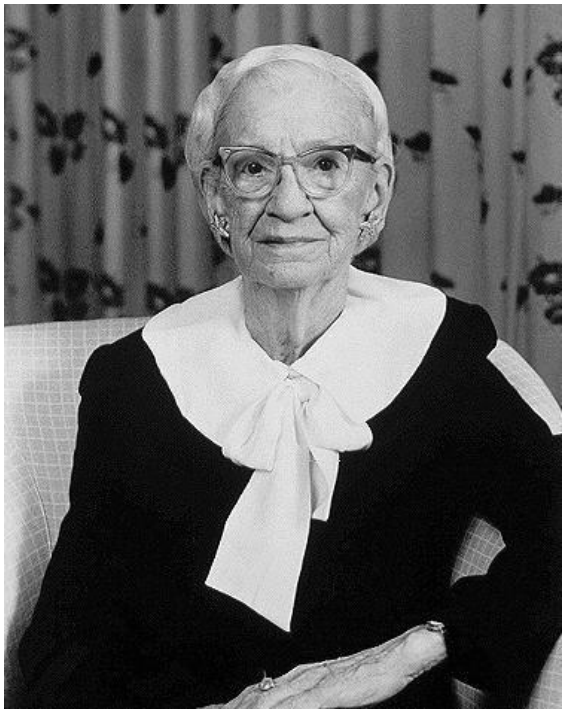
Sintaxe e Semântica

Linguagens de Programação

Prof. Ausberto S. Castro V.
ascv@uenf.br

Linguagens ?

- Introdução
- Como descrever a Sintaxe?
- Métodos Formais:
  - Gramática Livre de Contexto
  - Forma de Backus-Naur
  - BNF Estendedida
  - Grafos
- Análise Sintática
- Gramática de Atributos
- Semântica:
  - Operacional
  - Axiomática
  - Denotacional



Grace M. Hooper
1906-1992
Oficial da Marinha e ex-funcionária da UNIVAC. 3era Programadora do Mark I
Inventou o Primeiro Compilador, 1952
Esteve envolvida no projeto COBOL

Introdução

- **Descrição de uma linguagem: concisa e compreensível**
  - **Apresentação escrita**

```
DO
  READ(8, *, IOSTAT=iostatus) state(case), x(case, 1:nvar), y(case)
  IF (iostatus > 0) CYCLE                ! Error in data
  IF (iostatus < 0) EXIT                 ! End of file

  xx(0) = one                            ! A one is inserted as the first
                                         ! variable if a constant is being fitted.
  xx(1:nvar) = x(case, 1:nvar)           ! New variables and transformed variables
                                         ! will often be generated here.
  yy = y(case)
  CALL includ(wt, xx, yy)
  case = case + 1
END DO

WRITE(*, *)'No. of observations =', nobs
```

FORTRAN

```
l1 = 1; l2 = x/l1

do while ( abs(l1 - l2) > 1e-10 )
   l1 = (l1 + l2)/2.0
   l2 = x/l1
end do
```

C

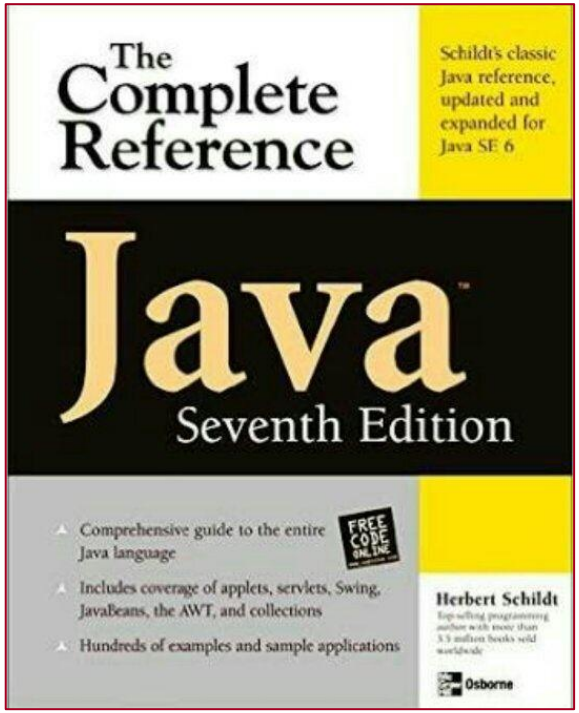
```
#include <stdio.h>
int main()
{
      double number, sum = 0;

      // loop body is executed at least once
      do
      {
            printf("Enter a number: ");
            scanf("%lf", &number);
            sum += number;
      }
      while(number != 0.0);

      printf("Sum = %.2lf",sum);

      return 0;
}
```

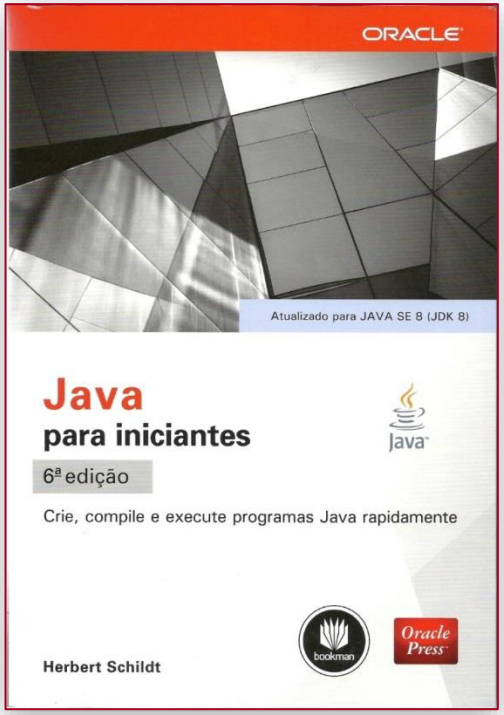
Introdução

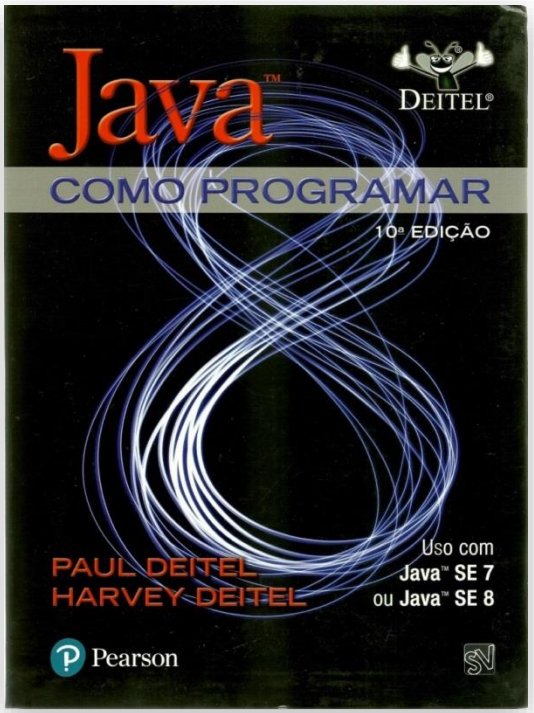
- **Problemas para a descrição de Linguagens**
  - **Diversidade de usuários**
    - **Programadores**
    - **Analistas**
    - **Professores-alunos**

Introdução

- **Problemas para a descrição de Linguagens**
  - **Manual de Referência da linguagem**
    - **Manual : Referência - Usuário**
    - **Livros**
    - **Cursos**

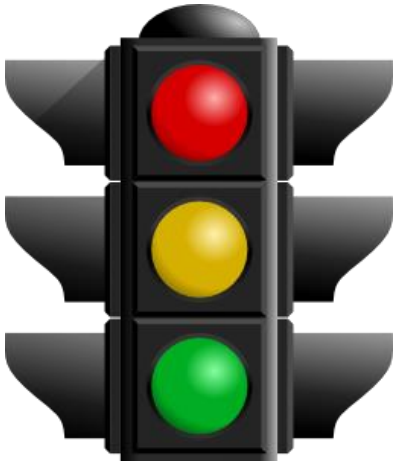






Linguagem Formal

- **Símbolo**
  - Uma *entidade abstrata* que representa uma **ú**nica ideia ou conceito, e utilizada na Matemática e na Computação, na forma de objetos, para construir determinados conjuntos.
    - **a, b, c, d, ...A, B, C, ..., 0,1,2,3,4,5,6,7,8,9**
    - **0, 1**
    - **x, y, z, v, w,**
    - **α, β, γ, δ, ε, π, λ, ω, θ, ϕ, Σ, Ω, Φ, Γ, Π, Ψ**
    - **begin, end, for, while, :=, >,**
    - **□, ∇, ⊗, ⊕, ◊**



- **Alfabeto Σ**
  - **Conjunto finito e não vazio de elementos chamados de *símbolos***
    - **Alfabeto binário**
    - **Alfabeto português**
    - **Alfabeto grego**
    - **Σ = {float, char, int, return, break, switch, if, else, main, <, =, …}**

Linguagem Formal

- **Palavra ω sobre um conjunto Σ**
  - Palavra, string, cadeia de caracteres, cadeia de símbolos
  - Sequencia finita de elementos de Σ
    - aabbabaa sobre Σ = {a,b}
    - 11221112 sobre Σ = {1,2}
    - 01010001001 sobre Σ = {0,1}
    - begin x := 4 end sobre Σ = {begin, end, for, while, :=, 1, 2, 3, 4, ...}

- **Conjunto de palavras, Σ***
  - Conjunto de palavras sobre um alfabeto Σ, definidas recursivamente:
    - A palavra nula $\lambda \in \Sigma^*$
    - Se $\omega \in \Sigma^*$ e $a \in \Sigma$ então $\omega a \in \Sigma^*$

Palavra ω sobre um conjunto Σ e Σ*

```c
#include <stdio.h>
int main()
{
      int n, i, sum = 0;

      printf("Enter a positive integer: ");
      scanf("%d",&n);

      for(i=1; i <= n; ++i)
      {
            sum += i;     // sum = sum+i;
      }

      printf("Sum = %d",sum);

      return 0;
}
```

Símbolos: {...

