2 PROGRAMAS, MÁQUINAS E COMPUTAÇÕES

2 PROGRAMAS, MÁQUINAS E COMPUTAÇÕES

Formalização dos conceitos

- Existem diferentes computadores, com diferentes arquiteturas e existem diversos tipos de linguagens de programação,
- Modelos matemáticos simples.
- Programas e máquinas são tratados como entidades distintas, mas complementares e necessárias para a definição de computação.

O conceito de *programa*

- um conjunto de operações e testes compostos de acordo com uma estrutura de controle.
- O tipo de estrutura de controle associada determina uma classificação de programas como:

  ⇒*monolítico*: baseada em desvios condicionais e incondicionais;

  ⇒*iterativo*: possui estruturas de iteração de trechos de programas;

  ⇒*recursivo*: baseado em sub-rotinas recursivas.

O conceito de *máquina*

- interpreta os programas de acordo com os dados fornecidos.
- é capaz de interpretar um programa desde que possua uma interpretação para cada operação ou teste que constitui o programa.

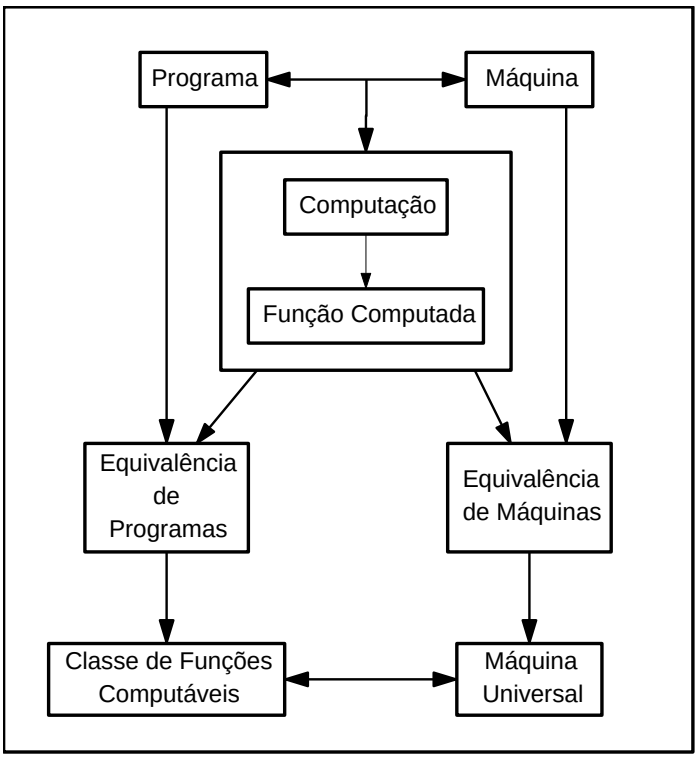
Computação e função computada.

- *Computação* é um histórico do funcionamento da máquina para o programa, considerando um valor inicial.
- *Função computada* é uma função (parcial) induzida a partir da máquina e do programa dados. É definida sempre que, para um dado valor de entrada, existe uma computação finita (a máquina pára).

Equivalência de programas e de máquinas

- *programas fortemente equivalentes*: se as correspondentes funções computadas coincidem para qualquer máquina;
- *programas equivalentes*: se as correspondentes funções computadas coincidem para uma dada máquina;
- *máquinas equivalentes*: se as máquinas podem simular umas às outras.

Resumo:



2.1 Programas

Um programa pode ser descrito como um conjunto estruturado de instruções que capacitam uma máquina aplicar sucessivamente certas operações básicas e testes em uma parte determinada dos dados iniciais fornecidos, até que esses dados tenham se transformado numa forma desejável.

Estrutura de controle de operações e testes.

- *Estruturação Monolítica*. É baseada em desvios condicionais e incondicionais, não possuindo mecanismos explícitos de iteração, sub-divisão ou recursão.
- *Estruturação Iterativa*. Possui mecanismos de controle de iterações de trechos de programas. Não permite desvios incondicionais.
- *Estruturação Recursiva*. Possui mecanismos de *estruturação* em sub-rotinas recursivas. Recursão é uma forma indutiva de definir programas. Também não permite desvios incondicionais.

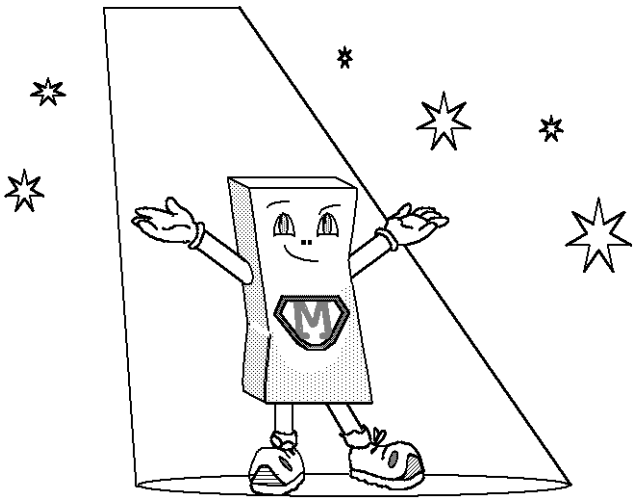
Composição de Instruções

- *Composição Seqüencial*. A execução da operação ou teste subseqüente somente pode ser realizada após o encerramento da execução da operação ou teste anterior.
- *Composição Não-Determinista*. Uma das operações (ou testes) compostas é escolhida para ser executada.
- *Composição Concorrente*. As operações ou testes compostos podem ser executados em qualquer ordem, inclusive simultaneamente. Ou seja, a ordem de execução é irrelevante.

Instruções: Operações e Testes.

- Não é necessário saber qual a natureza precisa das operações e dos testes que constituem as instruções. Serão conhecidas pelos seus nomes.
  - Identificadores de Operações: F, G, H, ...
  - Identificadores de Testes: T_1, T_2, T_3, ...
- um teste é uma operação de um tipo especial a qual produz somente um dos dois possíveis valores verdade, ou seja, *verdadeiro* ou *falso*, denotados por: v e f, respectivamente.
- uma operação que não faz coisa alguma, denominada: *operação vazia*, denotada pelo símbolo √.

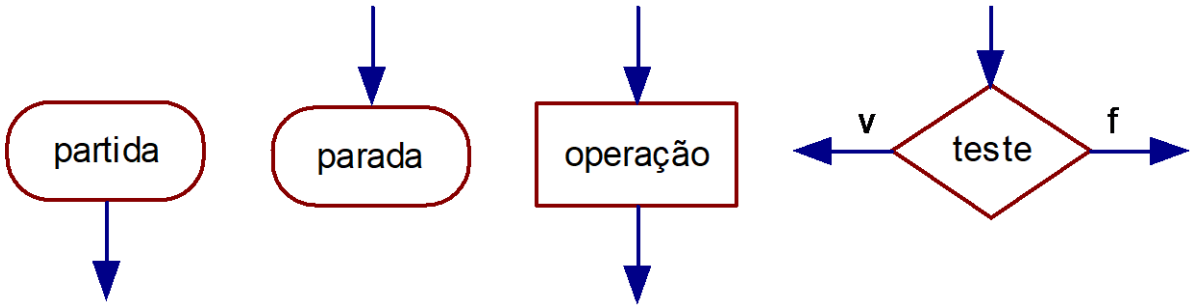
2.1.1 Programa Monolítico



Um *programa monolítico* é estruturado, usando desvios condicionais e incondicionais, não fazendo uso explícito de mecanismos auxiliares de programação. A lógica é distribuída por todo o bloco (monólito) que constitui o programa.

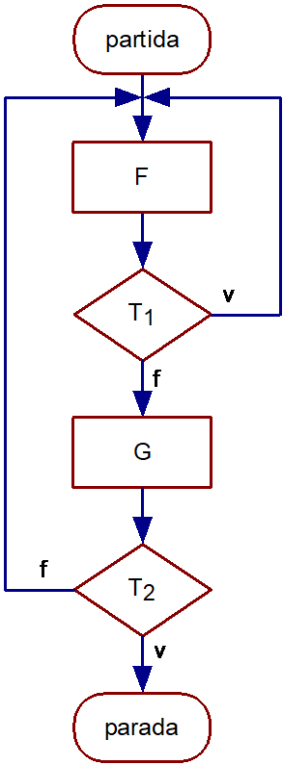
Fluxogramas

- É uma das formas mais comuns de especificar programas monolíticos;
- É um diagrama geométrico construído a partir de componentes elementares denominados



- No caso da operação vazia √, o retângulo correspondente à operação pode ser omitido, resultando simplesmente em uma seta.

Exemplo 2.1 Fluxograma figura 2.2



Instruções rotuladas

- Fluxogramas podem ser reescritos na forma de texto, usando instruções rotuladas. São utilizados rótulos.
- Uma instrução rotulada pode ser:

a) Operação. Indica a operação a ser executada seguida de um desvio incondicional para a instrução subseqüente.

b) *Teste*. Determina um desvio condicional, ou seja, que depende da avaliação de um teste.

c) Uma *parada* é especificada usando um desvio incondicional para um rótulo sem instrução correspondente.

d) Assume-se que a computação sempre inicia no rótulo 1:

```
1: faça F vá para 2
2: se T1 então vá_para 1 senão vá_para 3
3: faça G vá para 4
4: se T2 então vá_para 5 senão vá_para 1
```

figura 2.3 Instruções rotuladas

Formalização de Programa Monolítico

A definição formal de Programa Monolítico é melhor descrita, usando a notação de instruções rotuladas do que diagramas geométricos.

Definição 2.1

Rótulo, Instrução Rotulada.

a) Um *Rótulo* ou *Etiqueta* é uma cadeia de caracteres finita constituída de letras ou dígitos.

b) Uma *Instrução Rotulada* i é uma seqüência de símbolos de uma das duas formas a seguir (suponha que F e T são identificadores de operação e teste, respectivamente e que r_1, r_2 e r_3 são rótulos):

b.1) *Operação* r_1: faça F vá_para r_2

b.2) *Teste* r_1: se T então vá_para r_2 senão vá_para r_3

Exemplo: r_1: faça √ vá_para r_2

Definição 2.2

Programa Monolítico.

