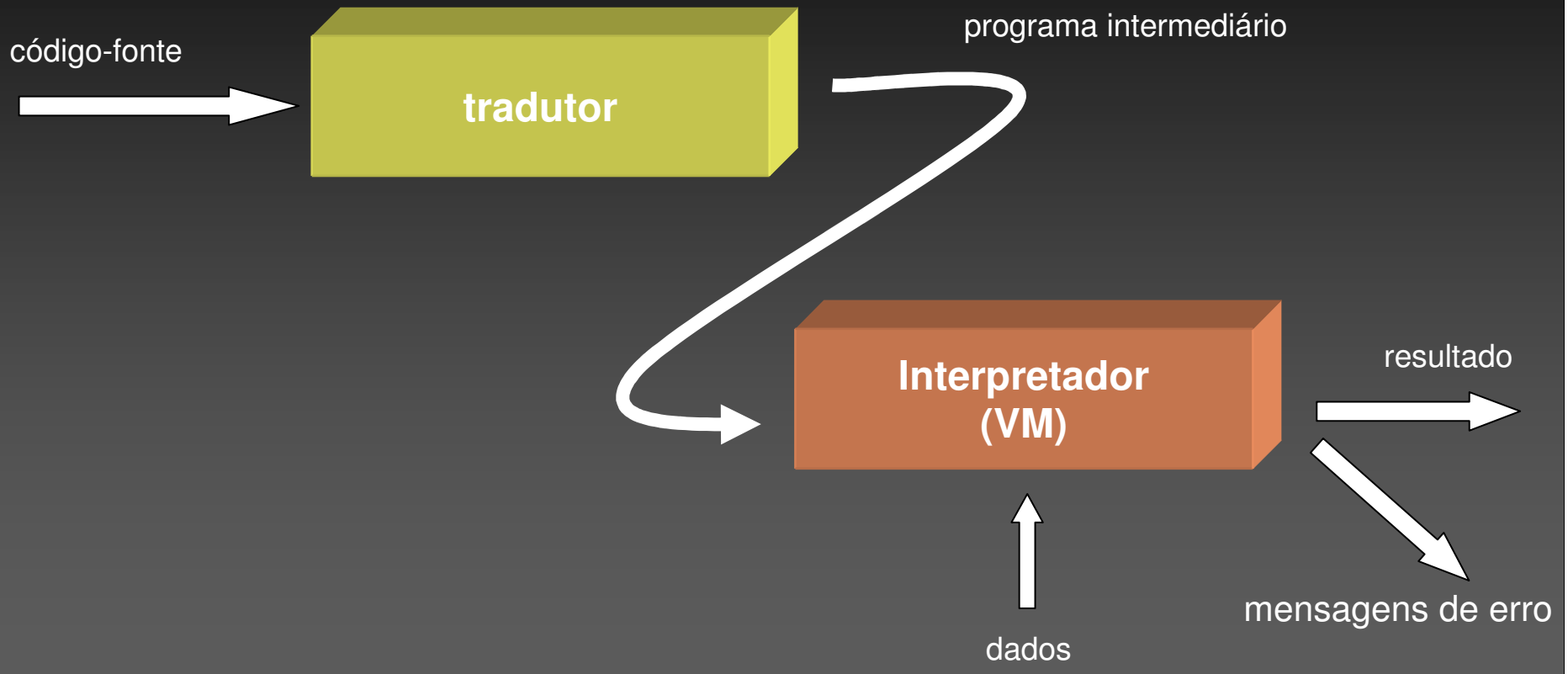
Compilador

- As linguagens fonte (*source languages*) podem tanto ser linguagens de programação (C, Pascal, Java, Fortran etc.) como linguagens especializadas (TEX,SQL, etc.)
- As linguagens alvo (*target languagens*) são também variadas tal como outra linguagem de programação, uma linguagem de máquina (*assembly*) ou uma outra representação

Compilador


código-fonte
compilador
programa objeto
mensagens de erro
dados
programa objeto
resultado