$\Sigma = \{ \ldots \}$

$\Sigma^* =$

$\omega_1 =$

$\omega_2 =$

$\omega_3 =$

Linguagem

- **Linguagem: conjunto de sequências de caracteres de algum alfabeto**

  X := (Y + 2) / (z – 3*k)

- **As seqüências são chamadas de sentenças ou instruções**

```
IF ( cond) then c1
write(x1,x2,s1,s2)
```

- **Uma linguagem tem sintaxe (formas) e semântica (significados) estreitamente relacionadas**
  - `Z := x + 4` (atribuição, variáveis, soma)

- **Metalinguagem: linguagem usada para descrever outra linguagem**

Sintaxe e Semântica

Sintaxe e Semântica

Sintaxe

FORMA

Expressões
Sentenças
e Unidades de programa

SIGNIFICADO

Semântica

Sintaxe

- **Sintaxe: "como se escreve"**
  - **Forma das expressões, instruções e unidades de programa**

    for(x=2; x<10; x++) **em C**

    for x=2 to 10 do **em Pascal**

  - **As regras de sintaxe especificam quais sequências de caracteres do alfabeto da linguagem estão nela**

  - **Regras que determinam como combinar as palavras**

Sintaxe

- **Sintaxe: "como se escreve"**
  - **Lexemas: são as unidades sintáticas de nível mais baixo**

    `x, y, , 5.34, begin`, a, A, *, +
  - **Tokens: é uma categoria (conjunto, tipo) de lexemas.**

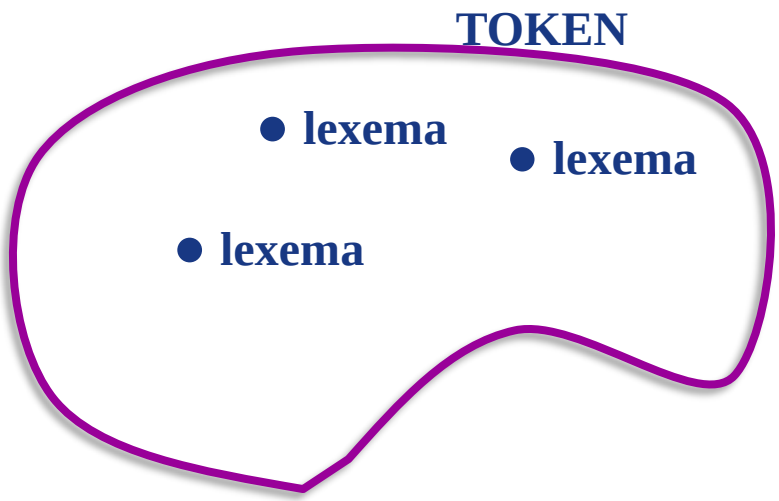
    operador, identificador, palavra-chave

Lexemas e Tokens

```
index = 2 * count + 17;
```

Token → conjunto
Lexema → elemento

Lexemas	*Tokens*
temp	*identificador*
=	*sinal_igual*
2	*int_literal*
*	*mult_op*
count	*identificador*
+	*soma_op*
17	*int_literal*
;	*ponto_e_virgula*



Lexemas e Tokens

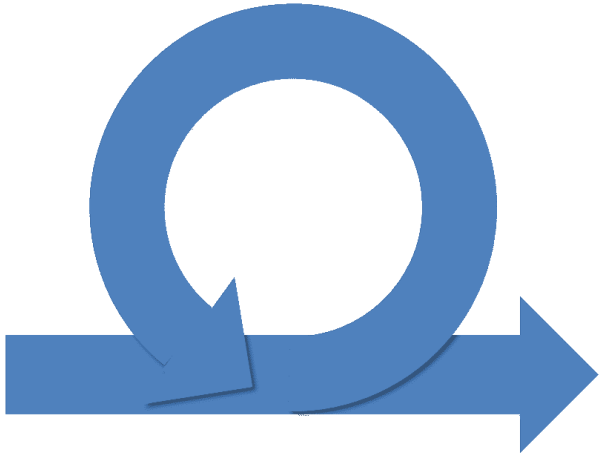
```
program posneg;
uses crt;
var
    no : integer;
begin
    clrscr;
    Write('Enger a number:');
    readln(no);
    if (no > 0) then
        writeln('You enter Positive Number')
    else
        if (no < 0) then
            writeln('You enter Negative number')
        else
            if (no = 0) then
                writeln('You enter Zero');
    readln;
end.
```

Lexemas:

Tokens:

Semântica

- **Semântica: "que significa"**
  - **O significado de expressões, instruções e unidades**
  - **Como os programas se comportam quando são executados no computador**
    - **Laço , iteração, procedimento**



Semântica

- **Significados:**
  - **FOR**
    - **laço para repetir um conjunto de instruções um número FINITO de vezes**
    - **for k = 1:18 { .....}**
  - **WHILE**
    - **Laço para repetir um conjunto de instruções, quando não é conhecido quantas vezes. Para isto se utiliza uma CONDIÇÃO de parada**
    - **while (k < x) { ...}**
  - **Do ... While**
  - **While ... do**

Sintaxe e Semântica

- **A Sintaxe *influencia*:**
  - **Como os programas são *escritos* por programadores**
  - **Como são *lidos* por outros programadores**
  - **Como são *analisados sintaticamente* pelo computador (compilador)**

- **A Semântica *determina*:**
  - **Como os programas são *escritos* (compostos) pelos programadores**
  - **Como os programas são *compreendidos* por outros programadores**
  - **Como os programas são *interpretados* (na execução) pelo computador**

Sintaxe
das Linguagens de Programação

Sintaxe

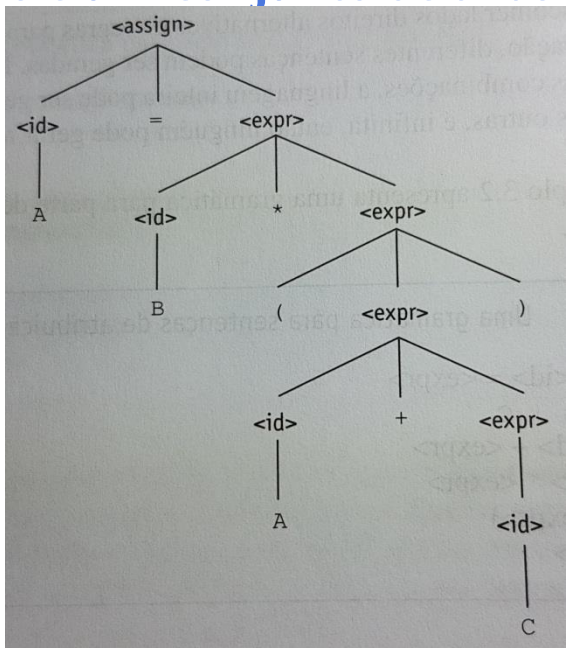
- **É o estudo das estruturas dos programas:**
  - **Estruturas: módulos, funções, laços, condicionais, etc.**
  - **Que programas são *"legais"* (legalmente válidos)**
    - **x = 2 x 2 = = x 2 2 = x**
  - **Quais são as *relações* entre os símbolos e as frases que aparecem dentro de um programa**
    - **<exp> + <exp> a + b "gato" + 15 2.5 + 4**

Classes de Sintaxe

- **Sintaxe concreta**
  - **Uma linguagem de programação é um conjunto de palavras (strings) construídos sobre um alfabeto.**
    - **Pascal = {Begin, end, for, while, x, y, 1, 2, 3, ... X := 5, Y = x-8, }**

- **Sintaxe abstrata**
  - **Uma linguagem de programação é um conjunto de árvores de derivação.**
    - **Árvore binário**

A = B*(A+C)



Sintaxe concreta

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
} else {
   /* statement(s) will execute if the boolean expression is false */
}
```