Um *Programa Monolítico* P é um par ordenado P = (I, r) onde:
I *Conjunto de Instruções Rotuladas* o qual é finito;
r *Rótulo Inicial* o qual distingue a instrução rotulada inicial em I.

Sabe-se que no conjunto dos Rótulos I

- não existem duas instruções diferentes com um mesmo rótulo;
- um rótulo referenciado por alguma instrução o qual *não* é associado a qualquer instrução rotulada é dito um *Rótulo Final*.

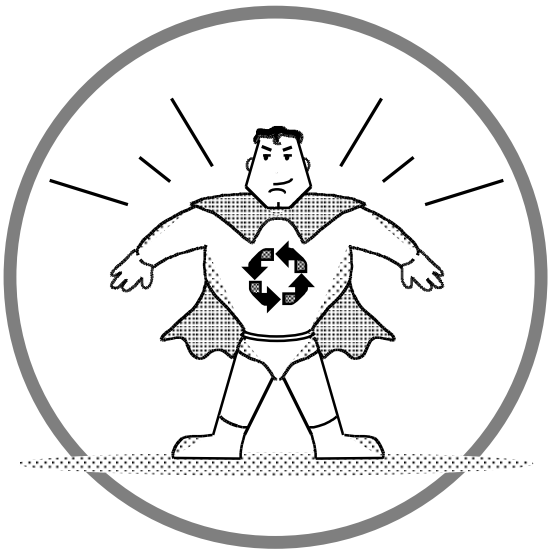
A definição de Programa Monolítico requer a existência de pelo menos uma instrução, identificada pelo rótulo inicial.

Observação 2.3

Programa Monolítico × Fluxograma.

Como um programa monolítico é definido usando a noção de fluxograma, ambos os termos são identificados e usados indistintamente.

2.1.2 Programa Iterativo



Programas iterativos são baseados em três mecanismos de composição (seqüenciais) de programas, os quais podem ser encontrados em um grande número de linguagens de alto nível, como, por exemplo, Algol 68, Pascal, e Fortran 90.

- *seqüencial*: composição de dois programas, resultando em um terceiro, cujo efeito é a execução do primeiro e, após, a execução do segundo programa componente;
- *condicional*: composição de dois programas, resultando em um terceiro, cujo efeito é a execução de somente um dos dois programas componentes, dependendo do resultado de um teste;
- *enquanto*: composição de um programa, resultando em um segundo, cujo efeito é a execução, repetidamente, do programa componente enquanto o resultado de um teste for verdadeiro.
- *até*: análoga à composição enquanto, excetuando que a execução do programa componente ocorre enquanto o resultado de um teste for falso.

Formalização de Programa Iterativo

Definição 2.4
Programa Iterativo

♦ Um *Programa Iterativo* P é indutivamente definido por:

a) A operação vazia √ constitui um programa iterativo;

b) Cada identificador de operação constitui um programa iterativo;

c) *Composição Seqüencial*. V;W

d) *Composição Condicional* (se T então V senão W)

e) *Composição Enquanto*. enquanto T faça (V)

resulta no programa que testa T e executa V, repetidamente, enquanto o resultado do teste for o valor *verdadeiro*.

f) *Composição Até*. até T faça (V)

resulta no programa que testa T e executa V, repetidamente, enquanto o resultado do teste for o valor *falso*.

Os parênteses foram utilizados nas cláusulas das demais composições para possibilitar a interpretação, de forma unívoca, por partes consistentes.

enquanto T faça V;W admite duas interpretações distintas,

- (enquanto T faça V);W
- enquanto T faça (V;W)

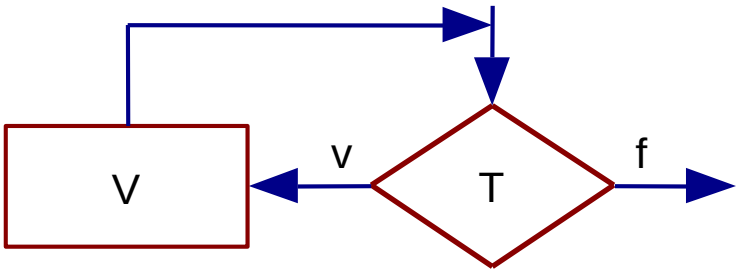


figura 2.4 - Fluxograma que simula a composição enquanto

Exemplo de Programas Iterativos

a) A operação vazia √ constitui um programa iterativo

b) O programa abaixo é do tipo iterativo. O uso de certa forma abusivo de linhas e identação objetiva facilitar a identificação visual da estruturação do programa.

```
(se T1
então   enquanto T2
        faça    (até T3
                faça  (V;W))
senão   (√))
```

figura 2.5 programa iterativo

Os programas tipo while de R. Bird são exemplos de programas iterativos.

A tradução de Programas iterativos para Monolíticos é muito intuitiva, da mesma forma que a tradução de programa **while** para **fluxograma** é intuitiva.

Tem-se que a classe de funções computadas por programas **while** é uma subclasse das funções computadas por **fluxogramas**.

2.1.3 Programa Recursivo

- Recursão é uma forma indutiva de definir programas.
- Sub-rotinas permitem a estruturação hierárquica de programas, possibilitando níveis diferenciados de abstração.

➢ Conjunto de *Identificadores de Sub-Rotinas* - R_1, R_2, ..

Definição 2.5
Expressão de Sub-Rotinas.

Uma *Expressão de Sub-Rotinas* (*Expressão*) E, é definida indutivamente por:

a) A operação vazia √ constitui uma expressão de sub-rotinas.

b) Cada identificador de operação constitui uma expressão de sub-rotinas.

c) Cada identificador de sub-rotina constitui uma expressão de sub-rotinas.

d) A *Composição Seqüencial*. Para D_1 e D_2 expressões de sub-rotinas, a composição seqüencial denotada por: $D_1;D_2$
resulta em uma expressão de subrotina cujo efeito é a execução de D_1 e, após, a execução de D_2.

e) A *Composição Condicional*. Se D_1 e D_2 são expressões de sub-rotinas e T é um identificador de teste, então a composição condicional denotada por: (se T então D_1 senão D_2)
resulta em uma expressão cujo efeito é a execução de D_1 se T é verdadeiro ou D_2 se T é falso. □

Definição 2.6
Programa Recursivo.

Um *Programa Recursivo* P tem a seguinte forma:

$$P \text{ é } E_0 \text{ onde } R_1 \textit{ def } E_1,\ R_2 \textit{ def } E_2,\ \ldots,\ R_n \textit{ def } E_n$$

onde (suponha $k \in \{1, 2, \ldots, n\}$):

- E_0 *Expressão Inicial* a qual é uma expressão de sub-rotinas;
- E_k *Expressão que Define* R_k, a expressão que define a sub-rotina identificada por R_k.
- A operação vazia √ constitui um programa recursivo que não faz coisa alguma.

Exemplo 2.6
Programa Recursivo.

P *é* R;Z *onde*
R *def* F;(se T então R senão G;S),
S *def* (se T então √ senão F;R)
Z *def* (se T então F;√ senão G;√)

figura 2.6 - Programa recursivo

- A computação de um programa recursivo consiste na avaliação da expressão inicial onde cada identificador de sub-rotina referenciado é substituído pela correspondente expressão que o define, e assim sucessivamente, até que seja substituído pela expressão vazia √, determinando o fim da recursão.

- Até agora, foram definidos três tipos de programas. Entretanto, esses programas são incapazes de descrever uma computação, pois não se tem a natureza das operações ou dos testes, mas apenas um conjunto de identificadores. A natureza das operações e testes é especificada na definição de máquina.

2.2 Máquinas

A máquina deve suprir todas as informações necessárias para que a computação de um programa possa ser descrita:

- cada identificador de operação deve caracterizar uma transformação na estrutura da memória da máquina;
- cada identificador de teste interpretado pela máquina deve ser associado a uma função verdade;
- nem todo identificador de operação ou teste é definido em uma máquina;
- para cada identificador de operação ou teste definido em uma máquina, existe somente uma função associada;
- deve descrever o armazenamento ou a recuperação de informações na estrutura de memória.

Definição 2.7
Máquina.

Uma *Máquina* é uma 7-upla

$$M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$$

V *Conjunto de Valores de Memória*;

X *Conjunto de Valores de Entrada*;

Y *Conjunto de Valores de Saída*;

π_X *Função de Entrada* tal que: $\pi_X: X \rightarrow V$

π_Y *Função de Saída* tal que: $\pi_Y : V \rightarrow Y$

Π_F *Conjunto de Interpretações de Operações* tal que, para cada identificador de operação F interpretado por M, existe uma única função: $\pi_F: V \rightarrow V$ em Π_F

Π_T *Conjunto de Interpretações de Testes* tal que, para cada identificador de teste T interpretado por M, existe uma única função: $\pi_T: V \rightarrow \{verdadeiro, falso\}$ em Π_T

Exemplo 2.7
Máquina de Dois Registradores.

