Interpretação e Compilação (de Linguagens de Programação)

"If you don't understand interpreters, you can still write programs; you can even be a competent programmer. But you can't be a master."

(Hal Abelson, prefácio de *Essentials of Programming Languages* de Friedman et al.)

Objectivos da Disciplina: Saber

- Quais são as técnicas usados na implementação de LPs ?
- Quais são as peças básicas de construção das LPs ?
- Como se descrevem, analisam e justificam as características das várias linguagens de programação com base nos conceitos básicos apresentados ?
- Como se desenham interpretadores e compiladores para linguagens de programação ?
- Como se consegue prever o comportamento dos programas escritos numa linguagem de programação de forma precisa ?
- Como se expressam propriedades sobre programas como segurança de tipos ?
- Como se descrevem e implementam algoritmos de verificação de linguagens de programação ?
- Como funcionam os ambientes de suporte à execução modernos (Java, .NET) ?

Objectivos da Disciplina: Fazer

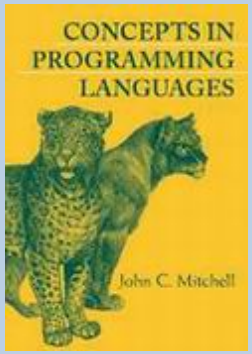
- Como se constroem analisadores sintácticos usando geradores ?
- Como se representam programas como dados para programas ?
- Como se exprime a semântica de uma linguagem ?
- Como se constrói um interpretador para linguagens funcionais ?
- E para uma linguagem orientada por objectos ?
- E se a máquina destino for uma plataforma insdustrial (.NET) ?
- Como se desenham algoritmos simples de análise de programas (sistemas de tipos) ?

"Mapa da Estrada"

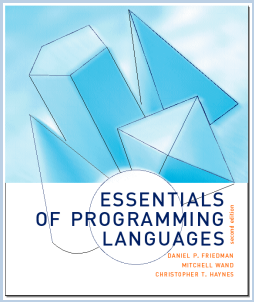
Em ICLP estudamos os conceitos fundamentais das linguagens de programação, abordando os seus aspectos sintácticos e semânticos tanto do ponto de vista dos fundamentos teóricos, como do ponto de vista da sua implementação:

- Expressões e valores
- Ligação e mecanismos de nomeação
- Estado (memória)
- Abstracção funcional e procedimental
- Tipos e sistemas de tipos
- Abstracção de dados
- Objectos, Classes e Módulos
- Técnicas de Compilação

Bibliografia

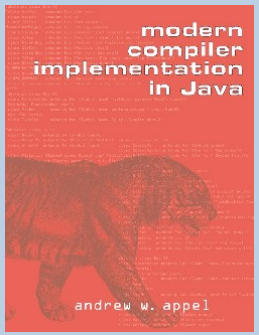


"*Concepts in Programming Languages*",
John C. Mitchell,
Cambridge University Press.
ISBN 0 521 78098 5



"Essentials of Programming Languages",
Daniel Friedman, Mitchell Wand, Christopher Haynes,
MIT Press.

"The Study of Programming Languages",
Ryan Stansifer,
Prentice Hall International Edition.



"Modern Compiler Implementation in Java"
Andrew W. Appel
Cambridge University Press

Unidade 1: Introdução

Esta unidade fala sobre conceitos gerais associados ao desenho de linguagens de programação e ambientes de programação.

- Expressividade das linguagens de programação
- Computabilidade e completude de Turing
- Sintaxe e semântica das linguagens de programação
- Sintaxe abstracta e sintaxe concreta
- Interpretação e compilação de programas
- Ambientes de execução - Máquinas Virtuais, Linguagens intermédias, portabilidade

Linguagens de Programação

- As linguagens de programação descrevem/especificam processos computacionais (computações).
- As linguagens são compostas por duas partes que devem ser descritas de uma forma precisa e não ambígua:
  - Sintaxe

> syntax |ˈsinˌtaks|
>
> noun
> the arrangement of words and phrases to create well-formed sentences in a language *: the syntax of English.*
> • a set of rules for or an analysis of this *: generative syntax.*
> • the branch of linguistics that deals with this.
> ORIGIN late 16th cent.: from French ***syntaxe***, or via late Latin from Greek ***suntaxis***, from ***sun- 'together'*** + ***tassein 'arrange.'***

  - Semântica

> semantics |səˈmantiks|
>
> plural noun [usu. treated as sing.]
> the branch of linguistics and logic concerned with meaning. There are a number of branches and subbranches of semantics, including **formal semantics**, which studies the logical aspects of meaning, such as sense, reference, implication, and logical form, **lexical semantics**, which studies word meanings and word relations, and **conceptual semantics**, which studies the cognitive structure of meaning.
> • the meaning of a word, phrase, sentence, or text *: such quibbling over semantics may seem petty stuff.*
> DERIVATIVES
> **semantician** |ˌsēmanˈti sh ən| |səˈmənˈtɪʃən| |-ˈtɪʃ(ə)n| noun
>
> **semanticist** |səˈmən(t)əsəst| |-tɪsɪst| noun

Linguagens de Programação

Exemplo de ambiguidade na sintaxe:
Qual o valor da expressão `f(10)` ?

```
int f(int x) {
  if ( x > 0 )
    if ( x < 10 ) return x;
  else return 10;
  return 0;
}
```

```
f(10) = ??
```