```
   if (k > 10) {
            x:= 3}
            else {
             y:=40
}
```

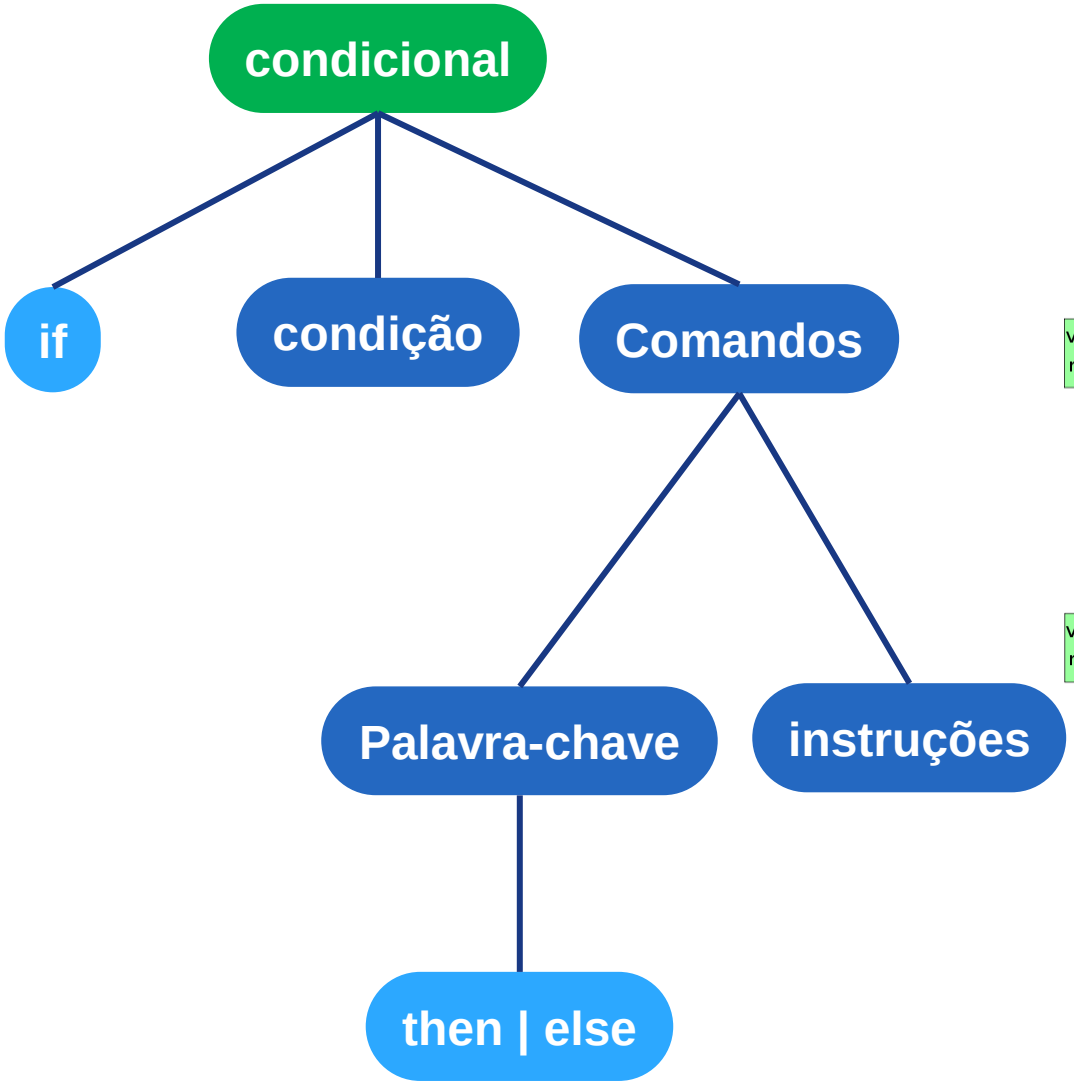
```
for ( init; condition; increment ) {
   statement(s);
}
```

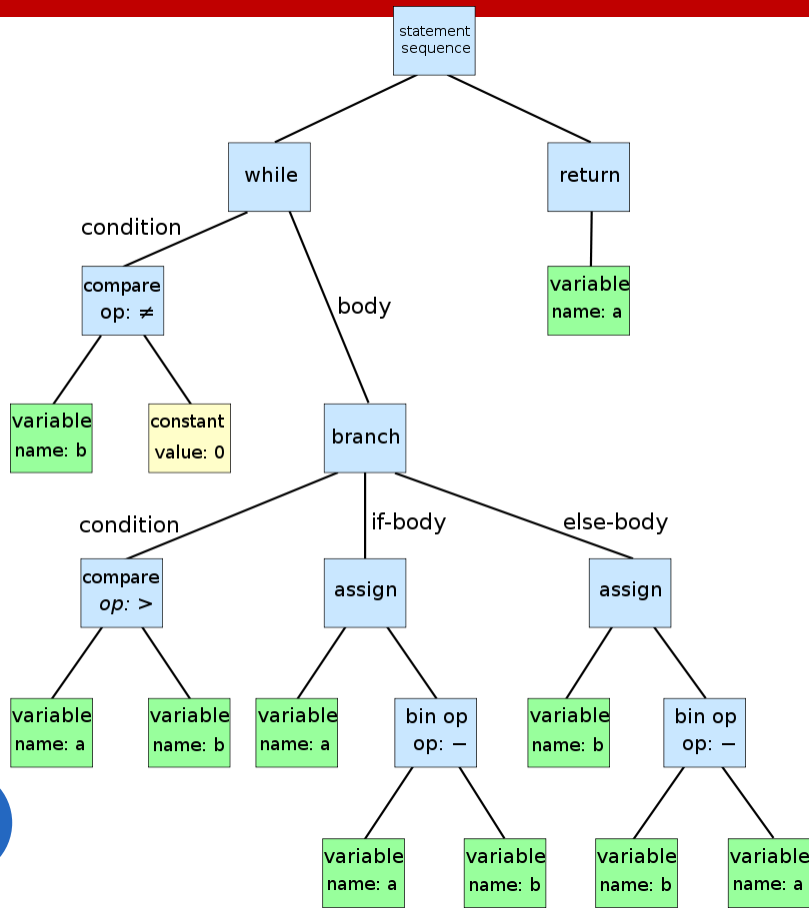
```
for ( i = 0; i <= j; i ++ ) {
          printf("Hello %d\n", i );
 }
```

```
return_type function_name( parameter list ) {
   body of the function
}
```

```
Demo( int par1, int par2)
 {
     int total = 10;
     printf("Hello World");
     total = total + 1;
 }
```

Sintaxe abstrata





Gramática

- Uma *gramática* (livre de contexto) G é uma quádrupla ordenada

$$G = (V, \Sigma, P, S)$$

onde :

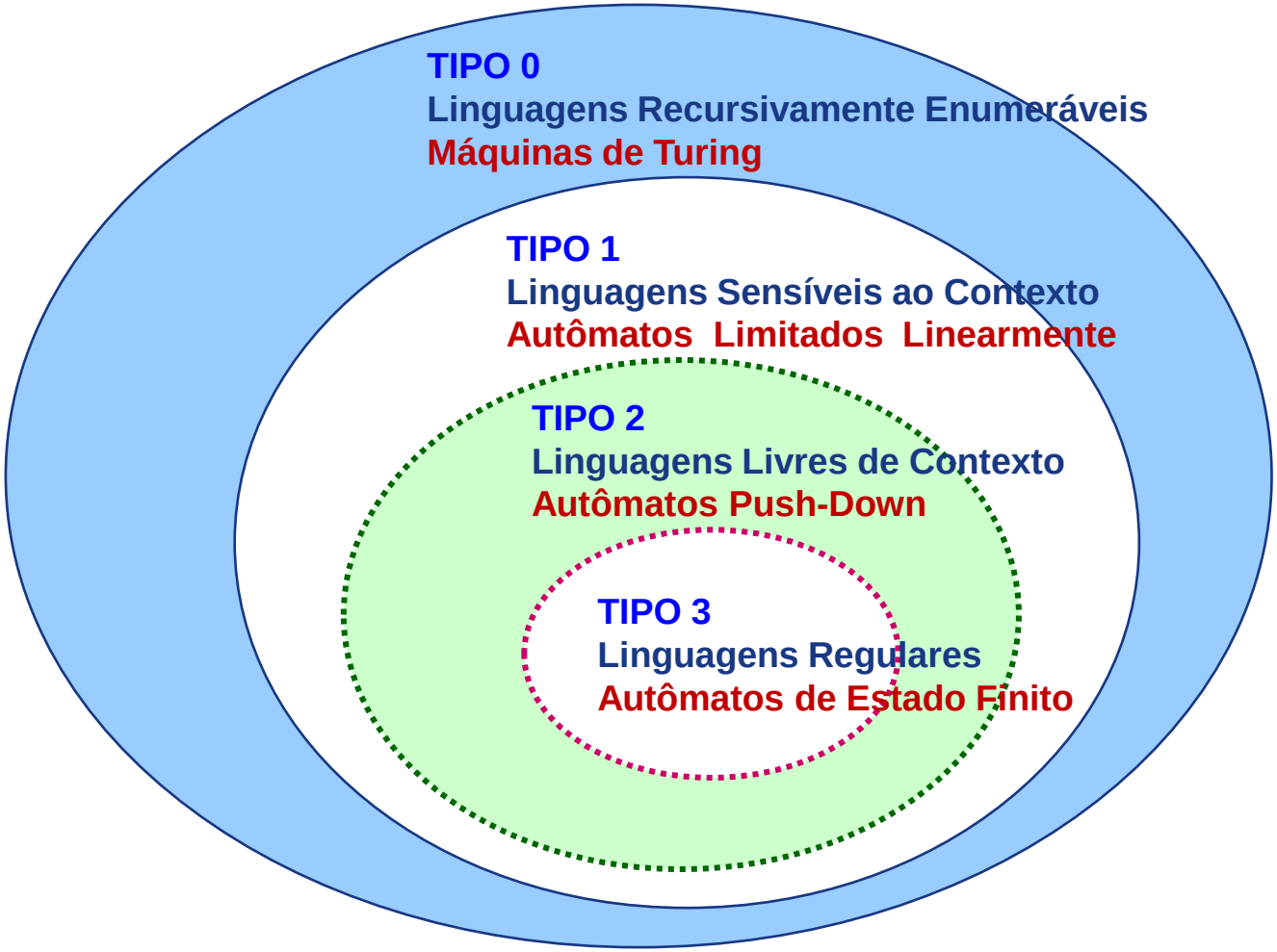
  - V é um conjunto finito de símbolos chamados *variáveis* ou *não-terminais*
  - Σ conjunto finito de símbolos *constantes* ou *terminais* (alfabeto)
  - P conjunto de *regras de produção* p: $A \rightarrow B$
  - S símbolo especial de V chamado *símbolo inicial*
- Gramáticas são usadas para gerar palavras bem formadas a partir de um alfabeto
- A *linguagem de uma gramática*, L(G), é o conjunto de sentenças geradas pela gramática G