Suponha uma especificação de uma máquina com dois registradores a e b os quais assumem valores em N, com duas operações e um teste:

- subtração de 1 em a, se a > 0;
- adição de 1 em b;
- teste se a é zero.
- Os valores de entrada são armazenados em a (zerando b) e a saída retorna o valor de b.

dois_reg = (N^2, N, N, armazena_a, retorna_b, {subtrai_a, adiciona_b}, {a_zero})

N^2, N, N - Conjuntos de Memória, Entrada e Saída

armazena_a: $N \rightarrow N^2$ tal que, $\forall n \in N$, armazena_a(n) = (n, 0)

retorna_b: $N^2 \rightarrow N$ tal que, $\forall (n, m) \in N^2$, retorna_b(n, m) = m

subtrai_a: $N^2 \rightarrow N^2$ tal que, $\forall (n, m) \in N^2$,

subtrai_a(n, m) = (n-1, m), se $n \neq 0$; subtrai_a(n, m) = (0, m), se n = 0

adiciona_b: $N^2 \rightarrow N^2$ tal que, $\forall (n,m) \in N^2$, adiciona_b(n,m) =(n, m+1)

a_zero: $N^2 \rightarrow$ {verdadeiro, falso} tal que, $\forall (n, m) \in N^2$,

a_zero(n, m) = verdadeiro, se n = 0; a_zero(n, m) = falso, se $n \neq 0$.

figura 2.7 máquina com dois registradores

- Afirma-se que P é um programa para máquina M se cada identificador de teste e operação em P tiver uma correspondente função de teste e operação em M, respectivamente.

Definição 2.8
Programa para uma Máquina.

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina

P um programa onde P_F e P_T são os conjuntos de identificadores de operações e de testes de P, respectivamente.

P é um Programa para a Máquina M se, e somente, se:

- para qualquer $F \in P_F$, existe uma única função $\pi_F: V \rightarrow V$ em Π_F;
- para qualquer $T \in P_T$, existe uma única função $\pi_T: V \rightarrow$ {verdadeiro, falso} em Π_T.

Exemplo 2.8 Programas para a Máquina de Dois Registradores.

```
Programa Iterativo itv_b←a
    até  a_zero
    faça (subtrai_a; adiciona_b)
```

figura 2.8 Programa iterativo para a máquina de dois registradores

```
Programa Recursivo rec_b←a
rec_b←a é R onde
        R def    (se a_zero então √ senão S;R),
        S def    subtrai_a; adiciona_b
```

figura 2.9 Programa recursivo para a máquina de dois registradores

2.3 Computações e Funções Computadas

- Será visto como as definições de programas e máquinas caminham juntas para a definição de computação.
- Uma *computação* é um histórico do funcionamento da máquina para o programa, considerando um valor inicial.
- Uma vez definida a noção de computação, pode-se inferir a natureza da função computada, por um dado programa, em uma dada máquina.

2.3.1 Computação

Uma computação de um programa monolítico em uma máquina é um histórico das instruções executadas e o correspondente valor de memória.

O histórico é representado na forma de uma cadeia de pares onde:

- cada par reflete um estado da máquina para o programa, ou seja, a instrução a ser executada e o valor corrente da memória;
- a cadeia reflete uma seqüência de estados possíveis a partir do estado inicial, ou seja, instrução (rótulo) inicial e valor de memória considerado.

Definição 2.9
Computação de Programa Monolítico em uma Máquina.

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina

$P = (I, r)$ um programa monolítico para M

onde L é o seu correspondente conjunto de rótulos.

Uma *Computação do Programa Monolítico* P *na Máquina* M é uma cadeia (finita ou infinita) de pares de $L \times V$:

$$(s_0, v_0)(s_1, v_1)(s_2, v_2)...$$

onde

- (s_0, v_0) é tal que $s_0 = r$ é o rótulo inicial do programa P e v_0 é o valor inicial de memória
- para cada par (s_k, v_k) da cadeia, onde $k \in \{0, 1, 2, ...\}$, tem-se que (suponha que F é um identificador de operação, T é um identificador de teste e r', r" são rótulos de L):

  a) ***Operação***. Se s_k é o rótulo de uma operação da forma:

  s_k: faça F vá_para r'

  então $(s_{k+1}, v_{k+1}) = (r', \pi_F(v_k))$

  s_k: faça √ vá_para r'

  então $(s_{k+1}, v_{k+1}) = (r', v_k)$

  b) ***Teste***. Se s_k é o rótulo de um teste da forma:

  s_k: se T então vá_para r' senão vá_para r"

  então $(s_{k+1}, v_{k+1}) = (? , v_k)$

  $s_{k+1} = r'$ se $\pi_T(v_k)$ = verdadeiro;

  $s_{k+1} = r''$ se $\pi_T(v_k)$ = falso.

- Uma *Computação* é dita *Finita* se a cadeia que a define é finita e é dita *Infinita* se a cadeia que a define é infinita.
- para um dado valor inicial de memória, a correspondente cadeia de computação é única, ou seja, a computação é determinística;
- um teste não altera o valor corrente da memória;
- em uma computação infinita, rótulo algum da cadeia é final.

Exemplo 2.9 Computação Finita de Programa Monolítico na Máquina de Dois Registradores.

- programa monolítico mon_b←a para a máquina dois_reg

Programa Monolítico mon_b←a

```
1: se a_zero então vá_para 9 senão vá_para 2
2: faça subtrai_a vá_para 3
3: faça adiciona_b vá_para 1
```

figura 2.10 Programa monolítico de computação finita

- Para o valor inicial de memória (3, 0), a computação finita é:

(1, (3, 0))	instrução inicial e valor de entrada armazenado
(2, (3, 0))	em 1, como a ≠ 0, desviou para 2
(3, (2, 0))	em 2, subtraiu do registrador a e desviou para 3
(1, (2, 1))	em 3, adicionou no registrador b e desviou para 1
(2, (2, 1))	em 1, como a ≠ 0, desviou para 2
(3, (1, 1))	em 2, subtraiu do registrador a e desviou para 3
(1, (1, 2))	em 3, adicionou no registrador b e desviou para 1
(2, (1, 2))	em 1, como a ≠ 0, desviou para 2
(3, (0, 2))	em 2, subtraiu do registrador a e desviou para 3
(1, (0, 3))	em 3, adicionou no registrador b e desviou para 1
(9, (0, 3))	em 1, como a = 0, desviou para 9

figura 2.11 computação finita

Exemplo 2.10 Computação Infinita de Programa Monolítico na Máquina de Dois Registradores.

- programa monolítico comp_infinita para a máquina dois_reg.

Programa Monolítico comp_infinita

```
1:  faça adiciona_b vá_para 1
```

figura 2.12 Programa monolítico de computação infinita

- Para o valor inicial de memória (3, 0), a computação infinita é

(1, (3, 0))	instrução inicial e valor de entrada armazenado
(1, (3, 1))	adicionou no registrador b e permanece em 1
(1, (3, 2))	adicionou no registrador b e permanece em 1
(1, (3, 3))	adicionou no registrador b e permanece em 1
...	repete 1, adicionando no registrador b, indefinidamente

figura 2.13 computação infinita

A computação de um programa recursivo em uma máquina é análoga à de um monolítico.
O histórico é representado na forma de uma cadeia de pares onde:

- cada par reflete um estado da máquina para o programa, ou seja, a expressão de sub-rotina a ser executada e o valor corrente da memória;
- a cadeia reflete uma seqüência de estados possíveis a partir do estado inicial.

A *Computação* é dita *Finita* ou *Infinita*, se a cadeia que a define é finita ou infinita, respectivamente.

- para um dado valor inicial de memória, a correspondente cadeia de computação é única, ou seja, a computação é determinística;
- um teste ou uma referência a uma sub-rotina não alteram o valor corrente da memória;
- em uma computação finita, a expressão √ ocorre no último par da cadeia e não ocorre em qualquer outro par;
- em uma computação infinita, expressão alguma da cadeia é √.

Definição 2.10
Computação de Programa Recursivo em uma Máquina.

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina

P um programa recursivo para M tal que:

P *é* E_0 *onde* R_1 *def* E_1, R_2 *def* E_2,...,R_n *def* E_n

Uma *Computação do Programa Recursivo* P *na Máquina* M é uma cadeia de pares da forma:

$(D_0, v_0)\ (D_1, v_1)\ (D_2, v_2)...$

onde

- (D_0, v_0) é tal que $D_0 = \mathbf{E_0;\sqrt{}}$ e v_0 é o valor inicial de memória;
- para cada par (D_k, v_k) da cadeia, onde $k \in \{0, 1, 2, ...\}$, tem-se que (suponha que F é um identificador de operação, T é um identificador de teste e C, C_1, C_2 são expressões de sub-rotina):

Caso 1. Se D_k é uma expressão de sub-rotina da forma:

$D_k = (\sqrt{};C)$

então $(D_{k+1}, v_{k+1}) = (C, v_k)$

Caso 2. Se D_k é uma expressão de sub-rotina da forma:

$D_k = F;C$

então $(D_{k+1}, v_{k+1}) = (C, \pi_F(v_k))$

Caso 3. Se D_k é uma expressão de sub-rotina da forma:

$D_k = R_i;C$

então $(D_{k+1}, v_{k+1}) = (E_i;C, v_k)$

Caso 4. Se D_k é uma expressão de sub-rotina da forma:

$D_k = (C_1;C_2);C$

então $(D_{k+1}, v_{k+1}) = (C_1;(C_2;C), v_k)$

Caso 5. Se D_k é uma expressão de sub-rotina da forma:

$D_k = E_k = (\text{se T então } C_1 \text{ senão } C_2);C$

então $(D_{k+1}, v_{k+1}) = (\ ?\ , v_k)$

$D_{k+1} = C_1;C$ se $\pi_T(v_k) =$ verdadeiro

$D_{k+1} = C_2;C$ se $\pi_T(v_k) =$ falso

Exemplo 2.11
Computação Infinita de Programa Recursivo.

- programa recursivo qq_máquina é um programa para qualquer máquina

Programa Recursivo qq_máquina

```
qq_máquina é R onde
                R def R
```

figura 2.14 Programa Recursivo cuja computação é infinita

- Para qualquer valor inicial de memória, a correspondente computação é sempre infinita e é a cadeia:

$$(R;\sqrt{}, v_0)\ (R;\sqrt{}, v_0)\ (R;\sqrt{}, v_0)\ldots$$

Exemplo 2.12
Computação finita de Programa Recursivo na máquina de um registrador.