Linguagens de Programação

Exemplo de ambiguidade na semântica:
Qual o valor da expressão `f(2)+g(3)` ?

```
#include "stdio.h"

int a = 0;

int f(int x) { a = a + 1; return x; }

int g(int y) { return y+a; }

int sum(int x, int y) { return x+y; }

int main() { printf("%d\n", sum(f(2),g(3))); }
```

```
f(2) + g(3) = ??
```

Linguagens de Programação

Exemplo de ambiguidade na semântica:
Qual o valor da expressão `f(2)+g(3)` ?

```
public class    A {
   static int a = 0;

   static int f(int x) {
          a = a + 1;
          return x;
   }
   static int g(int y) { return y + a; }

   static int sum(int x, int y) { return x + y; }

   public static void main(String[] args) {
          System.out.println(sum(f(2), g(3)));

   }
}
```

`f(2) + g(3) = ??`

Linguagens de Programação (Sintaxe)

A sintaxe caracteriza a forma como se escrevem os programas da linguagem, sem atender ao seu significado. A sintaxe é normalmente descrita por um conjunto de palavras ou "tokens", formando um léxico, e a estrutura em que se pode organizar os tokens para formar frases bem formadas.

```
Inteiro:  ("0" | ["1"-"9"]["0"-"9"]*)
Real: (["0"-"9"]) "." (["0"-"9"])* ("E" ...)?
Identificador: [a-z,A-Z,_][a-z,A-Z,_,0-9]*
palavras reservadas: int, float, void, while, class, ...
```

The `if` statement is in the form:

```
if (<expression>)
    <statement1>
else
    <statement2>
```

```
switch (<expression>)
{
    case <label1> :
        <statements 1>
    case <label2> :
        <statements 2>
        break;
    default :
        <statements 3>
}
```

Linguagens de Programação (Sintaxe)

A sintaxe caracteriza a forma como se escrevem os programas da linguagem, sem atender ao seu significado. A sintaxe é normalmente descrita por um conjunto de palavras ou "tokens", formando um léxico, e a estrutura em que se pode organizar os tokens para formar frases bem formadas.

Exemplo de observação sobre "sintaxe":

"Enquanto na linguagem C os blocos são delimitados por chavetas { e }, na linguagem Pascal usam-se os delimitadores `begin end`"

A sintaxe concreta de uma linguagem de programação pode ser descrita precisa e formalmente usando expressões regulares para os tokens e gramáticas para as frases (Teoria da Computação).

A partir gramática da linguagem constroem-se programas que reconhecem e processam os programas bem formados, os Parsers.

Linguagens de Programação (Sintaxe)

Um exemplo para começar

Derivação esquerda numa gramática LL(1)

Uma gramática não-LL(1)

$$S \rightarrow E;$$
$$E \rightarrow T \mid E + T$$
$$T \rightarrow a \mid (E)$$

Eliminando a recursividade à esquerda

$$S \rightarrow E;$$
$$E \rightarrow TX$$
$$X \rightarrow \lambda \mid +TX$$
$$T \rightarrow a \mid (E)$$

É uma gramática LL(1)!

Derivação esquerda de $a + a;$

Derivação	Palavra
S	$a + a;$
$\Rightarrow E;$	$a + a;$
$\Rightarrow TX;$	$a + a;$
$\Rightarrow aX;$	$a + a;$
$\Rightarrow a + TX;$	$a + a;$
$\Rightarrow a + aX;$	$a + a;$
$\Rightarrow a + a;$	$a + a;$

As produções são determinadas pelo próximo símbolo da palavra ainda não produzido: símbolo director da produção.

Linguagens de Programação (Sintaxe)

```
%token  NAME
%token  NUMBER
%token  EQ
%token PLUS MINUS TIMES DIV
%left MINUS PLUS
%left TIMES DIV
%nonassoc UMINUS

%%

statement_list
        : statement
        | statement statement_list

statement
        : NAME EQ expression ';' {vbltable[$1] = $3; }

expression
        : expression PLUS expression {$$ = $1 + $3;}
        | expression MINUS expression {$$ = $1 - $3;}
        | expression TIMES expression {$$ = $1 * $3;}
        | expression DIV   expression {$$ = $1 / $3;}
        | MINUS expression %prec UMINUS {$$ = - $2;}
        | '(' expression ')' { $$ = $2; }
        | NUMBER
        | NAME   { $$ = vbltable[$1]; }
```

Linguagens de Programação (Sintaxe)

```
void Start() :
{ }
{
    exp() <EOL>
}

void exp() :
{ }
{
    term() [ <PLUS> exp() ]
}

void term() :
{ }
{
    factor() [ <MULTIPLY> term() ]
}

void factor() :
{  }
{
    <CONSTANT>
|   <LPAR> exp() <RPAR>
}
```

Linguagens de Programação (Semântica)

A semântica descreve o significado das frases sintacticamente válidas de uma linguagem. A semântica das linguagens deve ser definida de maneira precisa.

10.4 Array Access

A component of an array is accessed by an array access expression (§15.13) that consists of an expression whose value is an array reference followed by an indexing expression enclosed by `[` and `]`, as in `A[i]`. All arrays are `0`-origin. An array with length *n* can be indexed by the integers `0` to *n*-1.

Arrays must be indexed by `int` values; `short`, `byte`, or `char` values may also be used as index values because they are subjected to unary numeric promotion (§) and become `int` values. An attempt to access an array component with a `long` index value results in a compile-time error.