```
<program> -> <stmts>
<stmts> -> <stmt> | <stmt> ; <stmts>
<stmt> -> <var> = <expr>
<var> -> a | b | c | d
<expr> -> <term> + <term> | <term> - <term>
<term> -> <var> | const
```

Gramáticas Livre de Contexto

- **Desenvolvidas por Noam Chomsky (lingüista) na década do 50**
- **Seu interesse era a natureza teórica das linguagens naturais**
- **Hierarquia de Chomsky: *4 classes de linguagens***
  - **Tipo 0: Lingaugens Recursivamente Enumeráveis**
    - **Máquina de Turing**
  - **Tipo 1: Linguagens Sensíveis ao Contexto**
    - **Máquina de Turing com memória limitada**
  - **Tipo 2: Linguagens Livres de Contexto**
    - **Linguagens de programação – Linguagens LR(k)**
  - **Tipo 3: Linguagens Regulares**
    - **Autômato Finito**
- **Usadas para descrever a sintaxe das linguagens de programação: *livres de contexto* e *regulares***

Hierarquia de Chomsky



Hierarquia de Chomsky

Gramática	Linguagens	Autômato	Regras de Produção
Tipo-0	Enumerável recursivamente	Máquina de Turing	Sem restrições
Tipo-1	Sensível ao contexto	Máquina de Turing Linear-limitada Não-determinístico	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Tipo-2	Livre de Contexto	Autômato de pilha Não-determinístico	$A \rightarrow \gamma$
Tipo-3	Regular	Autômato de estado finito	$A \rightarrow aB$ $A \rightarrow a$

Forma de Backus-Naur (BNF)

- Inventada por John Backus (1959, grupo ACM-GAMM) para descrever ALGOL 58.
- É uma gramática livre de contexto
- Modificada por Peter Naur para descrever ALGOL 60

- BNF = Backus-Naur Form

- É o método mais popular para descrever concisamente a sintaxe de uma linguagem de programação

Forma de Backus-Naur (BNF)

- Usa abstrações < … > para descrever estruturas sintáticas da linguagem.
  - <atribuição>
  - <OperaçãoBinaria>
  - <ExpressãoLógica>
  - <ListaIdentificadores>
- Uma *definição* de uma estrutura sintática é chamada de regra ou produção.
  - <LadoEsquerdo> → <ladoDireito> : "lado Esquerdo é definido como lado Direito"
- Uma *descrição* BNF é simplesmente um *conjunto de regras* da forma:

  LHS → RHS
- Exemplo: uma *instrução de atribuição* em C, é representada pela abstração <atribuição>.
  - "a abstração <atribuição> é definida como uma instância da abstração <var> seguida do lexema =, seguida de uma instância da abstração <expressão>"

<atribuição> → <var> = <expressão>

BNF: Elementos

- ***Símbolos não-terminais*: são as abstrações**
  - **Sempre se escrevem usando os símbolos < >**
  - **Chamados também de metavariáveis ou classes sintáticas**
- ***Símbolos terminais*: são os lexemas ou tokens das regras**
- **O símbolo separador da produção → ou ::=**
  - **significa *"é definido como"***
- **Símbolos lógicos**
  - **Se utiliza o OU lógico representado pelo símbolo |**

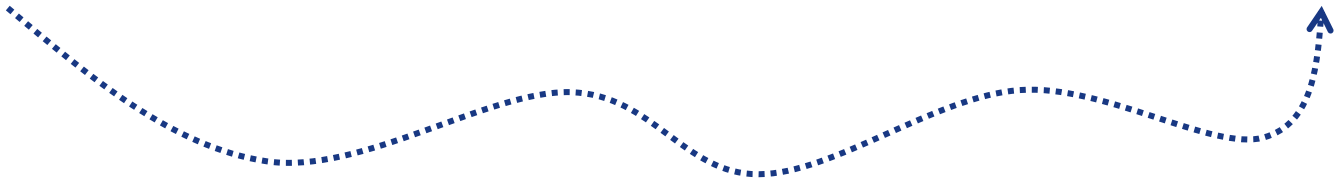
```
<Identificador> ::= <Letra>
                  | <Identificador><Letra>
                  | <Identificador><Digito>

<Letra> ::= a | b | c | ... | y | z

<Digito> ::= 0 | 1 | 2 | ... | 8 | 9
```

x, y
Tabela, xy
Tabela1, x2

Forma de Backus-Naur (BNF)

- **Uma regra é *recursiva* se o LHS aparecer em seu RHS**
- **Descrevendo listas:**

<lista_ident> → identificador | identificador, <lista_ident>



Outro exemplo ....

Forma de Backus-Naur (BNF)

- **Comando SWITCH**

<multipla-escolha> ::= **switch** <var> { <corpo> }

<corpo> ::= **case** <valVar> : <instrucao> ;

<comCase> ; **default** : <instruçao>

<comCase> ::= **case** <valVar> : <instrucao> ;

| **case** <valVar> : <instrucao> ; <comCase>

<instrucao> ::=

```
switch ( valor )
{
        case 1 :
        printf ("Domingo\n");
        break;

        case 2 :
        printf ("Segunda\n");
        break;

        case 3 :
        printf ("Terça\n");
        break;

        case 4 :
        printf ("Quarta\n");
        break;

        case 5 :
        printf ("Quinta\n");
        break;

        case 6 :
        printf ("Sexta\n");
        break;

        case 7 :
        printf ("Sabado\n");
        break;

        default :
        printf ("Valor invalido!\n");
}
```

Outro exemplo...

BNF - Gramáticas e derivações

- Uma BNF é um dispositivo generativo pra definir linguagens
- As sentenças da linguagem são geradas por uma sequência de aplicações das regras, iniciando-se com um símbolo não-terminal chamado de *símbolo de início*.
- Uma geração de sentença é chamada de *derivação*
- Em uma gramática para uma linguagem completa, o símbolo de início representa um **programa completo**
- Cada string de símbolos na derivação é uma *forma sentencial*
- Uma *sentença* é uma forma sentencial que contém unicamente símbolos terminais
- Uma *derivação à extrema esquerda* é aquela na qual o não-terminal substituído é sempre o da extrema esquerda.
- Existem outras formas (ordem) de derivação. A ordem de derivação não tem nenhum efeito sobre a linguagem gerada por uma gramática.
  - O símbolo => significa "deriva"

BNF - Gramáticas e derivações

- **Exemplo de Gramática**

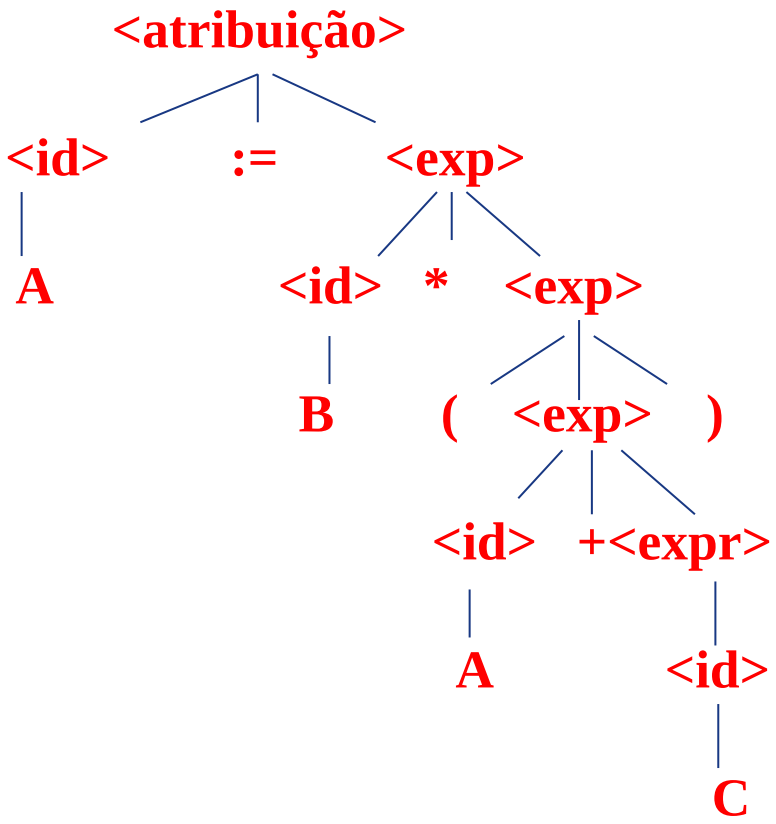
```
<program> -> <stmts>
<stmts> -> <stmt> | <stmt> ; <stmts>
<stmt> -> <var> = <expr>
<var> -> a | b | c | d
<expr> -> <term> + <term> | <term> - <term>
<term> -> <var> | const
```

- **Exemplo de Derivação:** **A = B + cte**