- Para um dado conjunto A, a *Função Identidade em* A é aquela que associa, a cada elemento do conjunto, o próprio elemento, ou seja: $id_A: A \to A$, tal que, para qualquer $a \in A$, tem-se que $id_A(a) = a$.
- Considera a Máquina de um Registrador um_reg

$$um\_reg = (N, N, N, id_N, id_N, \{ad, sub\}, \{zero\})$$

N, N, N – Conjuntos de Memória, Entrada e Saída

$id_N: N \to N$ é a função identidade em N

ad: $N \to N$ tal que, $\forall n \in N$, $ad(n) = n+1$

sub: $N \to N$ tal que, $\forall n \in N$, $sub(n) = n-1$

$sub(n) = n-1$, se $n \neq 0$; $sub(n) = 0$, se $n = 0$

zero: $N \to \{verdadeiro, falso\}$ tal que, $\forall n \in N$,

$zero(n) = verdadeiro$, se $n = 0$; $zero(n) = falso$, caso contrário.

figura 2.15 máquina um-reg

- Considere o programa recursivo duplica para a máquina um_reg .

Programa Recursivo duplica

```
duplica é R onde
        R def (se zero então √ senão sub;R;ad;ad)
```

figura 2.16 programa recursivo cuja computação é finita

Para o valor inicial de memória 3, a computação finita é

(R;√, 3)	valor de entrada armazenado
((se zero então √ senão (sub;R;ad;ad));√, 3)	caso 3
((sub;R;ad;ad);√, 3)	como n ≠ 0, executa senão
(sub;(R;ad;ad);√, 3)	caso 4, composição seqüencial
((R;ad;ad);√, 2)	subtraiu 1 da memória
(R;((ad;ad);√), 2)	caso 4, composição seqüencial
((se zero então √ senão (sub;R;ad;ad));((ad;ad);√), 2)	caso 3
((sub;R;ad;ad);(ad;ad);√, 2)	como n ≠ 0, executa senão
(sub;(R;ad;ad);(ad;ad);√, 2)	caso 4, composição seqüencial
((R;ad;ad);(ad;ad);√, 1)	subtraiu 1 da memória
(R;(ad;ad);(ad;ad);√, 1)	caso 4, composição seqüencial
((se zero então √ senão(sub;R;ad;ad));(ad;ad);(ad;ad);√, 1)	caso 3
((sub;R;ad;ad));(ad;ad);(ad;ad);√, 1)	como n ≠ 0, executa senão
(sub;(R;ad;ad);(ad;ad);(ad;ad);√, 1)	caso 4, composição seqüencial
((R;ad;ad);(ad;ad);(ad;ad);√, 0)	subtraiu 1 da memória
(R;(ad;ad);(ad;ad);(ad;ad);√, 0)	caso 4, composição seqüencial
((se zero então √ senão (sub;R;ad;ad));(ad;ad);(ad;ad);(ad;ad);√, 0)	caso 3
(√;(ad;ad);(ad;ad);(ad;ad);√, 0)	como n = 0, executa então
((ad;ad);(ad;ad);(ad;ad);√, 0)	caso 1
(ad;(ad;(ad;ad);(ad;ad);√), 0)	caso 4, composição seqüencial
((ad;(ad;ad);(ad;ad);√), 1)	adicionou 1 na memória
(ad;((ad;ad);(ad;ad);√), 1)	caso 4, composição seqüencial
(((ad;ad);(ad;ad);√), 2)	adicionou 1 na memória
(ad;(ad;(ad;ad);√), 2)	caso 4, composição seqüencial
((ad;(ad;ad);√), 3)	adicionou 1 na memória
(ad;((ad;ad);√), 3)	caso 4, composição seqüencial
(((ad;ad);√), 4)	adicionou 1 na memória
(ad;((ad);√), 4)	caso 4, composição seqüencial
(((ad);√), 5)	adicionou 1 na memória
(ad;(√), 5)	caso 4, composição seqüencial
((√), 6)	adicionou 1 na memória
(√, 6)	fim da recursão

Figura 2.17 Computação finita na máquina de um-reg do programa duplica

- Note-se que o valor final na memória é o valor inicial duplicado.

2.3.2 Função Computada

A computação de um programa deve ser associada a uma entrada e a uma saída. Adicionalmente, espera-se que a resposta (saída) seja gerada em um tempo finito. Essas noções induzem a definição de *função computada*.

A ***função computada por um programa monolítico*** sobre uma máquina:

- a computação inicia na instrução identificada pelo rótulo inicial, com a memória contendo o valor inicial resultante da aplicação da função de entrada sobre o dado fornecido;
- executa, passo a passo, testes e operações, na ordem determinada pelo programa, até atingir um rótulo final, quando pára;
- o valor da função computada pelo programa é o valor resultante da aplicação da função de saída ao valor da memória quando da parada.

Definição 2.11
Função Computada por um Programa Monolítico em uma Máquina.

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina

P um programa monolítico para M.

A **função computada pelo programa monolítico P na máquina M** denotada por:

$$\langle P, M\rangle: X \rightarrow Y$$

é uma função parcial definida para $x \in X$ se a cadeia:

$$(s_0, v_0)(s_1, v_1)...(s_n, v_n)$$

é uma computação finita de P em M, onde o valor inicial da memória é dado pela função de entrada, ou seja, $v_0 = \pi_X(x)$. Neste caso, a imagem de x é dada pela função de saída aplicada ao último valor da memória na computação, ou seja:

$$\langle P, M\rangle(x) = \pi_Y(v_n)$$

Exemplo 2.13
Função computada por programa monolítico na máquina de dois registradores.

- Considere o programa monolítico mon_b←a para a máquina dois_reg .
- A correspondente função computada é a função identidade,

$$\langle \text{mon\_b} \leftarrow \text{a, dois\_reg} \rangle : \mathbb{N} \rightarrow \mathbb{N}$$

tal que, para qualquer $n \in \mathbb{N}$, tem-se que:

$$\langle \text{mon\_b} \leftarrow \text{a, dois\_reg} \rangle (n) = n$$

- Por exemplo, para o valor entrada de 3, tem-se que:
  - $\pi X(3) = (3, 0)$;
  - $\langle \text{mon\_b} \leftarrow \text{a, dois\_reg} \rangle (3) = \pi Y(0, 3) = 3$.
  - Portanto, $\langle \text{mon\_b} \leftarrow \text{a, dois\_reg} \rangle$ é definida para 3.

Exemplo 2.14
Função computada por programa monolítico na máquina de dois registradores.

- Considere o programa monolítico comp_infinita para a máquina dois_reg.
- A função computada $\langle \text{comp\_infinita, dois\_reg} \rangle : \mathbb{N} \rightarrow \mathbb{N}$, para a entrada de valor 3:
  - $\pi_X(3) = (3, 0)$;
  - Como a cadeia é infinita, a função computada *não* é definida para o valor de entrada 3.

A **função computada por um programa recursivo R sobre uma máquina M** é análoga à de um monolítico:

- a computação inicia na expressão inicial concatenada com a operação vazia e com a memória contendo o valor inicial resultante da aplicação da função de entrada sobre o dado fornecido;
- executa, passo a passo, testes e operações, na ordem determinada pelo programa, até que a expressão de sub-rotina resultante seja a expressão vazia, quando pára;
- o valor da função computada pelo programa é o valor resultante da aplicação da função de saída ao valor da memória quando da parada.

Definição 2.12

Função computada por um programa recursivo em uma máquina.

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina

P um programa recursivo para M.

A *Função Computada pelo Programa Recursivo* P *na Máquina* M denotada por:

$$\langle P, M\rangle: X \to Y$$

é uma função parcial definida para $x \in X$ se a seguinte cadeia é uma computação finita de P em M: $(D_0, v_0)\ (D_1, v_1)...(D_n, v_n)$

onde:

- $D_0 = E_0;\sqrt{}$ é expressão inicial de P
- $v_0 = \pi_X(x)$
- $D_n = E_n = \sqrt{}$

Neste caso, tem-se que: $\langle P, M\rangle(x) = \pi_Y(v_n)$

Exemplo 2.15 função computada por programa recursivo.

- Considere o programa recursivo qq_máquina e uma máquina $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ qualquer.
- A correspondente função computada é: $\langle$qq_máquina, M$\rangle: X \to Y$ e é indefinida para qualquer entrada.

Exemplo 2.16

Função computada por programa recursivo na máquina de um registrador.

- Considere o programa recursivo duplica para a máquina um_reg.
- A correspondente função computada é: $\langle$duplica, um_reg$\rangle: N \to N$ e é tal que, para qualquer $n \in N$:

$$\langle \text{duplica, um\_reg}\rangle(n) = 2n.$$

Exercicios 2.7 (solução):

Dê a definição formal de computação $\langle P, M \rangle$ *de um programa iterativo em uma máquina M e defina função computada.*

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina e P um programa iterativo para M.