All array accesses are checked at run time; an attempt to use an index that is less than zero or greater than or equal to the length of the array causes an `ArrayIndexOutOfBoundsException` to be thrown.

in *The Java Language Specification*

Linguagens de Programação (Semântica)

A semântica descreve o significado das frases sintacticamente válidas de uma linguagem. A semântica das linguagens deve ser definida de maneira precisa.

Exemplos de observações sobre "semântica":

"Em C, um vector é modelado por um apontador, mas em Pascal um vector é um valor primitivo"

"A linguagem ML (OCaml) é uma linguagem imperativa, mas centrada no uso de funções"

"Na linguagem Java os argumentos são sempre passados por valor, mas em Pascal também podem ser passados por referência"

Linguagens de Programação (Semântica)

A semântica descreve o significado das frases sintacticamente válidas de uma linguagem. A semântica das linguagens deve ser definida de maneira precisa.

A semântica de uma linguagem pode ser definida precisamente por uma função computável I que atribui um significado a cada programa (ou fragmento de programa)

$$I : PROG \rightarrow DENOT$$

PROG = conjunto dos programas

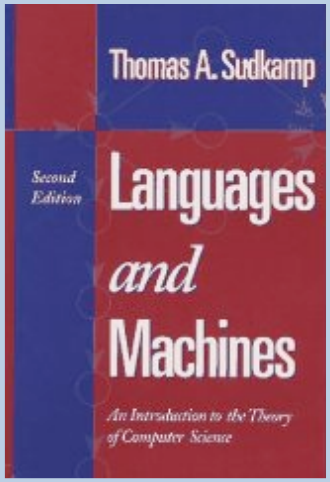
DENOT = conjunto dos significados possíveis (denotações)

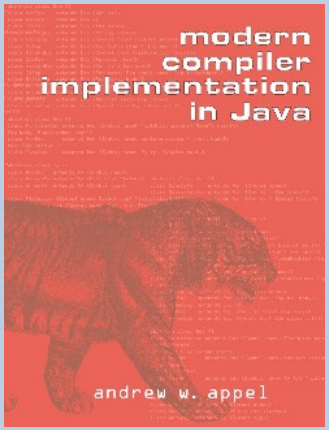
A função semântica I pode ser vista como um **algoritmo** que "sabe como interpretar" todos os programas (sintacticamente correctos) de uma linguagem, determinando o seu **valor** ou **efeito**

Resumo: Ideias a reter!

- Uma linguagem de programação é definida por sintaxe e semântica
- A sintaxe de uma linguagem é definida por uma gramática e é reconhecida e manipulada por uma ferramenta, um analisador sintáctico (*Parser*).
- A semântica de uma linguagem é definida por um algoritmo interpretador que dado um programa dá uma denotação (valor, efeito, programa noutra linguagem, etc)

Leituras: Ideias a procurar!!





Aulas de Teoria da Computação sobre gramáticas - LL(1)

Capítulos 1, 2 e 3: "Modern Compiler Implementation in Java"

"Languages and Machines", Thomas A. Sudkamp, Part II.

http://en.wikipedia.org/wiki/Syntax_of_programming_languages

http://en.wikipedia.org/wiki/Backus–Naur_Form

http://en.wikipedia.org/wiki/Semantics#Computer_science

Interpretação e compilação

Qual a diferença entre um compilador e um interpretador?

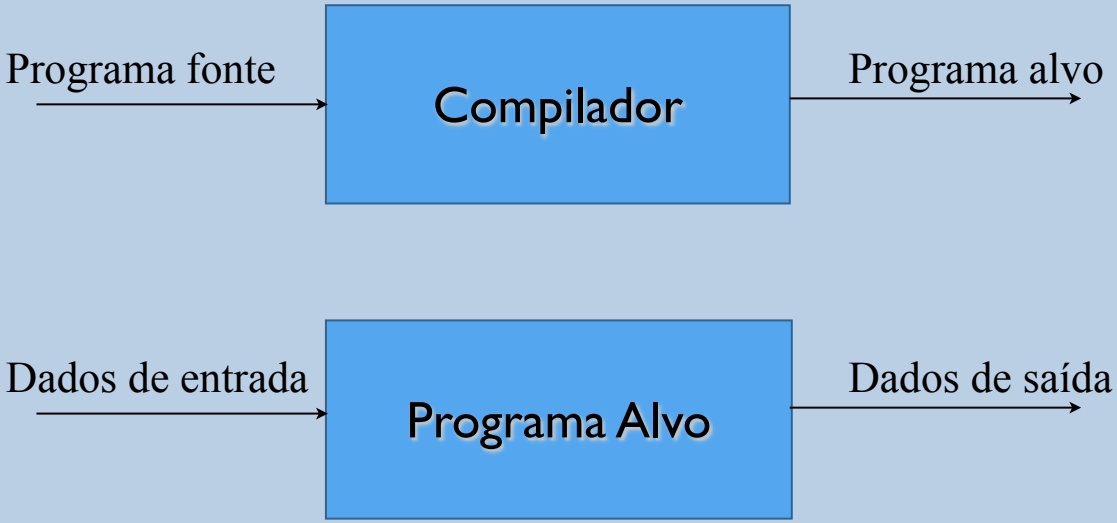
O que é uma linguagem intermédia?

O que é mais eficiente? O que é um JIT?

Qual é a arquitectura de um compilador?