```
<program> => <stmts>
          => <stmt>
          => <var> = <expr>
          => a = <expr>
          => a = <term> + <term>
          => a = <var> + <term>
          => a = b + <term>
          => a = b + const
```

BNF – Árvores de Análise

- **Árvores de Análise** (parse tree): estruturas sintáticas hierárquicas descritas pela gramática
- Cada vértice interno de uma árvore de análise é rotulado com um símbolo não-terminal. Cada folha é rotulada com um símbolo terminal
- Instrução **A := B*(A + C)**
- Uma gramática é **ambígua** quando gera uma sentença para a qual há duas ou mais árvores de análise distintas
- Problemas de ambigüidade
  - Precedência de operadores.
  - Associatividade de operadores



BNF Estendida

- **EBNF = Extended BNF**
- **Usadas para aumentar a legibilidade e capacidade de escrita**
- **Três extensões:**
  - **Parte opcional de um RHS: delimitado por colchetes [ ]**

    <seleção> → if (<expressão>) <instrução> [else <instrução>];
  - **Uso de chaves em um RHS para indicar que parte nelas contida pode ser repetida indefinidamente ou omitida completamente {, }**

    <lista_ident> → <identificador> {, <identificador> }
  - **Opções de múltipla escolha: opções colocadas entre parênteses e separadas pelo operador |**

    <for_stmt> → for <var> := <expr> (to | downto) <expr> do <stmt>

Grafos de Sintaxe

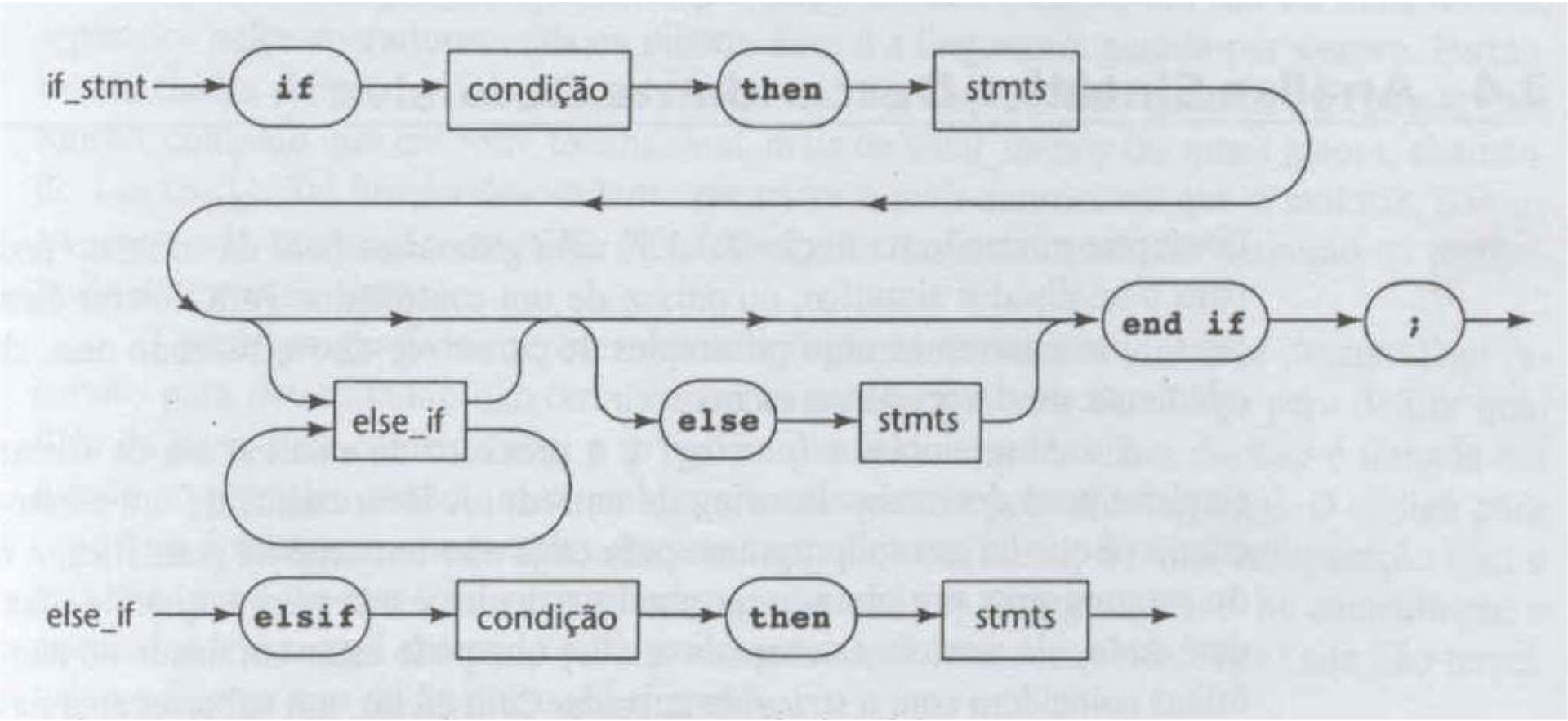
- **Um grafo (vértices + arestas) de sintaxe usa diferentes tipos de vértices para representar símbolos terminais e não terminais**
- **Os vértices retangulares contêm os nomes das unidades sintáticas (não-terminais). Os círculos ou elipses contêm símbolos terminais**
- **Exemplo:** `<else_if>` → **`elseif`** `<condição>` **`then`** `<stmts>`

- **Exemplo:**

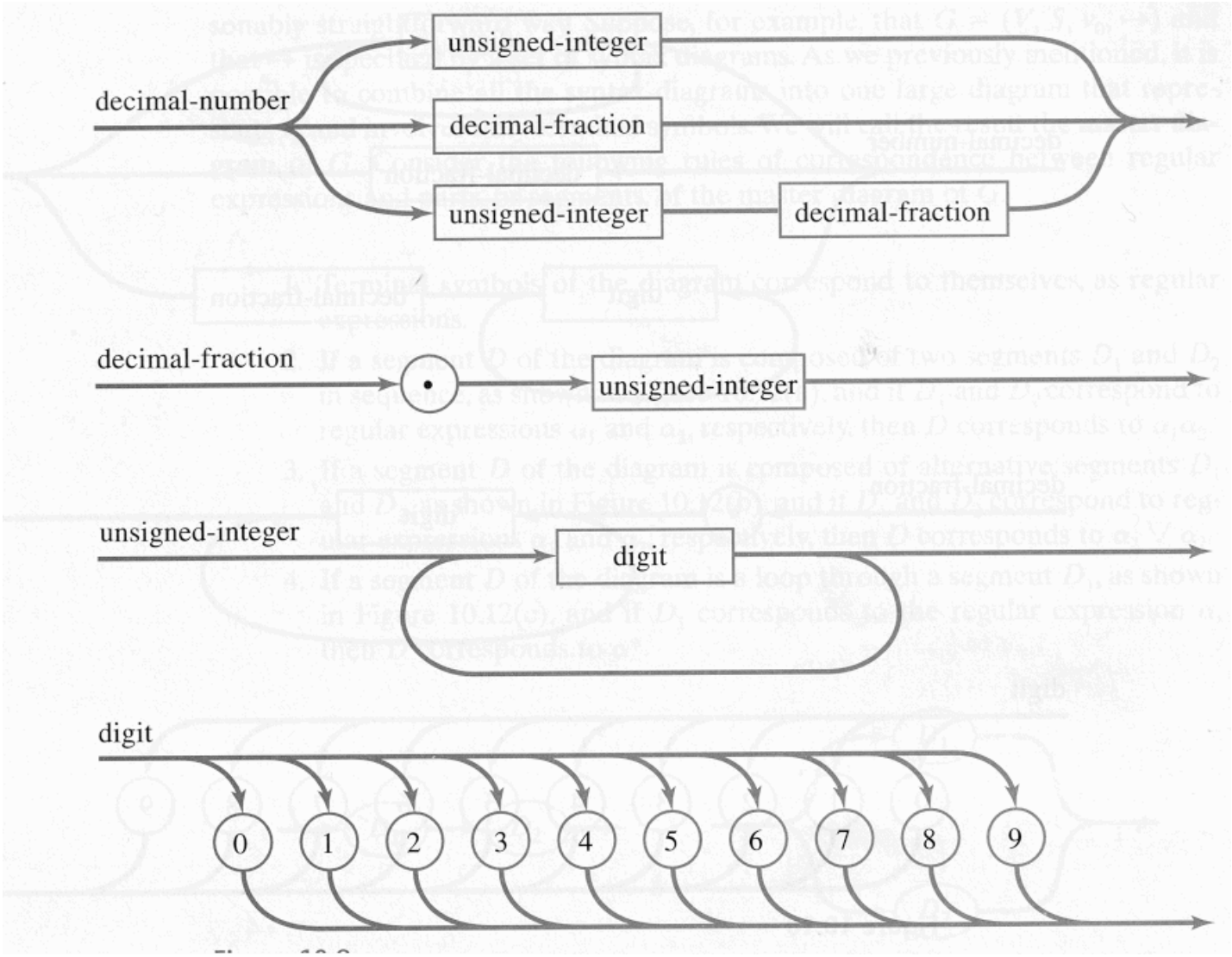
`<if_stmt>` → **`if`** `<condição>` **`then`** `<stmts>` { `<else_if>`}
[**`else`** `<stmts>`] **`end if`**

Grafos de Sintaxe - Exemplo

<if_stmt> → if <condição> then <stmts> { <else_if>}
[else <stmts>] end if



Grafos de Sintaxe - Exemplo



Sintaxe e Semântica Estática

- **Características de uma linguagem difíceis de descrever com BNF.**
  - **Exemplo: regras de compatibilidade de tipos (Java)**
    - **Inteiro ← Real ?**
  - **Declaração de variáveis antes de ser referenciadas (Pascal)**
  - **Nome da função = nome do arquivo (Matlab)**
- **Semântica estática: um tipo de sintaxe**
  - **Conjunto de regras relacionadas com as formas legais dos programas**
  - **Geralmente declaram restrições de tipo**
  - **Estática: a sua análise é feita na compilação**
- **Mecanismos: Gramática de Atributos**

Gramática de Atributos

- São gramáticas com:
  - adição de atributos,
  - funções de computação de atributos, e
  - funções predicadas
- Os *atributos* são semelhantes a variáveis na medida em que podem ter valores atribuídos a eles
- As *funções de computação de atributos* (funções semânticas) são associadas a regras gramaticais para especificar como os valores dos atributos são computados.