A **computação do programa iterativo P na Máquina M** é uma cadeia de pares da forma: $(X_0, v_0)\ (X_1, v_1)\ (X_2, v_2)\ldots$ onde

- (X_0, v_0) é tal que $\mathbf{X_0;\sqrt{}}$ é todo o programa iterativo inicial P concatenado ao programa vazio e v_0 é o valor inicial armazenado na memória;
- para cada par $\mathbf{(X_j, v_j)}$ da cadeia, onde $j \in \{0, 1, 2, ...\}$, tem-se que X_j são programas Iterativos (suponha que F é um identificador de operação, T é um identificador de teste e $\mathbf{W}$, $\mathbf{W_1}$, $\mathbf{W_2}$ são programas iterativos):

a) Se X_j é da forma: $\mathbf{X_j = \sqrt{};\ W}$

então, tem-se que:

$X_{j+1} = \mathbf{W}$ e

$\mathbf{v_{j+1} = v_j}$

b) Se X_j é da forma: $\mathbf{X_j = F;\ W}$

então, tem-se que:

$X_{j+1} = \mathbf{W}$ e

$\mathbf{v_{j+1} = \pi_F\ (v_j)}$

c) Se X_j é da forma $\mathbf{X_j = (W_1;\ W_2);\ W}$

então, tem-se que:

$X_{j+1} = \mathbf{W_1;(W_2;\ W)}$ **e**

$\mathbf{v_{j+1} = v_j}$

d) Se X_j é da forma $\mathbf{X_j = (}$se **T** então $\mathbf{W_1}$ senão $\mathbf{W_2)}$**; W**

então, tem-se que:

$X_{j+1} = \mathbf{W_1;\ W}$ se $\pi_T\ (v_j)$ = verdadeiro

$\mathbf{W_2;\ W}$ se $\pi_T\ (v_j)$ = falso

e $\mathbf{v_{j+1} = v_j}$

e) Se X_j é da forma X_j = enquanto **T** faça **(W_1); W**

então, tem-se que:

X_{j+1} = **W_1**; enquanto **T** faça **(W_1);W** se $\pi_T(v_j)$= verdadeiro

W se $\pi_T(v_j)$ = falso

e $v_{j+1} = v_j$

f) Se X_j é da forma X_j = até **T** faça **(W_1); W**

então, tem-se que:

X_{j+1} = **W_1**; até **T** faça **(W_1);W** se $\pi_T(v_j)$= *falso*

W se $\pi_T(v_j)$ = *verdadeiro*

e $v_{j+1} = v_j$

Exercicio 2.8 - Função computada por um programa iterativo em uma máquina. (solução):

Sejam $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina e P um programa iterativo para M.

A ***função computada pelo programa iterativo P na máquina M*** denotada por:

$$\langle P, M\rangle: X \to Y$$

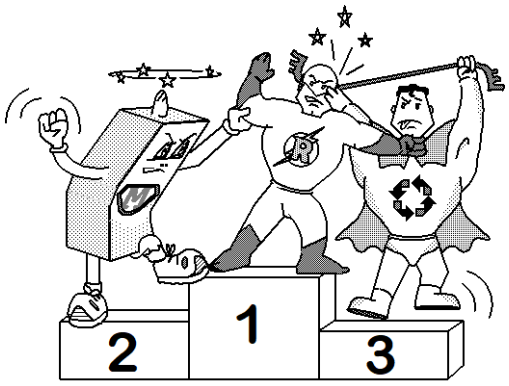
é uma função parcial definida para $x \in X$ se a cadeia:

$$(x_0, v_0)(x_1, v_1)...(x_n, v_n)$$

é uma computação finita de P em M, onde $X_n = \sqrt{}$, e o valor inicial da memória é dado pela função de entrada, ou seja, $v_0 = \pi_X(x)$. Nesse caso, a imagem de x é dada pela função de saída aplicada ao último valor da memória na computação, ou seja:

$$\langle P, M\rangle(x) = \pi_Y(v_n)$$

2.4 Equivalências de Programas e Máquinas



Relação equivalência forte de programas. Um par de programas pertence à relação se as correspondentes funções computadas coincidem para *qualquer* máquina;

Relação equivalência de programas em uma máquina. Um par de programas pertence à relação se as correspondentes funções computadas coincidem para uma *dada* máquina;

Relação equivalência de máquinas. Um par de máquina pertence à relação se as máquinas podem se simular mutuamente. A simulação de uma máquina por outra pode ser feita usando programas diferentes.

- A Relação Equivalência Forte de Programas é especialmente importante pois, ao agrupar diferentes programas em classes de equivalências de programas cujas funções coincidem, fornece subsídios para analisar propriedades de programas como complexidade estrutural.
- Um importante resultado é que programas recursivos são mais gerais que os monolíticos, os quais, por sua vez, são mais gerais que os iterativos.

Igualdade de Funções Parciais. Duas funções parciais $f, g: X \rightarrow Y$ são ditas iguais, ou seja, $f = g$, se, e somente se, para cada $x \in X$: ou $f(x)$ e $g(x)$ são indefinidas; ou ambas são definidas e $f(x) = g(x)$.

Composição Sucessiva de Funções. Para uma dada função $f: S \rightarrow S$, a composição sucessiva de f com ela própria é denotada usando expoente, $f^n = f \circ f... \circ f$ (n vezes)

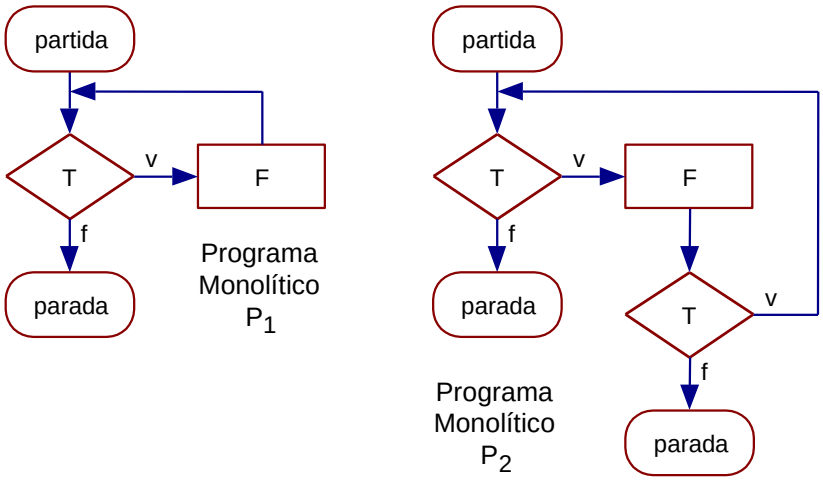
2.4.1 Equivalência Forte de Programas

Definição 2.13
Relação equivalência forte de programas, programas fortemente equivalentes.

- Sejam P e Q dois programas arbitrários, não necessariamente do mesmo tipo.
- Então o par (P, Q) está na *Relação Equivalência Forte de Programas*, denotada por: $P \equiv Q$ se, e somente se, para qualquer máquina M, as correspondentes funções parciais computadas são iguais, $\langle P, M\rangle = \langle Q, M\rangle$.
- *Neste caso, P e Q são ditos Programas Fortemente Equivalentes.*

Exemplo 2.17 Programas fortemente equivalentes.

- Considere os quatro programas na figura abaixo.
- Os programas monolíticos P_1 e P_2 , o iterativo P_3 e o recursivo P_4 são todos fortemente equivalentes.



Programa Iterativo P_3

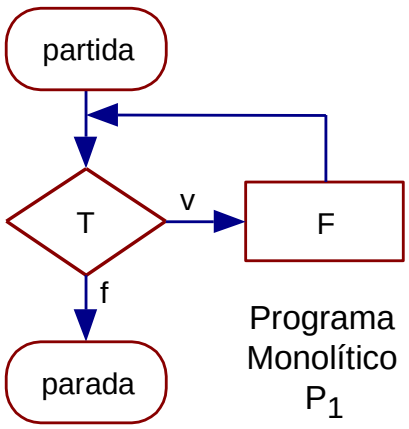
```
enquanto T
faça  (F)
```

Programa Recursivo P_4

```
P4 é R onde
      R def (se T então F;R senão √)
```

Figura 2.18 Programas Fortemente Equivalentes

Programa monolítico



Programa Monolítico P_1

```
1:seTentão vá_para 2 senão vá_para3
2:faça F vá_para 1
```

conjunto de instruções rotuladas

Sejam

$M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$ uma máquina arbitrária

$x \in X$ tal que $\pi_X(x) = v$.

Se $\langle P_1, M\rangle$ é definida para x

computação é dada pela cadeia:

$$(1, v)\ (2, v)\ (1, \pi_F(v))\ (2, \pi_F(v))\ (1, \pi_F^2(v))\ (2, \pi_F^2(v))$$
$$\ldots(1, \pi_F^n(v))\ \ (3, \pi_F^n(v))$$

supondo que n é o menor natural tal que $\pi_T(\pi_F^n(v)) = falso$.

Neste caso: $\langle P_1, M\rangle(x) = \pi_Y(\pi_F^n(v))$

Programa Recursivo P_4

```
P4 é R onde
    R def (se T então F;R senão √)
```

Se $\langle P_4, M\rangle$ é definida para x, Computação é dada por:

$(R;\surd, v)$
$((\text{se } T \text{ então } F;R \text{ senão} \surd);\surd, v)$
$(F;R;\surd, v)$

$(R;\surd, \pi_F(v))$
$((\text{se } T \text{ então } F;R \text{ senão } \surd);\surd, \pi_F(v))$
$(F;R;\surd, \pi_F(v))$

$(R;\surd, \pi_F^2(v))$
$((\text{se } T \text{ então } F;R \text{ senão } \surd);\surd, \pi_F^2(v))$
$(F;R;\surd, \pi_F^2(v))$

...
$(R;\surd, \pi_F^n(v))$
$((\text{se } T \text{ então } F;R \text{ senão } \surd);\surd, \pi_F^n(v))$
$(\surd;\surd, \pi_F^n(v))$
$(\surd, \pi_F^n(v))$

figura 2.20 computação

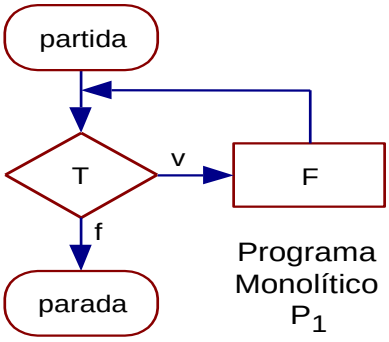
supondo que n é o menor natural tal que $\pi_T(\pi_F^n(v)) = \text{falso}$.