Compiladores

"Um compilador é um programa que lê um programa numa linguagem (fonte) e o traduz para um programa equivalente noutra linguagem (alvo). Um papel importante do compilador é detectar erros no programa fonte. Se a linguagem alvo for uma linguagem máquina (executável) então o programa pode ser chamado para processar dados de entrada e produzir dados de saída." (in Aho et al)



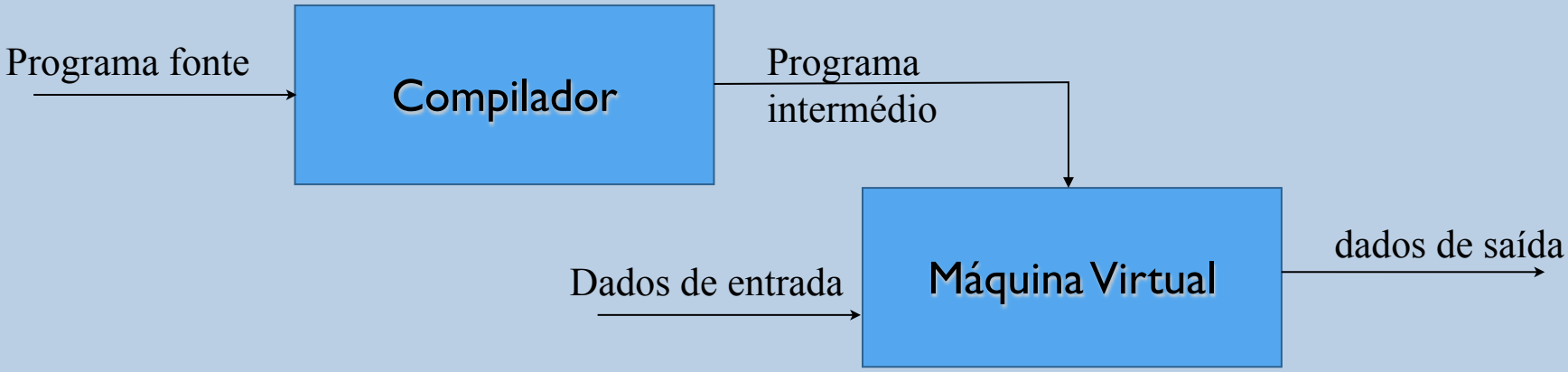
Interpretadores

Um interpretador é um programa que lê um programa numa linguagem (fonte) e produz um valor ou um efeito no seu próprio estado. Um interpretador é normalmente mais lento na produção dos dados de saída.

Interpretadores

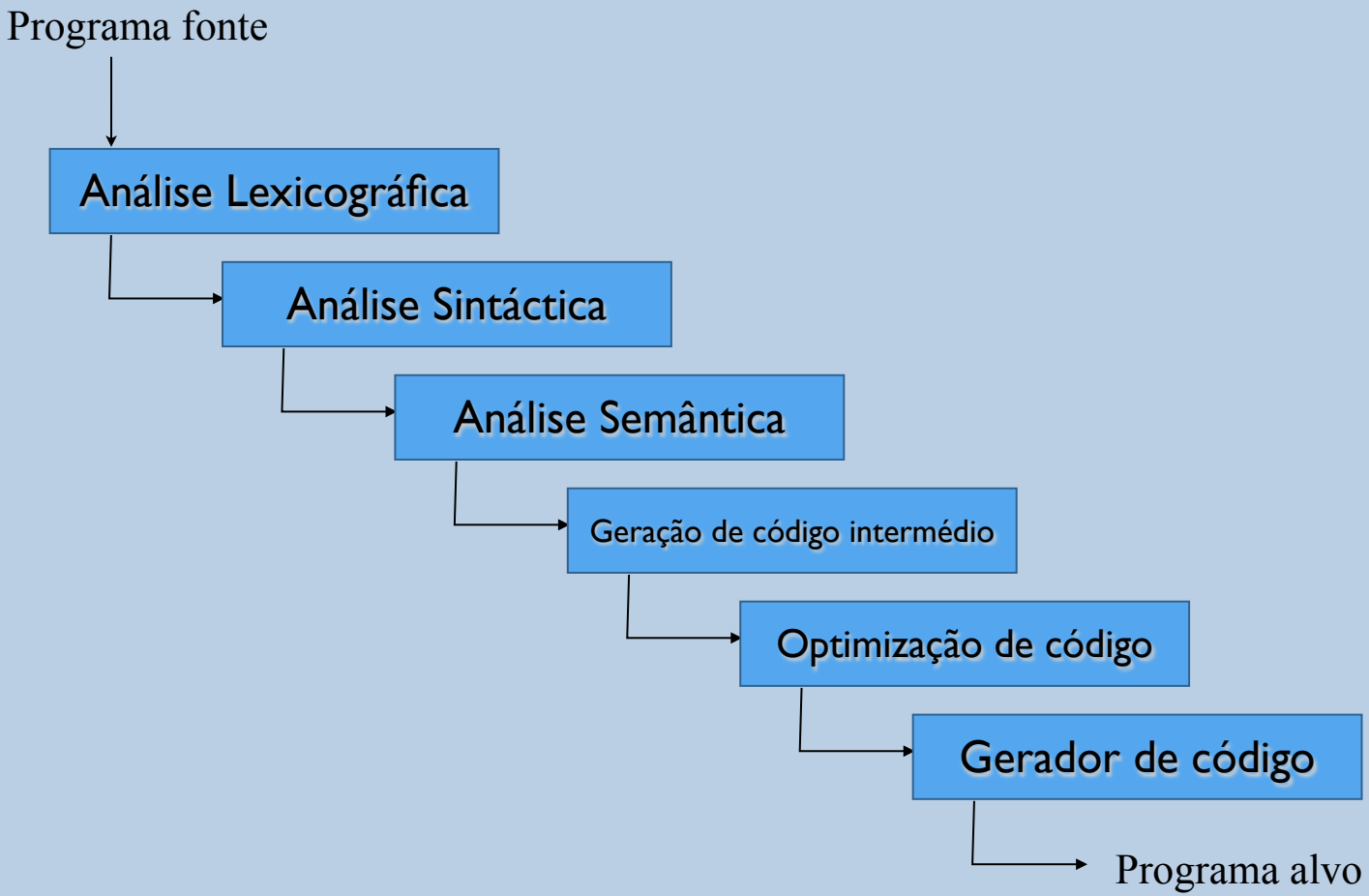
Um interpretador é um programa que lê um programa numa linguagem (fonte) e produz um valor ou um efeito no seu próprio estado. Um interpretador é normalmente mais lento na produção dos dados de saída.