Interpretador

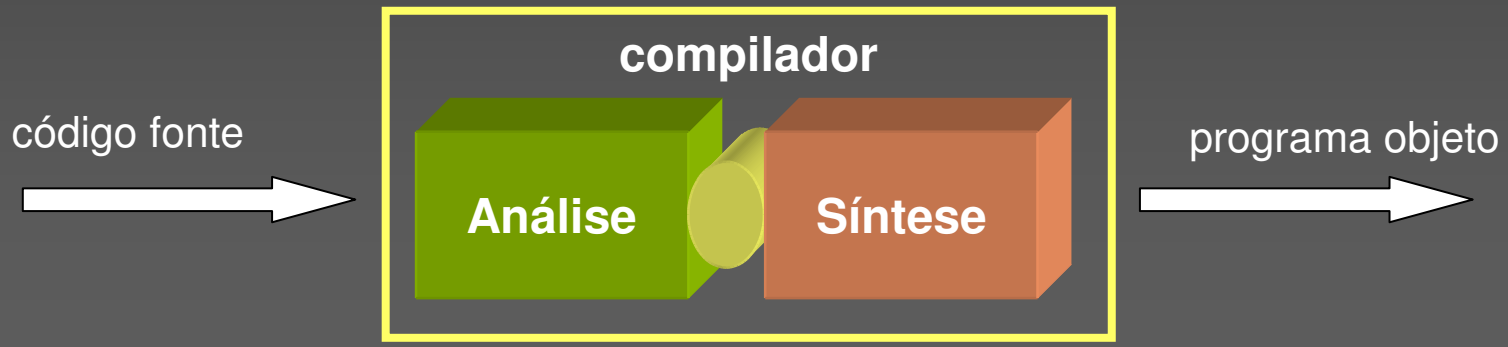


Tipos de Compiladores

- Single-Pass: compilação numa única leitura do programa fonte
- Multi-Pass: compilação através de várias leituras do programa fonte
- Load-And-Go: compilação e a execução do programa fonte
- Debugging: compilação permitindo a depuração do programa fonte
- Optimizing: compilação e a otimização do programa alvo

Modelo Análise-Síntese

- A compilação pode ser dividida em duas partes
  - Análise – divide o código-fonte criando uma representação intermediária
  - Síntese – monta o código desejado a partir da representação intermediária



Análise

- Inclui possíveis mecanismos de pré-processamento
- Divide o programa fonte em suas partes constituintes
- Cria uma representação intermediária do programa fonte
- É uma tarefa relativamente simples

Análise

- Além de compiladores, outras ferramentas utilizam técnicas de análise:
  - Editores de estrutura
  - Verificadores de sintaxe
  - Formatadores para impressão
- Fases
  1. Léxica
  2. Hierárquica
  3. Semântica

Análise Linear (Léxica)

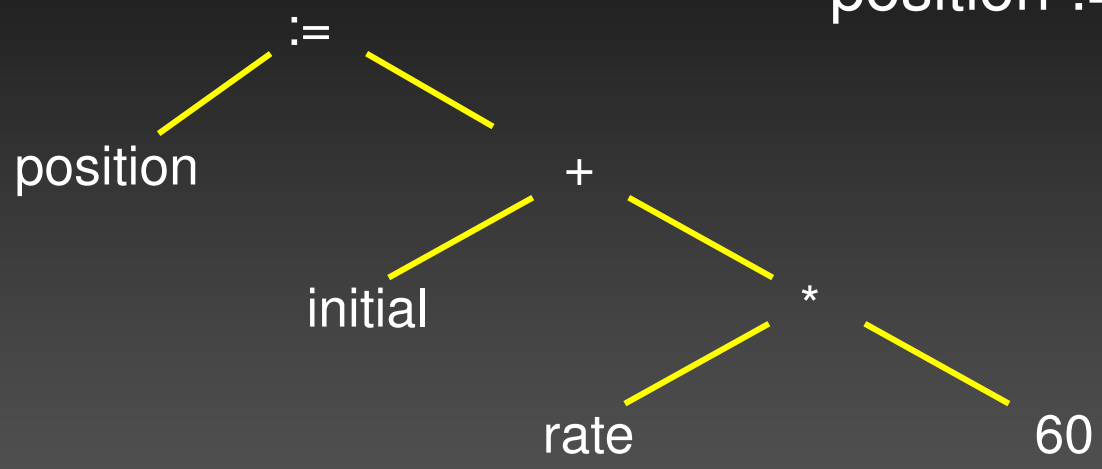
- Caracteres são lidos da esquerda para direita e agrupados em *tokens* (seqüência de caracteres que possui um significado. ex: for, while, if)
- Espaços em branco são eliminados. Exemplo: position := initial + rate * 60
  - Identificadores: position, initial, rate
  - Operador de atribuição: :=
  - Operadores: +, *
  - Número: 60

Análise Hierárquica (Sintática)

- *Tokens* são agrupados hierarquicamente em "frases gramaticais" que serão utilizadas pelo compilador para sintetizar a saída do programa.
- As operações do código-fonte são reconhecidas e armazenadas em uma estrutura hierárquica chamada "árvore sintática"
- Cada nó representa uma operação e cada nó-filho representa os argumentos da operação

Árvore Sintática

position := initial + rate * 60



Análise Semântica

- Realiza verificações a fim de assegurar que os componentes do programa estão agrupados corretamente de acordo com a semântica da linguagem
- Nesta fase é realizada uma verificação de tipo

Síntese

- Constrói o programa alvo a partir da representação intermediária produzida pela análise
- É uma tarefa relativamente complexa
- Utiliza a maior parte das técnicas especializadas

Sintaxe

- É especificada usando gramáticas livres de contexto (BNF – Backus-Naur Form):
  - Conjunto finito de *símbolos terminais*: ‘>=’, ‘while’, ‘;’.
  - Conjunto finito de *símbolos não-terminais*: Programa, Comando, Expressão, Declaração.
  - Um *Símbolo inicial* (um dos não-terminais): Programa
  - Conjunto finito de *regras de produção*.