- As *funções predicadas*: declaram a parte da sintaxe e as regras semânticas estáticas da linguagem

Gramática de Atributos - Exemplo

1. Regra de sintaxe: <atribuição> → <var> := <expr>
   Regra semântica: <expr>.tipo_esperado ← <var>.tipo_efetivo
2. Regra de sintaxe: <expr> → <var>[2] + <var>[3]
   Regra semântica: <expr>.tipo_efetivo ← if (<var>[2].tipo_efetivo = int) e
   (var[3].tipo_efetivo = int)
   then int
   else real
   end if
   Predicado: <expr>.tipo_efetivo = <expr>.tipo_esperado
3. Regra de sintaxe: <expr> → <var>
   Regra semântica: <expr>.tipo_efetivo ← <var>.tipo_efetivo
   Predicado: <expr>.tipo_efetivo = <expr>.tipo_esperado
4. Regra de sintaxe: <var> → A | B | C
   Regra semântica: <var>.tipo_efetivo ← look-up(<var>.string)

Semântica
das Linguagens de Programação

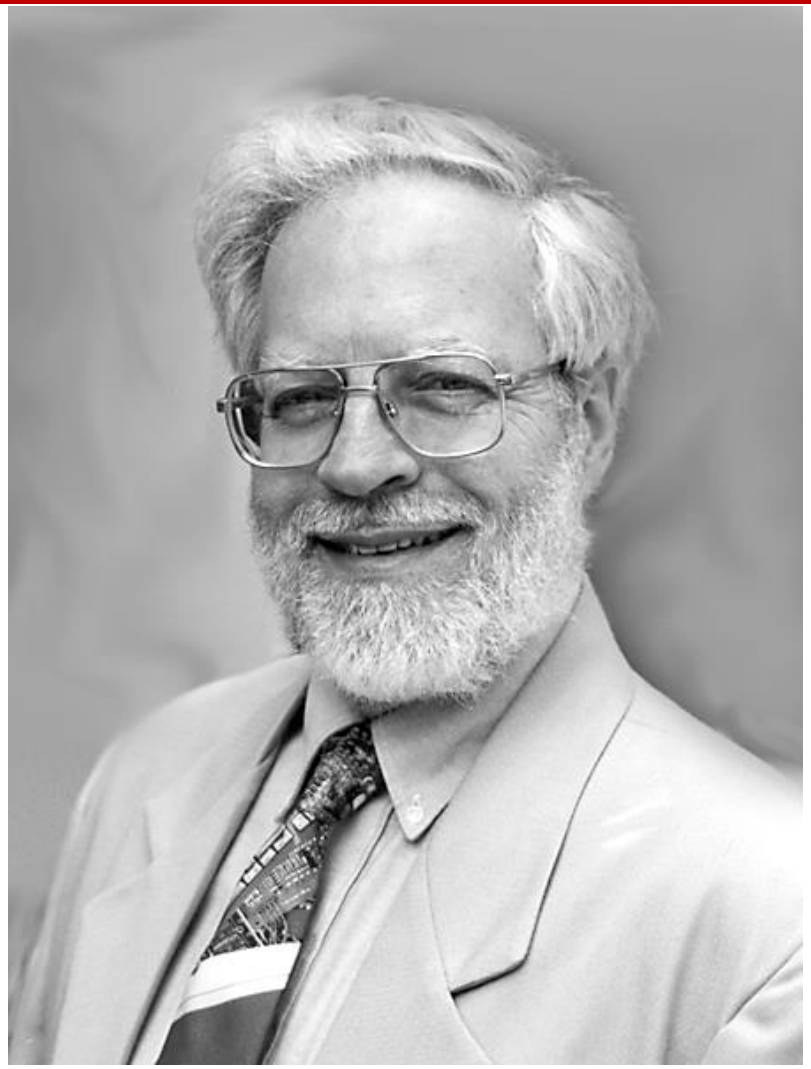
Semântica

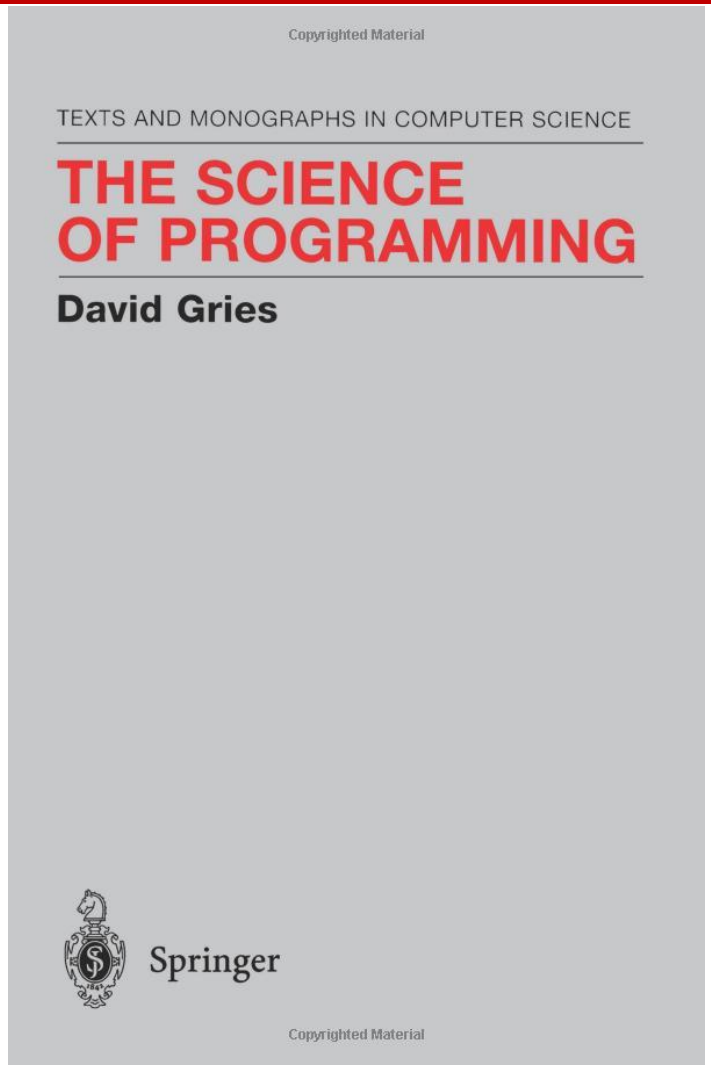
- **É a explicação do significado dos programas "legais" escritos numa linguagem de programação**
  - **É estudo do comportamento que produzem os programas quando são executados pelo computador.**
- **Classes**
  - **Semântica Estática**
    - **Propriedades que podem ser determinadas por inspeção do programa e que não mudam durante a execução**
    - **Exemplo: identificação e verificação de tipos,**
    - **int x; ➔ x será sempre inteiro**
  - **Semântica Dinâmica**
    - **Propriedades que podem mudar durante a execução do programa**
    - **Exemplo: atribuição de valores a áreas particulares de memória, atualização do registrador PC**
    - **int *x ➔ endereço de x em determinado momento da execução**

Abordagens Semânticas

Caracterização	Semântica		
	Axiomática	Denotacional	Operacional
Objetos manipulados	Asserções	Domínios e funções semânticas	Estados abstratos de máquina
Significado das frases da linguagem	Relações entre asserções {P} S {Q}	Denotações F: Dom → denot	Funções de Transição de Estado
Verificação de programas	Dedução no sistema lógico de Hoare	Indução de Ponto Fixo	Indução Computacional

Semântica Axiomática





David Gries
The Science of Programming

Semântica Axiomática

- **Define a semântica de uma linguagem de programação através de um *conjunto de axiomas e regras de inferência*, as quais permitem a prova de propriedades de programas (por exemplo, correção)**
- **Nesta abordagem**
  - **Cada instrução S de um programa é precedida e seguida por uma expressão lógica que especifica restrições a variáveis**

    {P} S {R}
  - **As expressões lógicas são chamadas de predicados ou asserções**
  - **P = pré-condição**
  - **R = pós-condição**

{pré-condição} programa {pós-condição}

Semântica Axiomática

- ***Estado*** **de uma computação**
  - **Conjunto de valores atuais de cada uma das variáveis que ocorrem no programa**
- **A computação começa em um *estado inicial* e termina em um *estado final*.**
- **A condição que caracteriza o conjunto de TODOS os estados iniciais de modo que pela sua ativação, o estado final satisfará a pós-condição, é chamada de *pré-condição mais fraca* (weakest pre-condition, wp)**

$$\{wp(S,R)\}\ S\ \{R\}$$