No caso da linguagem Java é utilizada uma linguagem intermédia que é interpretada por uma máquina virtual (a JVM). Para além disso, o código intermédio é compilado em tempo de execução (JIT) para código máquina e corrido directamente no processador real.



Arquitectura de um compilador

- **Um compilador divide-se tradicionalmente nas seguintes fases:**

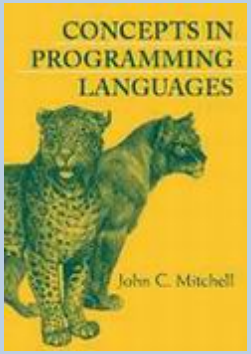


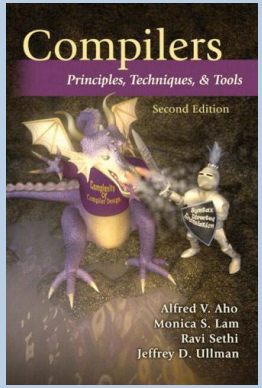
Leituras: Ideias a procurar!!

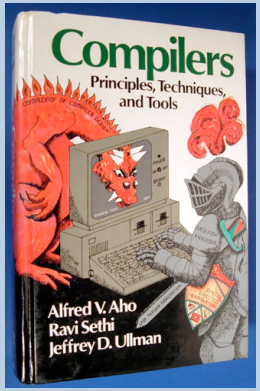
Secção 4.1:"Concepts in Programming Languages"

Capítulo 1:"Compilers: Principles, Techniques, and Tools"