Neste caso: $\langle P_4, M\rangle(x) = \pi_Y(\pi_F^n(v))$

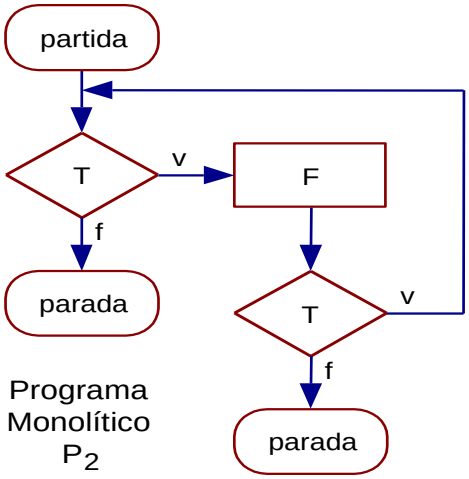
Portanto, $P_1 \equiv P_4$ pois, para qualquer máquina M, tem-se que: $\langle P_1, M\rangle = \langle P_4, M\rangle$

Relação Equivalência Forte de Programas

- permite identificar diferentes programas em uma mesma classe de equivalência, ou seja, identificar diferentes programas cujas funções computadas coincidem, para qualquer máquina;
- as funções computadas por programas fortemente equivalentes têm a propriedade de que as mesmas operações são efetuadas na mesma ordem, independentemente do significado das mesmas;
- fornece subsídios para analisar a complexidade estrutural de programas. Por exemplo, analisando os programas monolíticos equivalentes fortemente P_1 e P_2 pode-se concluir que P_1 é estruturalmente "mais otimizado" que P_2, pois contém um teste a menos.



Programa Monolítico P_1



Programa Monolítico P_2

- **para todo programa iterativo, existe um programa monolítico fortemente equivalente;**

- **para todo programa monolítico, existe um programa recursivo fortemente equivalente.**

- **Entretanto, a inversa não necessariamente é verdadeira,** ou seja, relativamente à Relação Equivalência Forte de Programas, programas recursivos são mais gerais que os monolíticos, os quais, por sua vez, são mais gerais que os iterativos, induzindo uma hierarquia de classes de programas

figura 2.21 Hierarquia induzida pela relação equivalência forte de programas

Teorema 2.14
Equivalência forte de programas: iterativo → monolítico.

Para qualquer programa iterativo P_i, existe um programa monolítico P_m, tal que $P_i \equiv P_m$.

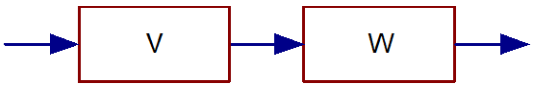
PROVA: Seja P_i um programa iterativo qualquer. Seja P_m um programa monolítico indutivamente construído como segue:

a) Para a operação vazia √ corresponde o fluxograma elementar:

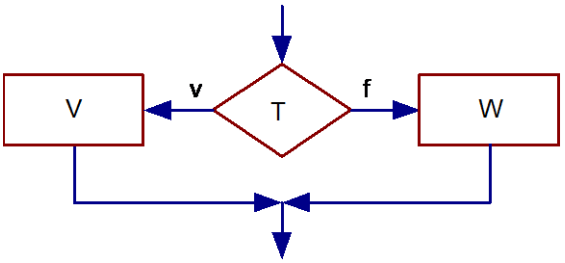
b) Para cada identificador de operação F de P_i corresponde o seguinte fluxograma elementar:

F	

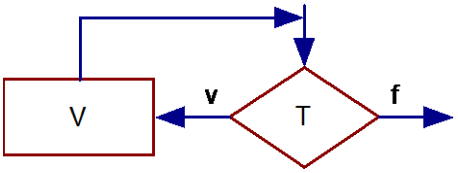
c) Composição Seqüencial.

V;W	

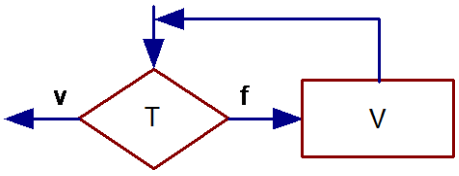
d) Composição Condicional.

(se T então V senão W)	

e) Composição Enquanto.

enquanto T faça (V)	

f) Composição Até.

até T faça (V)	

Teorema 2.15
Equivalência forte de programas: monolítico → recursivo.
Para qualquer programa monolítico P_m, existe um programa recursivo P_r, tal que $P_m \equiv P_r$.

PROVA: Seja P_m um programa monolítico qualquer onde $L = \{r_1, r_2, ..., r_n\}$ é o correspondente conjunto de rótulos. Suponha que, em P_m, r_n é o *único* rótulo final. Então P_r é um programa recursivo construído a partir de P_m e é tal que:

P_r é R_1 onde R_1 def E_1, R_2 def E_2, ...,R_n def √

para $k \in \{1, 2, ..., n-1\}$, E_k é definido como segue:

a) *Operação*. Se r_k é da forma: `rk: faça F vá_para rk'`
então E_k é a seguinte expressão de sub-rotinas: `F;Rk'`

b) *Teste*. Se r_k é da forma:
`rk: se T então vá_para rk' senão vá_para rk"`
então E_k é a seguinte expressão de sub-rotinas:
`(se T então Rk' senão Rk")`

Corolário 2.16
Equivalência forte de programas: iterativo → recursivo.
Para qualquer programa iterativo P_i, existe um programa recursivo P_r, tal que $P_i \equiv P_r$.

Teorema 217
Equivalência forte de programas:recursivo ↛ monolítico.
Dado um programa recursivo P_r qualquer, não necessariamente existe um programa monolítico P_m, tal que $P_r \equiv P_m$.

PROVA: (Por Absurdo).

- Para provar, é suficiente apresentar um programa recursivo que, para uma determinada máquina, não apresente programa monolítico fortemente equivalente.

  - Considere o programa recursivo duplica e a máquina um_reg.
  - A função computada $\langle$duplica, um_reg$\rangle$: N $\rightarrow$ N, para qualquer n $\in$ N:

    $\langle$duplica, um_reg$\rangle$(n) = 2n

  - Suponha que existe um programa monolítico P_m que computa a mesma função, ou seja, que $\langle P_m$, um_reg$\rangle$: N $\rightarrow$ N e:

    $\langle$duplica, um_reg$\rangle$ = $\langle P_m$, um_reg$\rangle$
  - Suponha que P_m é constituído de k operações ad.
  - Suponha n $\in$ N tal que n $\geq$ k.

Então,

  - para que $\langle P_m$, um_reg$\rangle$ (n) = 2n, é necessário que P_m execute n vezes a operação ad.

Mas, como n $\geq$ k,

  - então pelo menos uma das ocorrências de ad será executada mais de uma vez, ou seja, existe um ciclo em P_m.

- Na função computada por dois programas equivalentes fortemente, as mesmas operações são efetuados na mesma ordem;
- Com um registrador, não é possível fazer o controle do ciclo e, ainda, utilizá-lo como acumulador, pois como controlador é subtraída uma unidade e como acumulador são somadas duas unidades, resultando em um ciclo executado indefinidamente.
- Portanto, a computação resultante é infinita e a correspondente função não é definida para n, o que é **um absurdo**, pois é suposto que os dois programas são equivalentes fortemente.
- Logo, não existe um programa monolítico fortemente equivalente ao programa recursivo duplica.

Comentários importantes:

Para melhor entender o esse resultado:

- um programa de qualquer tipo não pode ser modificado dinamicamente, durante uma computação;

- um programa, para ser fortemente equivalente a outro, não pode conter ou usar facilidades adicionais como memória auxiliar ou operações extras;

- para que um programa monolítico possa simular uma recursão sem um número finito e predefinido de quantas vezes a recursão pode ocorrer, seriam necessárias infinitas opções de ocorrências das diversas operações ou testes envolvidos na recursão em questão;

- infinitas opções implicam um programa infinito, o que contradiz a definição de programa monolítico, o qual é constituído por um conjunto *finito* de instruções rotuladas.

Teorema 2.18
Equivalência forte de programas: monolítico ↛ iterativo.

Dado um programa monolítico P_m qualquer, não necessariamente existe um programa iterativo P_i, tal que $P_m \equiv P_i$.

PROVA: (Por Absurdo).

- Para provar, é suficiente apresentar um programa monolítico que, para uma determinada máquina, não apresente programa iterativo fortemente equivalente.

- Considere o programa monolítico par e a máquina um_reg

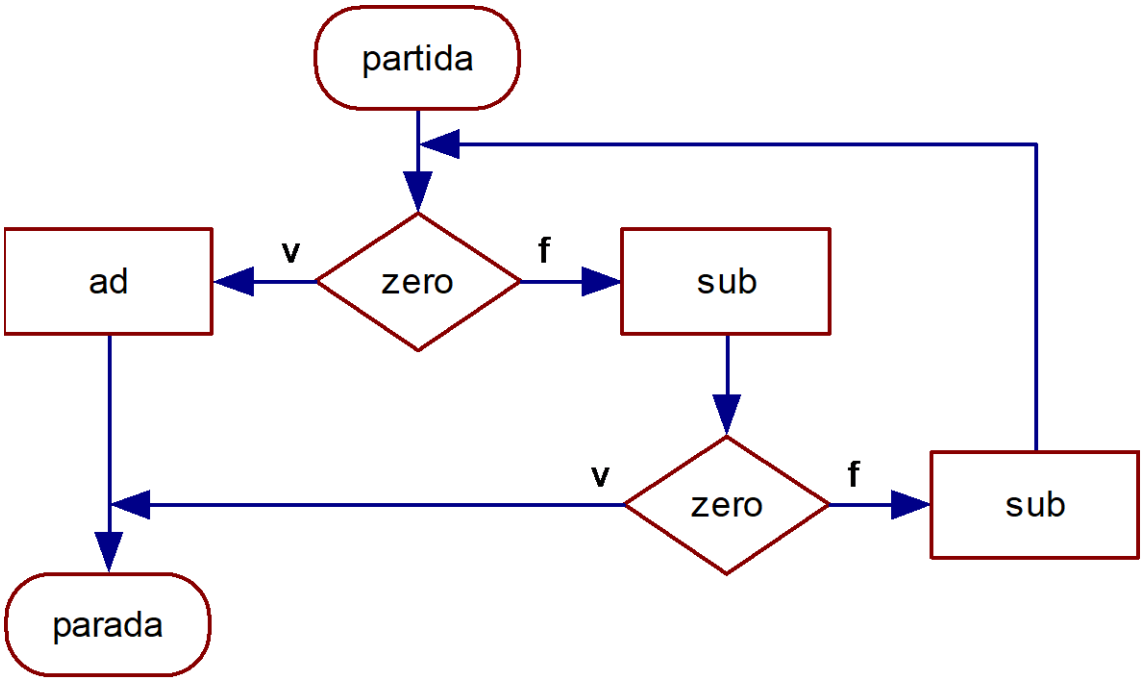


figura 2.22 programa monolítico

- a função computada ⟨par, um_reg⟩: N → N é tal que, para qualquer n ∈ N:

  ⟨par, um_reg⟩(n) = 1, se n é par;
  ⟨par, um_reg⟩(n) = 0, se n é ímpar.