- **A pré-condição mais fraca é a menos restritiva que garantirá a validade de pós-condição associada**

Semântica Axiomática

- **Exemplo**
  - **Instrução: soma = 3*z + 1**

    { wp } soma = 3*z + 1 {soma > 1}
  - **pré-condições válidas:**

    {z > 10} **{z > 100}** **{z > 500}**
  - **Pré-condição mais fraca wp: { z > 0 }**

    Soma = 3*z + 1 > 1

    ⇔ 3*z > 0

    ⇔ z > 0

Semântica Axiomática

- **Atribuição** $\{R_{x \to E}\}\ x = E\ \{R\}$
  - $R_{x \to E}$: em R, todas as instâncias de x substituídas por E
  - Exemplo: {wp} x = 2*y -3 { x > 25} wp = {y>14}
- **Seqüência**
  - Exemplo: {wp} y=3*x + 1; x = y+3 {x < 10} wp={x <2}
- **Seleção**

$$\frac{\{P1\}\ S1\ \{P2\},\quad \{P2\}\ S2\ \{P3\}}{\{P1\}\ \ S1;S2\ \ \{P3\}}$$

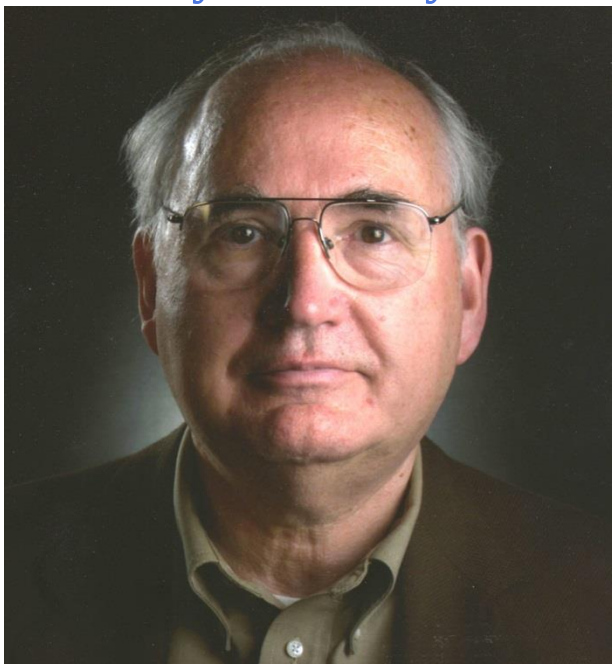
  - Exemplo: {wp} if (x>0) then y = y – 1 else y = y +1 {y > 0}

wp = {y > 1}

$$\frac{\{B \text{ and } P\}\ S1\ \{R\},\ \{(\text{not } B) \text{ and } P\}\ S2\ \{R\}}{\{P\}\ \text{if B then S1 else S2}\ \{R\}}$$

Semântica Denotacional

- **Dana S. Scott**
  - **"Domains for Denotational Semantics", 1982**
  - **Carnegie Mellon University, Pittsburgh**
- **Carl A. Gunter**
  - **"Semantic Domains", 1990**
  - **University of Pennsylvania**



Dana S Scott

Carl A. Gunter

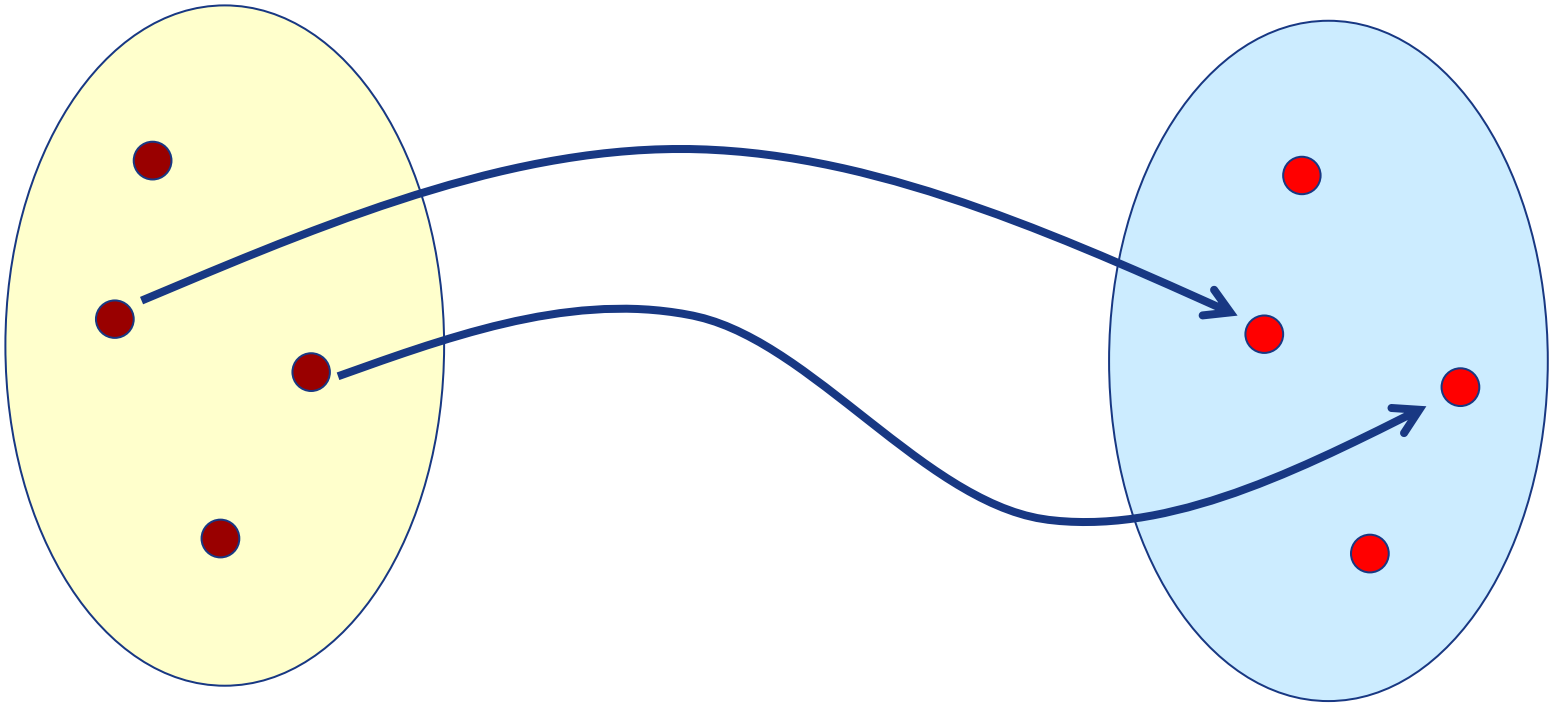
Semântica Denotacional

- **É o método mais rigoroso para descrever o significado de programas**
- **Esta fundamentada na Teoria de Funções Recursivas**
- **Objetivo: Realizar um mapeamento entre os programas escritos em uma linguagem e objetos matemáticos**

  Programas ⟶ obj.matemáticos

  - **Definir para cada *entidade da linguagem*, um *objeto matemático* e uma *função***
  - **Os objetos matemáticos são rigorosamente definidos**
  - **Estes objetos representam o significado exato de suas entidades que representam**
  - **O método é denominado *denotacional* porque os objetos *denotam* o *significado de suas entidades sintáticas* correspondentes**
- **Elementos principais da abordagem**
  - **Funções semânticas**
  - **Domínios semânticos**

Semântica Denotacional



Entidades Sintáticas de uma Linguagem de Programação

Objetos ou entidades matemáticos bem definidos

Semântica Denotacional

- **Consideremos as *expressões*: uma parte de um texto de programa "é realmente" um número (Ex. 3x + sin(2y))**
- **Definamos uma <u>função</u> (a cada x exatamente um f(x)):**

$$[\text{ -- }] : Exp \rightarrow \mathbb{N}$$

$$E \rightarrow [E] = n$$

$\mathbb{N}$ é chamado de *domínio semântico* de *Exp*

- **Para cada construção sintática de uma linguagem necessitamos um domínio semântico**
- **A construção e estudo de tais domínios é chamada de Teoria dos Domínios**
  - **Teoria dos Domínios de Scott**

Semântica Denotacional

- **Definição Semântica Denotacional: 5 partes**
  - **Categorias sintáticas**
  - **Gramática BNF**
    - **Define a estrutura das categorias sintáticas**
  - **Domínio de valores**
    - **As entidades matemáticas**
  - **Funções Semânticas**
    - **As signaturas para mapeamentos da sintaxe para os domínios**