http://en.wikipedia.org/wiki/Compiler



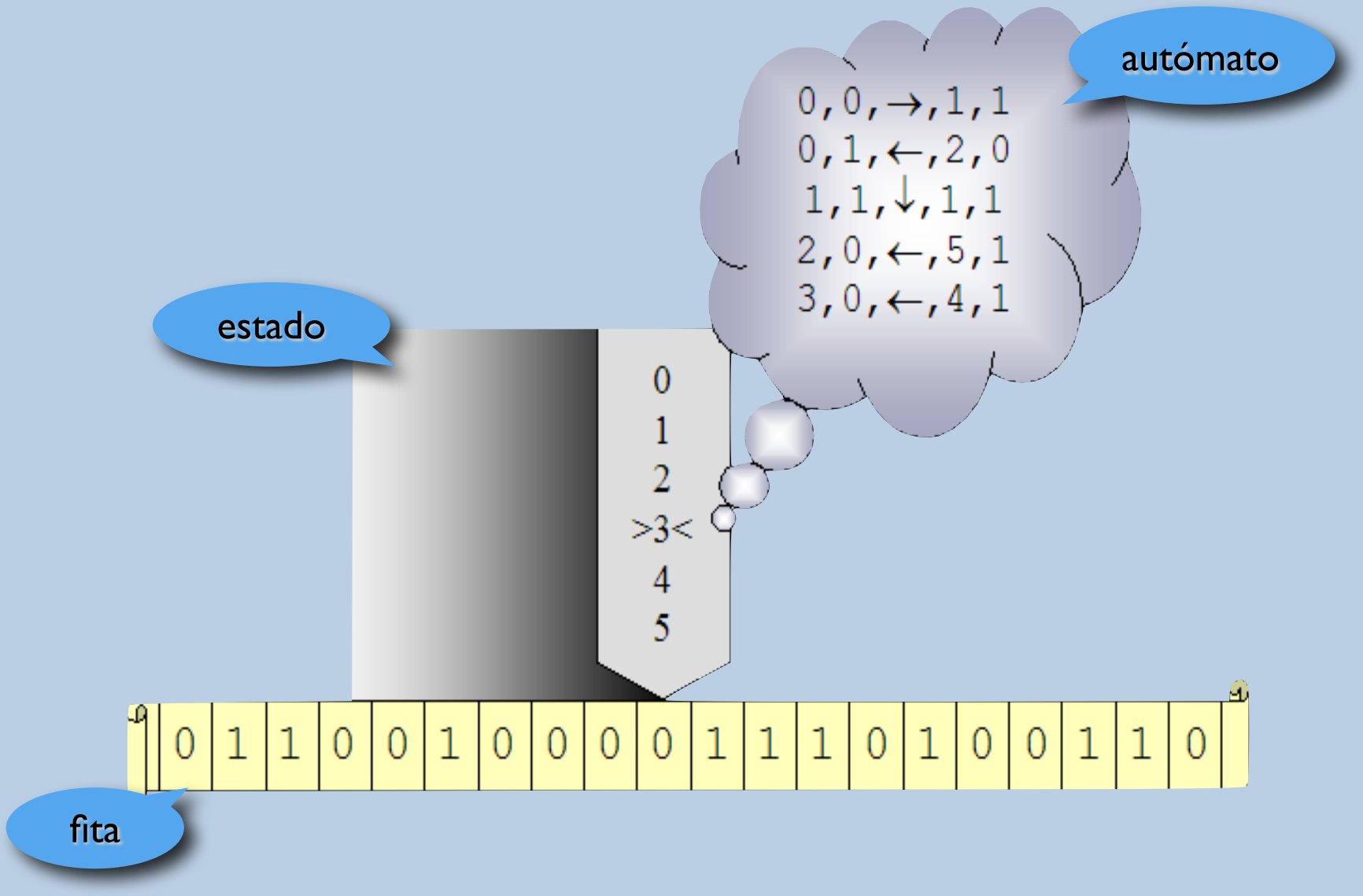




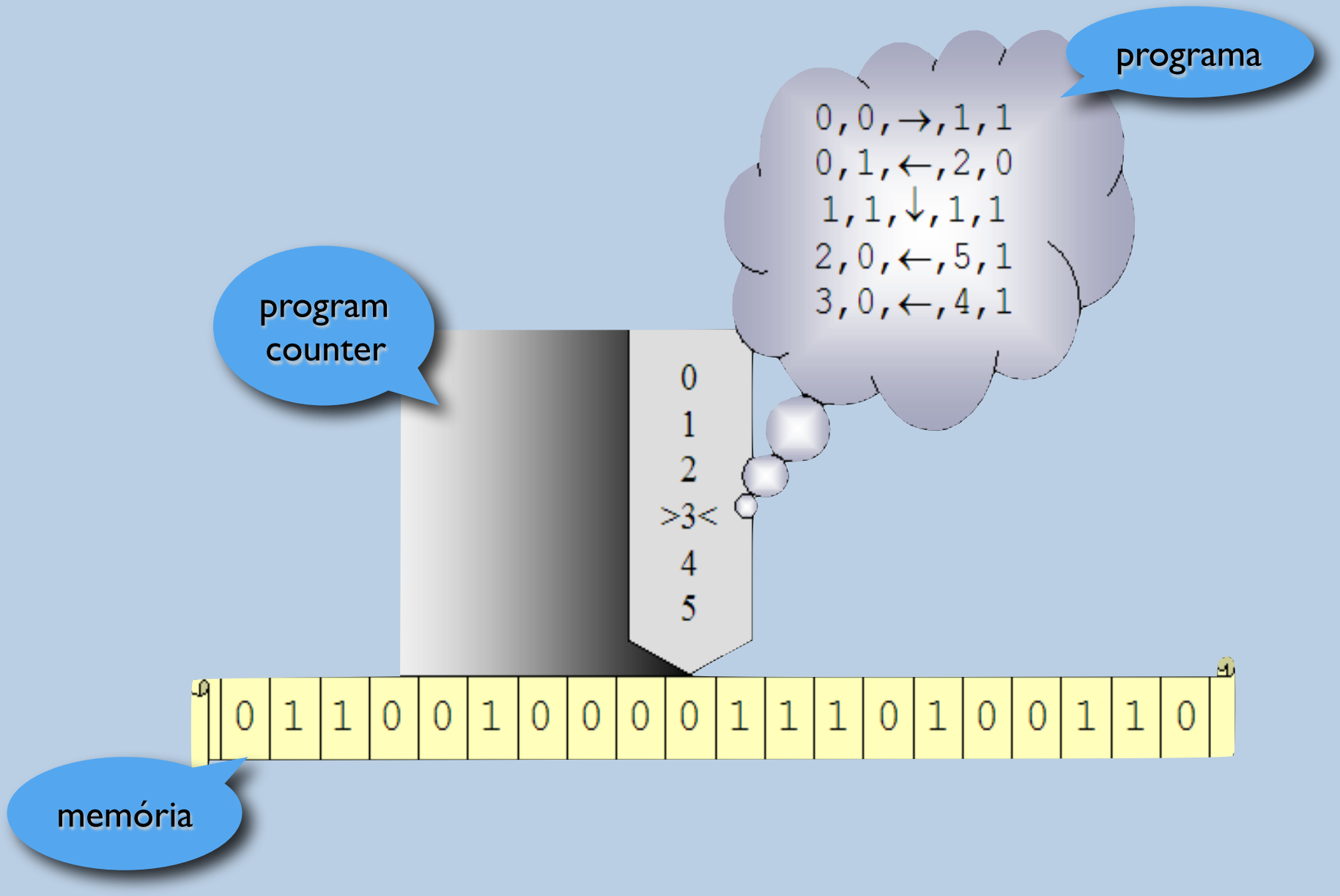
Ambientes de execução: Máquinas Virtuais (software processors)

- Máquina de Turing (Turing, 1931)
  - A primeira …
- SECD (Landin, 1962)
  - Stack, Environment, Code, Dump
- P-code machine (Wirth 1972)
  - Máquina de Pilha, Pascal p-code compiler
- JVM (Sun, 1995)
  - Multi-threaded, tipificada, dedicada à linguagem Java
- CLR (Microsoft, 2000)
  - Multi-threaded, tipificada, Multi linguagem