Sintaxe: Exemplo BNF

<comando> ::= <while> | <atrib> | ...

<while>::= while <expr_bool> do <comando>

<atrib>::= <variavel>:= <expr_arit>

<expr_bool>::= <expr_arit> < <expr_arit>

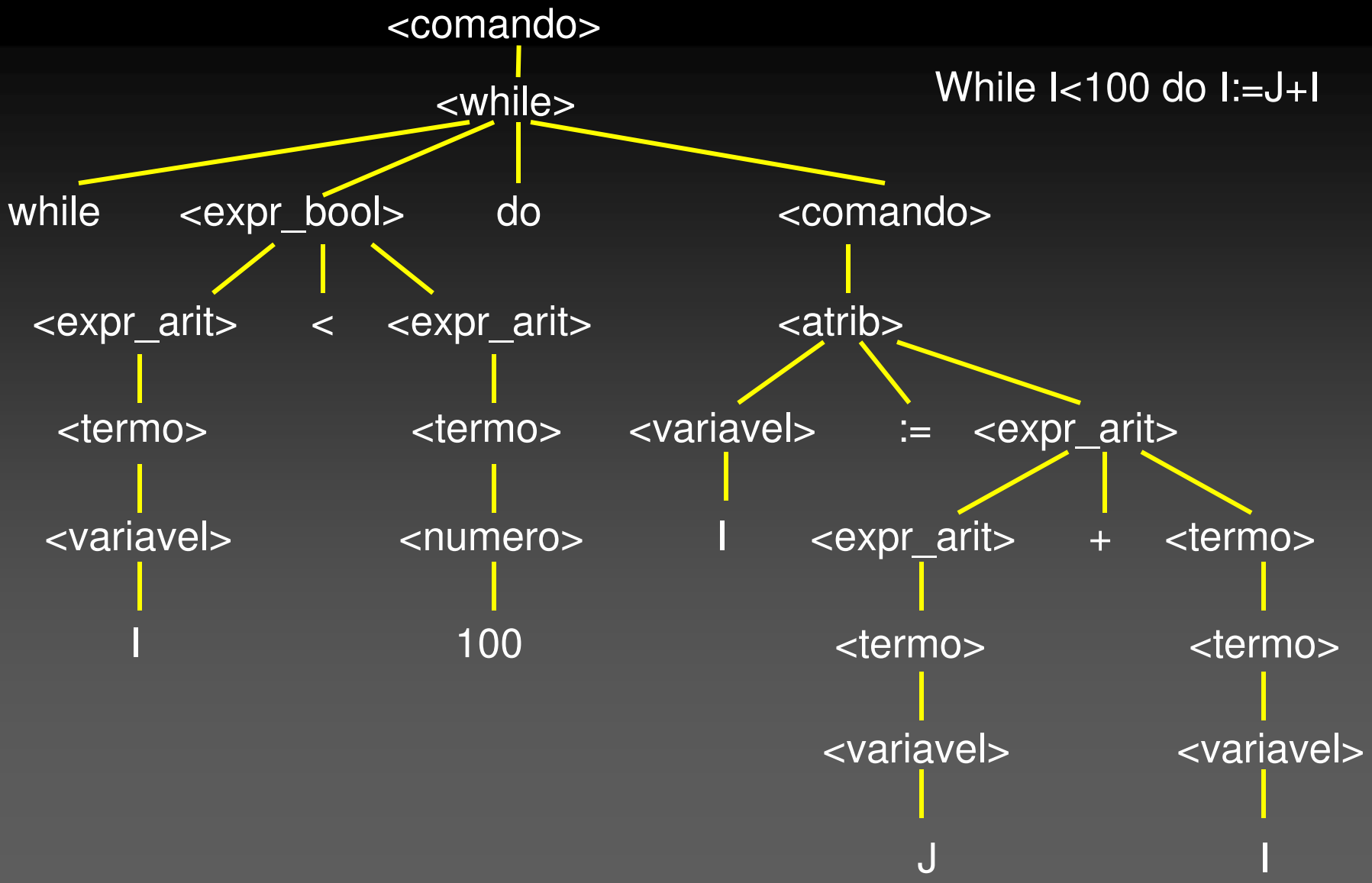
<expr_arit>::=<expr_arit>+<termo>|<termo>

<termo>::= <numero> | <variavel>

<numero>::=100

<variavel>::= I | J

Árvore de Derivação



Até a próxima semana!