  - **Equações Semânticas**
    - **As regras que definem as funções semânticas (mapeamentos)**

Semântica Denotacional

- **Categorias Sintáticas**
  - D in Digits (dígitos decimais)
  - N in Num (números naturais decimais)
- **Sintaxe BNF**
  - D ::= 0 | 1 | ... | 9
  - N ::= D | N D
- **Domínios de valores**
  - Nat = { 0, 1, 2, 3, 4, ... } Números Naturais
- **Funções Semânticas**
  - DD : Digits --> Nat
  - MM : Num --> Nat
- **Equações Semânticas**
  - DD [0] = 0
  - DD [1] = 0
  - ...
  - DD [9] = 9
  - MM [D] = DD [D]
  - MM [N D] = 10 * MM [N] + MM [D]

Numeral é um Dígito
Número é Natural

Semântica Denotacional

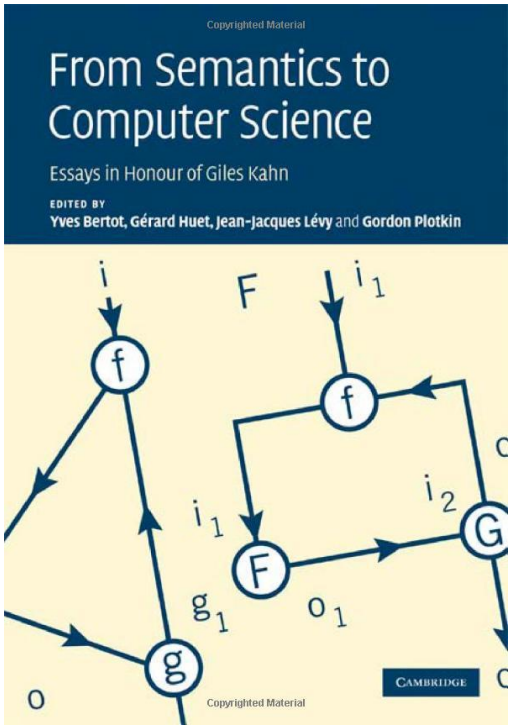
- **Domínios Primitivos**
  - B = { true, false}
  - X = {a , b, c, …}
  - N = {0, 1, 2, 3, …}
- **Domínios Não-Primitivos**
  - D1 x D2 (Produto cartesiano)
  - D1 ∪ D2 (União)
  - D1 $\longrightarrow$ D2 (espaço de funções)
  - D* = D ∪ D ∪ D … ∪ D

Semântica Denotacional

- Pode ser utilizada para provar a correção de programas
- Fornece uma rigorosa maneira de pensar acerca dos programas
- Pode ser de grande ajuda nos projetos de linguagens
- Tem sido utilizada nos sistemas de geração de compiladores.

Semântica Operacional

- **Gordon Plotkin**
  - **"A Structural Approach to Operational Semantics",**
  - **University of Aarhus, Denmark, 1981**
  - **University of Edinburgh**

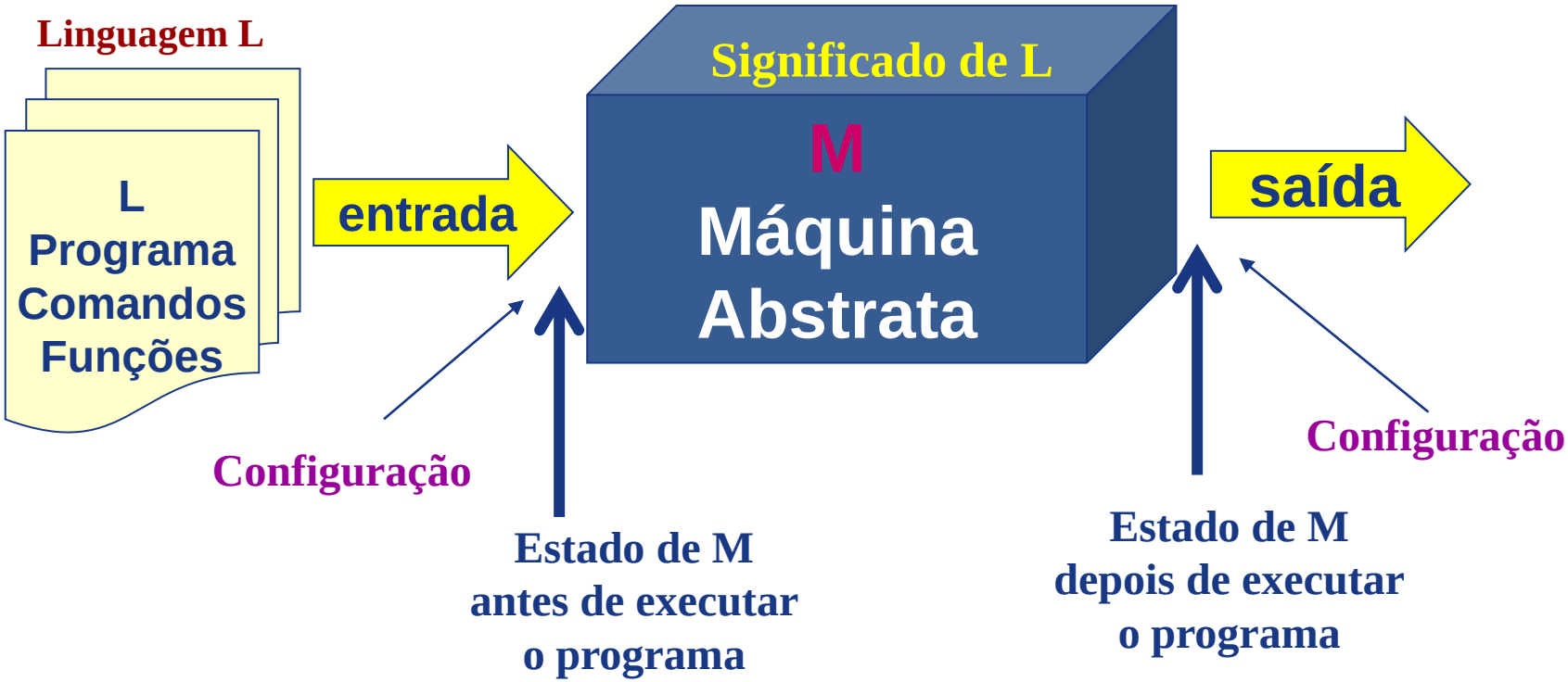


From logic to computer science

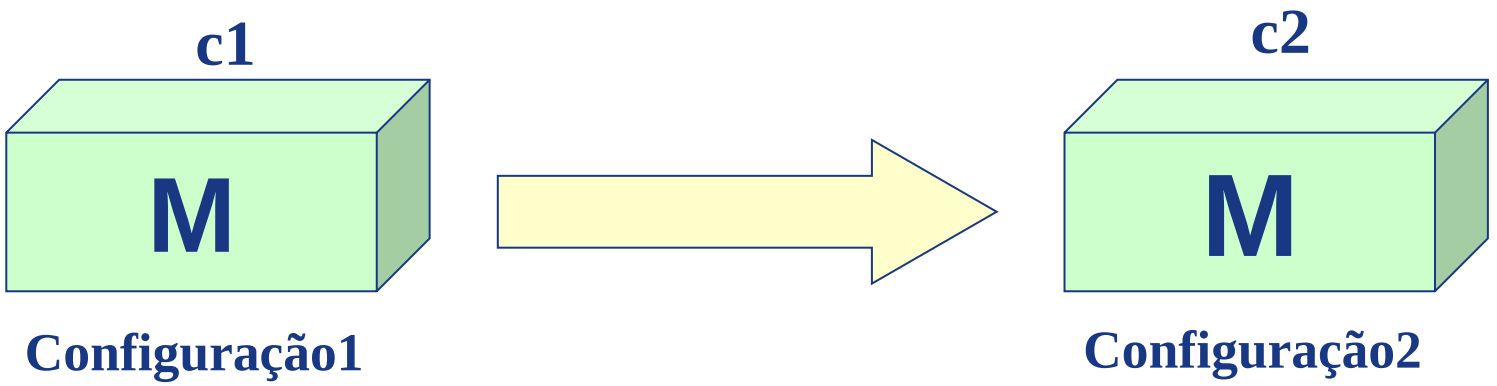
Semântica Operacional

- **Define uma linguagem em termos de uma *operação de uma máquina* (possivelmente abstrata) executando programas**
- **O significado de *frases de programas* são dados em termos de as *etapas da computação* que elas poder ter no momento da execução do programa.**
- **Esta relacionada diretamente com uma implementação**
- **Semântica Operacional é descrita utilizando:**
  - **Um Sistema de Transição (entre configurações), ou**
  - **Uma Máquina Abstrata (com vários estados)**
    - **A Structural Approach to Operational Semantics – Gordon Plotkin, 1981**

Semântica Operacional



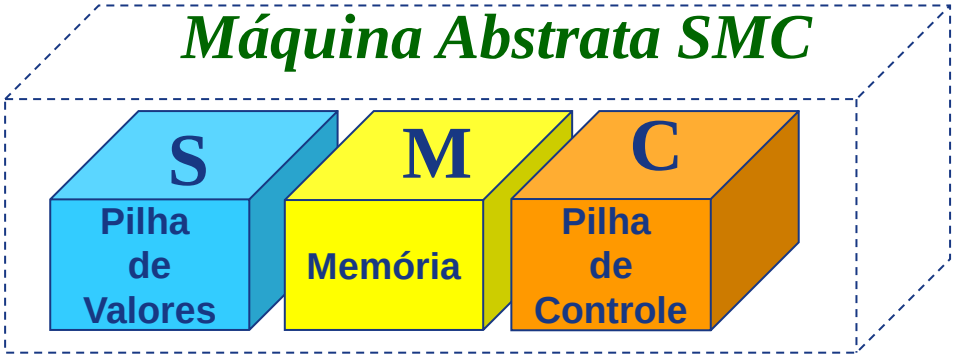
Semântica Operacional



Sistema de transição

- Configuração ou estado inicial
- Configuração ou estado terminal

Semântica Operacional



Configuração típica (estado) : $< S, M, C >$

Soma de dois inteiros:

$<e, M, 2 + 3 > \Rightarrow <e, M, 2\ 3 + > \Rightarrow <2, M, 3 + > \Rightarrow <3\ 2, M, + > \Rightarrow <5, M, e>$

Comando de atribuição:

C := I

$<m\ v\ S, M, := C> \Rightarrow <S, M[m/v], C>$

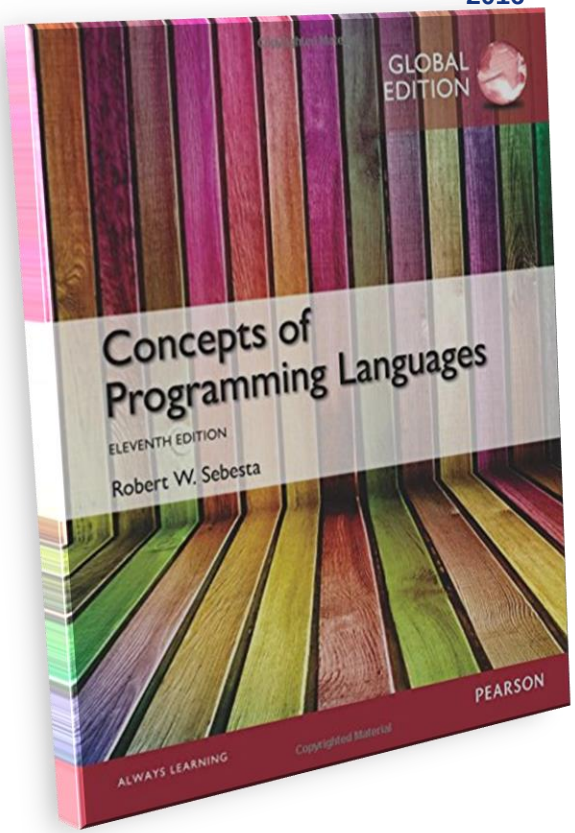
M[n/v]: trocar M(v) por n

$<e, <z, M(z)>, z := 4> \Rightarrow <z, <z, M(z)>, 4 := > \Rightarrow <4\ z, <z, M(z)>, := > \Rightarrow <e, <z, 4>, e>$

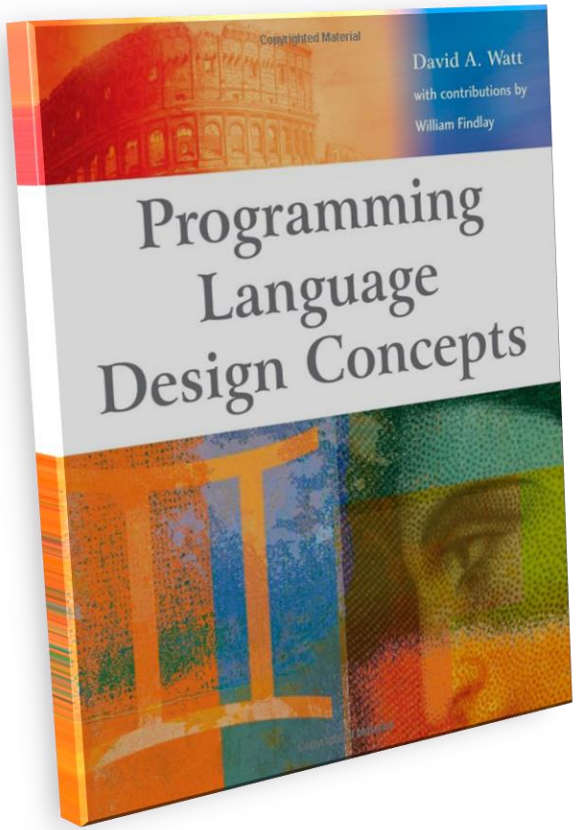
Bibliografia

Bibliografia Complementar

Pearson Education Limited; 11 edition 2016



Wiley; 1 edition (May 21, 2004)



http://www.levenez.com/lang/

Prof. Dr. Ausberto S. Castro Vera
Ciência da Computação
UENF-CCT-LCMAT
Campos, RJ

ascv@computer.org
ascv@uenf.br