Máquina de Turing (Turing, 1931)

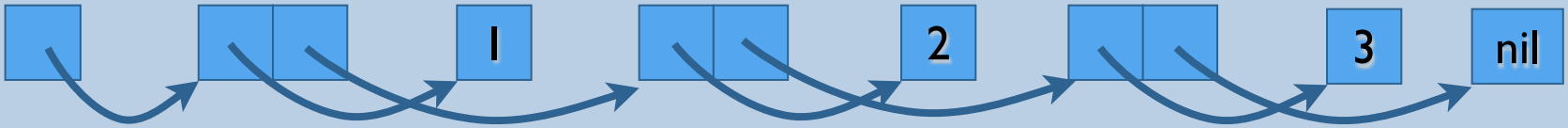


Máquina de Turing (Turing, 1931)



SECD - machine (Landin, 1962)

- A primeira desenhada para implementar o cálculo lambda
- Tem quatro registos a apontar para listas ligadas
  - S : stack
  - E : Environment
  - C : Code
  - D : Dump
- Cada posição da memória pode ter um átomo (12) ou uma lista (um par com dois endereços (o do primeiro elemento e o da lista que se segue)
- A lista (1 2 3) pode ser representada por:



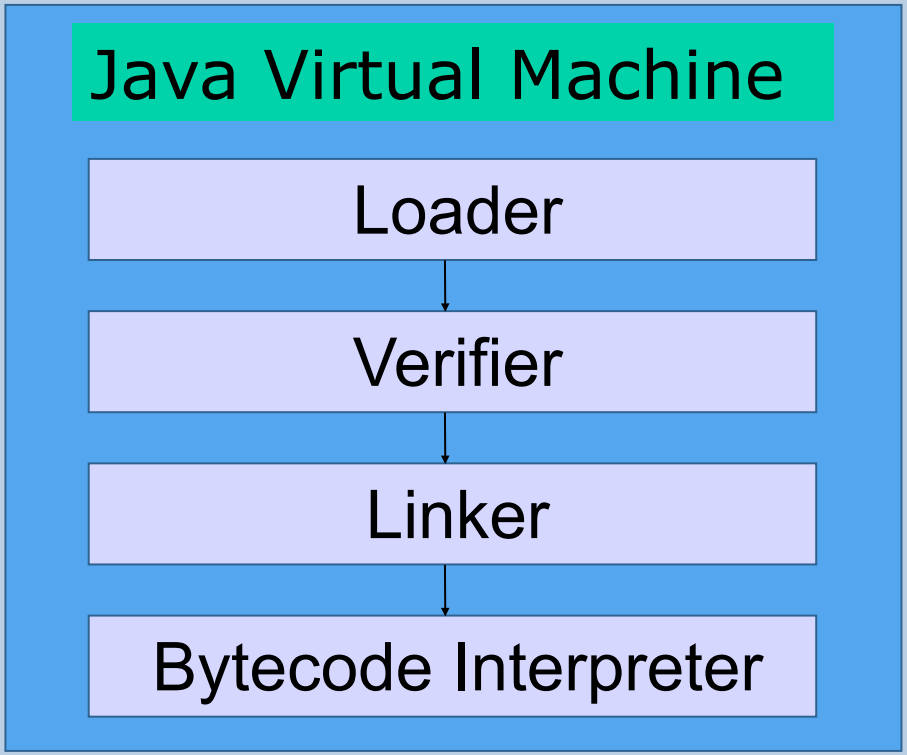
- As instruções: nil, ldc, ld, sel, join, ap, ret, dum, rap

P-code machine (Wirth 1972)

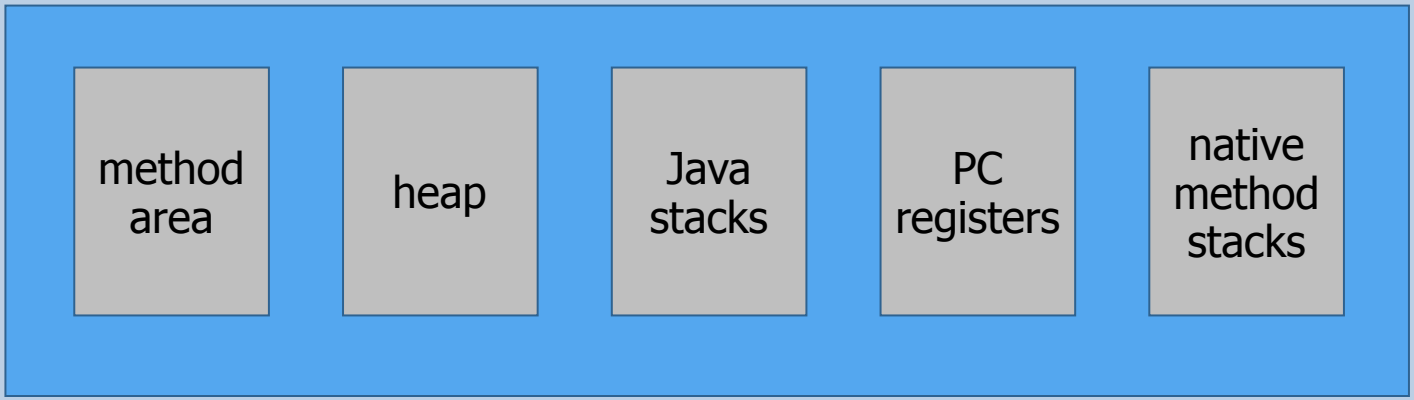
- É uma máquina de pilha o que quer dizer que as instruções da máquina vão buscar os operandos à pilha e colocam o seu resultado de volta na pilha.
- Máquina simples de implementar apenas com uma pilha partilhada entre informação de controlo e dados.

```
procedure interpret;
   const stacksize = 500;

   var
      p,b,t: integer; {program-, base-, topstack-registers}
      i: instruction; {instruction register}
      s: array [1..stacksize] of integer; {datastore}

   function base(l: integer): integer;
      var b1: integer;
   begin
      b1 := b; {find base l levels down}
      while l > 0 do
         begin b1 := s[b1]; l := l - 1
         end;
      base := b1
   end {base};

begin
   writeln(' start pl/0');
   t := 0; b := 1; p := 0;
   s[1] := 0; s[2] := 0; s[3] := 0;
   repeat
      i := code[p]; p := p + 1;
      with i do
      case f of
      lit: begin t := t + 1; s[t] := a end;
      opr: case a of {operator}
           0: begin {return}
                 t := b - 1; p := s[t + 3]; b := s[t + 2];
              end;
           1: s[t] := -s[t];
           2: begin t := t - 1; s[t] := s[t] + s[t + 1] end;
```