- Ou seja, retorna o valor 1 sempre que a entrada é par, e zero, caso contrário.

- Suponha que existe um **programa iterativo** P_i que computa a mesma função, ou seja, que $\langle P_i, um\_reg\rangle: N \rightarrow N$ e:

$$\langle par, um\_reg\rangle = \langle P_i, um\_reg\rangle$$

  - Suponha que P_i é constituído de k operações sub.
  - $n \in N$ tal que $\mathbf{n \geq k}$.

- Então, é necessário que P_i execute n vezes a operação sub.
- Mas, como $\mathbf{n \geq k}$, então pelo menos uma das ocorrências de sub será executada mais de uma vez, ou seja, existe um ciclo iterativo (do tipo enquanto ou até) em P_i.
- Em qualquer caso, o ciclo terminará sempre na mesma condição, independentemente se o valor for par ou ímpar.
- Portanto, a computação resultante é incapaz de distinguir entre os dois casos, o que é um absurdo, pois é suposto que os dois programas são equivalentes fortemente.
- Logo, não existe um programa iterativo fortemente equivalente ao programa monolítico par.

Observação 2.19:
Poder computacional dos diversos tipos de programas.

- Os teoremas acima podem dar a falsa impressão de que o poder computacional da classe dos programas recursivos é maior que a dos monolíticos que, por sua vez, é maior que a dos iterativos.
- É importante constatar que as três classes de formalismos possuem o mesmo poder computacional, ou seja:
  - para qualquer programa recursivo (respectivamente, monolítico) e para *qualquer* máquina, existe um programa monolítico (respectivamente, iterativo) e *existe* uma máquina tal que as correspondentes funções computadas coincidem.
- Para efeito de análise de poder computacional, pode-se considerar máquinas distintas para programas distintos e não necessariamente existe uma relação entre as operações e testes (e a ordem de execução) dos programas.

2.4.2 Equivalência de Programas

Definição 2.20
Relação equivalência de programas em uma máquina

- Uma noção de equivalência mais fraca
- Sejam P e Q dois programas arbitrários, não necessariamente do mesmo tipo e uma máquina M qualquer. Então o par (P, Q) está na ***Relação Equivalência de Programas na Máquina*** M, denotado por:

$$P \equiv_M Q$$

se, e somente se, as correspondentes funções parciais computadas são iguais, ou seja:

$$\langle P, M\rangle = \langle Q, M\rangle$$

- Neste caso, P e Q são ditos ***Programas Equivalentes na Máquina*** M, ou simplesmente *Programas* M-*Equivalentes*.

- Existem máquinas nas quais não se pode provar a existência de um algoritmo para determinar se, dados dois programas, eles são ou não M-equivalentes.

2.4.3 Equivalência de Máquinas

Definição 2.21 – Simulação forte de máquinas

- Sejam $M = (V_M, X, Y, \pi_{X_M}, \pi_{Y_M}, \Pi_{F_M}, \Pi_{T_M})$ e
  $N = (V_N, X, Y, \pi_{X_N}, \pi_{Y_N}, \Pi_{F_N}, \Pi_{T_N})$
  duas máquinas arbitrárias.
- N *simula fortemente* M se, e somente se, para qualquer programa P para M, existe um programa Q para N, tal que as correspondentes funções parciais computadas coincidem, ou seja:
  $$\langle P, M\rangle = \langle Q, N\rangle$$
- É importante observar que a igualdade de funções exige que os conjuntos de domínio e contra-domínio sejam iguais.
- Pode-se contornar essa dificuldade, tornando menos restritiva a definição de simulação, através da noção de codificações.

Definição 2.22 Simulação de máquinas

- Sejam $M = (V_M, X_M, Y_M, \pi_{X_M}, \pi_{Y_M}, \Pi_{F_M}, \Pi_{T_M})$ e
  $N = (V_N, X_N, Y_N, \pi_{X_N}, \pi_{Y_N}, \Pi_{F_N}, \Pi_{T_N})$
  duas máquinas arbitrárias.
- N *Simula* M se, e somente se, para qualquer programa P para M, existe um programa Q para N e existem
  Função de Codificação c: $X_M \rightarrow X_N$
  Função de Decodificação **d: $Y_N \rightarrow Y_M$**
  tais que:
  $$\mathbf{\langle P, M\rangle = d \circ \langle Q, N\rangle \circ c}$$

Definição 2.23 Relação equivalência de máquinas

- Sejam M e N duas máquinas arbitrárias.
- Então o par (M, N) está na ***relação equivalência de máquinas***, se, e somente se:
  M simula N e N simula M.

2.5 Conclusões

- Foram introduzidos os conceitos de programa e de máquina, os quais são usados para construir as definições de computação e de função computada.
  Foram estudados três tipos de programas: monolítico, iterativo e recursivo. Os recursivos são mais gerais que os monolíticos os quais, por sua vez, são mais gerais que os iterativos.
- Apresentaram-se as noções de equivalência de programas e de máquinas

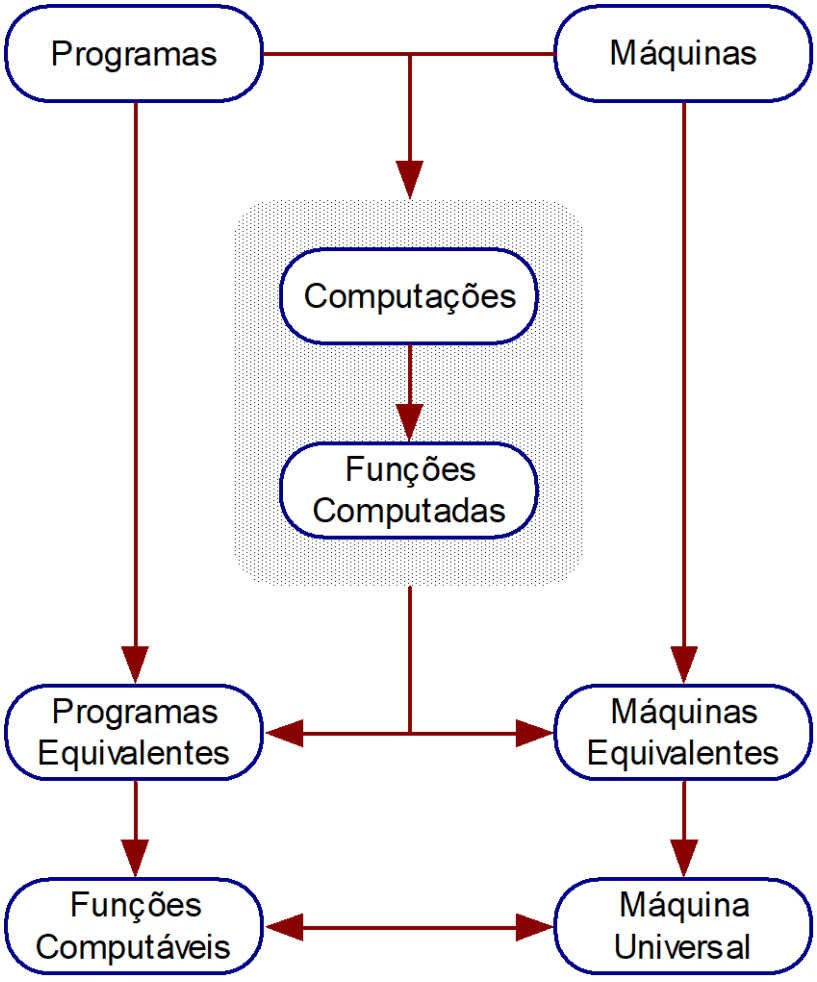


Figura 2.23 Relação entre os conceitos

2.6 Exercícios

Exercício 2.1
Identifique e compare construções análogas às usadas nas definições de programas monolítico, iterativo e recursivo em linguagens de programação como: Pascal, C ou outra de seu conhecimento.

Exercício 2.2
Desenhe um fluxograma que corresponde a cada um dos seguintes programas

a) P2 = ({ r1: faça √ vá_para r2 }, r1)
b) Composição até (programa iterativo);
c) Programa sem instrução alguma;
d) Programa sem instrução de parada.

Exercício 2.3
Relativamente a programas iterativos:

a) Em que situação a execução de: enquanto T faça V
   V pode não ser executado?
b) Por que a operação vazia √ constitui um programa iterativo?
c) Por que pode-se afirmar que:
   a tradução de um programa iterativo para um monolítico é imediata?

Exercício 2.4
Relativamente a computação:

a) Por que é possível afirmar que a computação de um programa monolítico em uma máquina, para um dado valor inicial de memória, é determinística?
b) Analogamente para um programa iterativo?
c) Analogamente para um programa recursivo?

Exercício 2.5
Caracterize e diferencie computação e função computada.

Exercício 2.6
Defina computação de programas iterativos em uma máquina.

Exercício 2.7
Dê a definição formal da função computada ⟨W, M⟩ de um programa iterativo W em uma máquina M.