Java Virtual Machine (Sun, 1995)

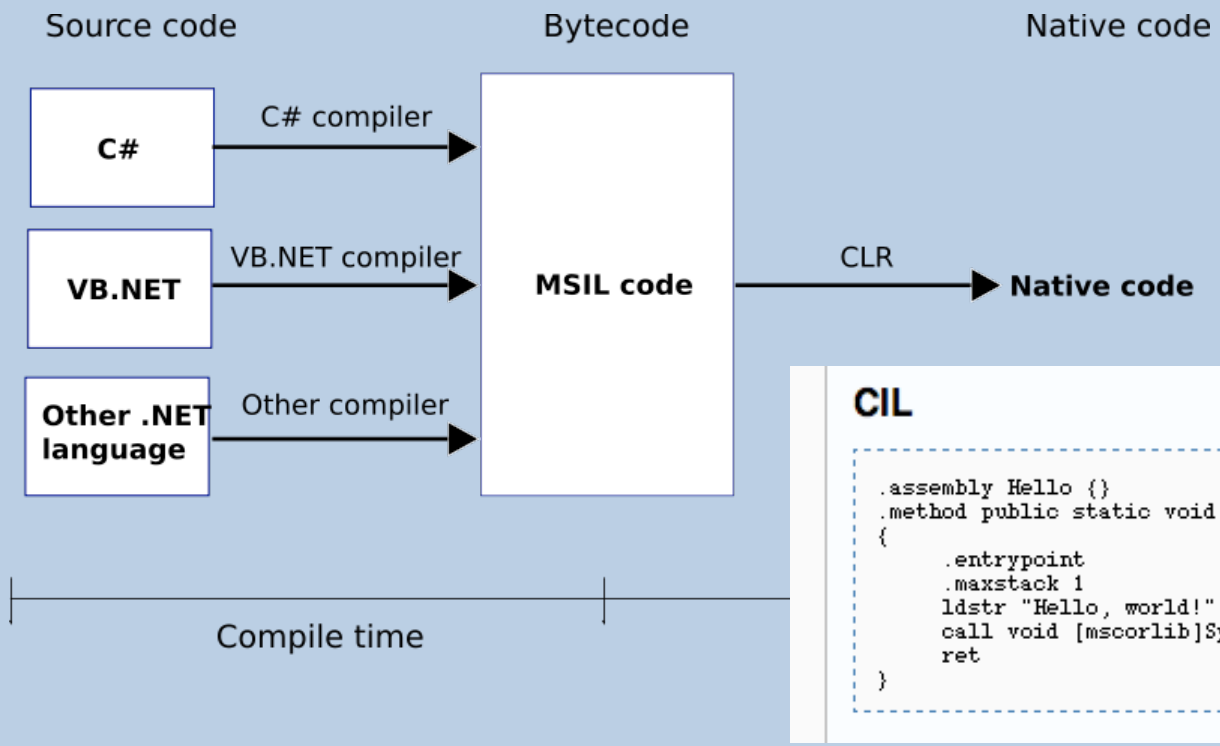



```
// Bytecode stream: 03 3b 84
// 00 01 1a 05 68 3b a7 ff f9
// Disassembly:
iconst_0 // 03
istore_0 // 3b
iinc 0, 1 // 84 00 01
iload_0 // 1a
iconst_2 // 05
imul // 68
istore_0 // 3b
goto -7 // a7 ff f9
```



Common Language Runtime (Microsoft, 2000)

- Máquina de pilha, base para a plataforma Microsoft .NET
- Desenhada para executar programas de várias linguagens.
- CIL - Common Intermediate Language



CIL

```
.assembly Hello {}
.method public static void Main() cil managed
{
      .entrypoint
      .maxstack 1
      ldstr "Hello, world!"
      call void [mscorlib]System.Console::WriteLine(string)
      ret
}
```

Leituras: Ideias a procurar!!

http://en.wikipedia.org/wiki/Turing_machine

http://en.wikipedia.org/wiki/SECD_machine

http://en.wikipedia.org/wiki/P-code_machine

http://en.wikipedia.org/wiki/Common_Language_Runtime

http://en.wikipedia.org/wiki/JVM