Exercício 2.8
Defina computação finita para programas iterativos.

Exercício 2.9
Escreva um programa iterativo onde a computação seja infinita.

Exercício 2.10
Relativamente à função computada por um programa em uma máquina:

a) Considere o programa monolítico mon_b←a para a máquina dois_reg. A correspondente função computada ⟨mon_b←a, dois_reg⟩ é total?
b) Considere o programa monolítico comp_infinita para a máquina dois_reg. Para quais valores do domínio a função computada ⟨comp_infinita, dois_reg⟩ é definida?
c) Considere o programa recursivo qq_máquina e uma máquina M = (V, X, Y, π_X, π_Y, Π_F, Π_T) qualquer. Por que a correspondente função computada ⟨qq_máquina, M⟩ é indefinida para qualquer entrada?

Exercício 2.11
Traduza os programas monolíticos representados por fluxogramas em programas recursivos e simplifique se possível.

a) Fluxograma 1 representado na **Figura 2.24**;
b) Fluxograma 2 representado na **Figura 2.25**;
c) Fluxograma 3 representado na **Figura 2.26;**
d) Fluxograma 4 representado na **Figura 2.27;**
e) Fluxograma 5 representado na **Figura 2.28.**

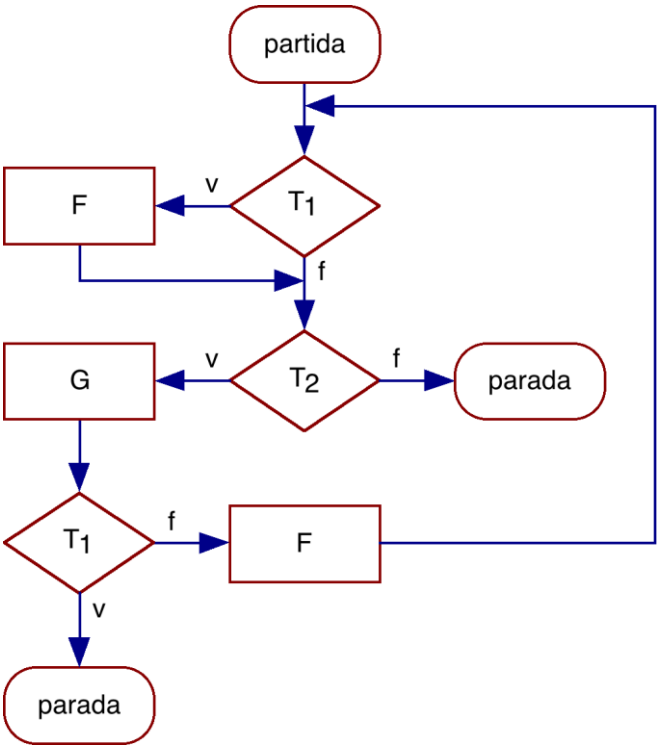


Figura 2.24 Fluxograma 1

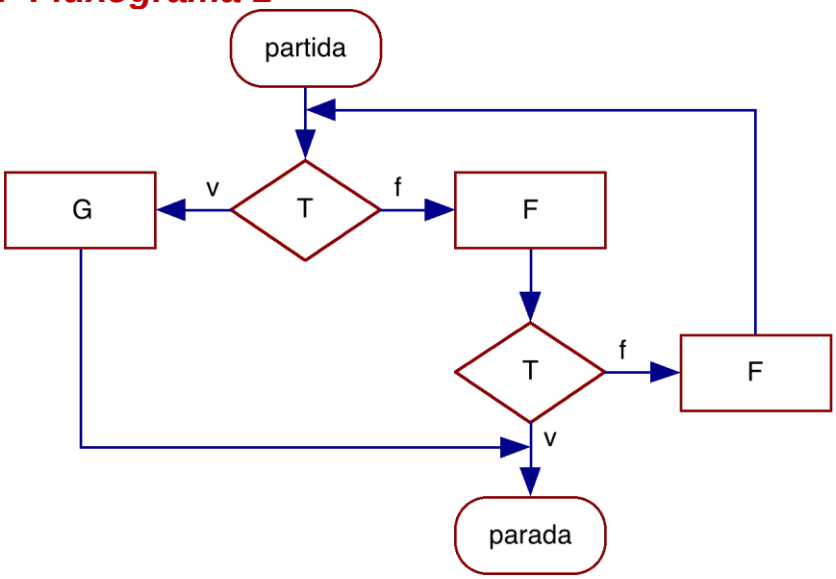


Figura2.25 Fluxograma 2

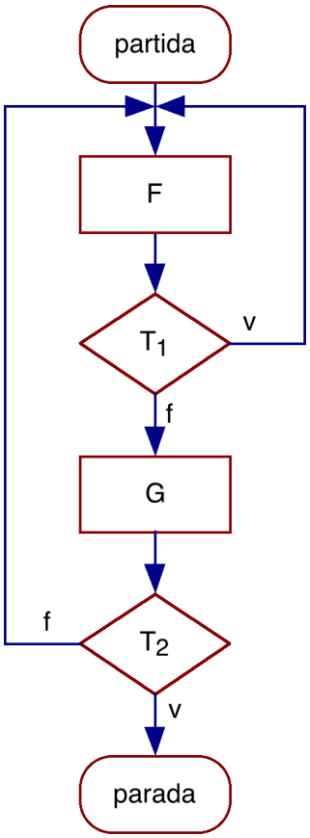


Figura 2.26 Fluxograma 3

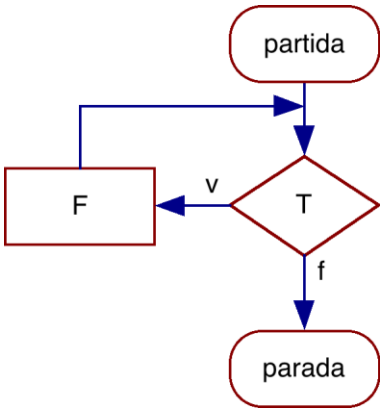


Figura 2.27 Fluxograma 4

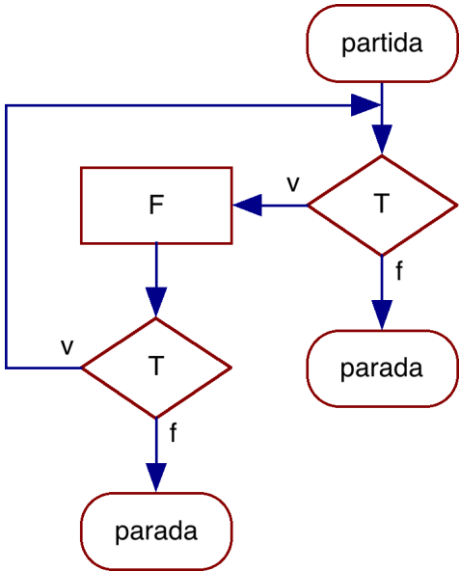


Figura 2.28 Fluxograma 5

Exercício 2.12

Traduza o programa iterativo representado na ***Figura 2.29*** em programa monolítico, nas formas de:

a) Fluxograma;

b) Instruções rotuladas.

```
(se    T1
então enquanto T2
      faça  (até   T3
            faça  (V;W))
senão (√))
```

Figura 2.29 Programa iterativo

Exercício 2.13

Traduza o programa recursivo representado na ***Figura 2.30*** em programa iterativo.

```
P é R1 onde
      R1 def (se T então F;R2 senão R1),
      R2 def G; (se T então F;R1 senão √)
```

Figura 2.30 Programa recursivo

Exercício 2.14

Suponha que se escreva $M_1 \leq M_2$ se a máquina M_2 simula M_1, mas M_1 pode ter na sua definição outros testes e operações a mais. Qual é a relação entre $\langle P, M_1 \rangle$ e $\langle P', M_2 \rangle$ se $M_1 \leq M_2$? Qual a relação do poder computacional da máquina M_1 em relação a máquina M_2?

Exercício 2.15

Suponha que na definição de uma máquina, sejam admitidas funções parciais na especificação dos identificadores de operações e testes. Dê a definição apropriada da função computada por um programa em uma máquina em tal caso.

Exercício 2.16

Traduza os programas iterativos W_1 e W_2 definidos na Figura e na Figura para programas recursivos.

Programa Iterativo W_1

```
enquanto T
faça  (F;(se T então faça √ senão faça G))
```

Figura 2.31 Programa iterativoW_1

Programa Iterativo W_2

```
enquanto T
faça  (F;enquanto T faça (F);G)
```

Figura 2.32 Programa iterativoW_2

Exercício 2.17

Sobre computação de programa recursivo em uma máquina, marque a afirmação incorreta:

a) É um histórico das instruções executadas e os correspondentes valores de memória;
b) Um teste e referência a uma sub-rotina não alteram o valor corrente da memória;
c) Em uma computação finita, a expressão (√) ocorre no último par da cadeia e não ocorre em qualquer outro par;
d) Para um dado valor inicial de memória, a correspondente cadeia de computação é única, ou seja, a computação é não-determinística;
e) Em uma computação infinita, expressão alguma da cadeia é (√).

Exercício 2.18

Qual das alternativas abaixo representa a hierarquia induzida pela relação Equivalência Forte de Programas?

a) Programas Recursivos ≤ Programas Monolíticos ≤ Programas Iterativos;
b) Programas Monolíticos ≤ Programas Recursivos ≤ Programas Iterativos;
c) Programas Iterativos ≤ Programas Recursivos ≤ Programas Monolíticos;
d) Programas Monolíticos ≤ Programas Iterativos ≤ Programas Recursivos;
e) Programas Iterativos ≤ Programas Monolíticos ≤ Programas Recursivos.

Exercício 2.19

Relativamente à Equivalência de Programa, marque a alternativa errada:

a) Qualquer programa monolítico tem um fortemente equivalente recursivo;
b) Os programas recursivos podem simular qualquer programa iterativo;
c) Para todo programa iterativo, existe um monolítico equivalente fortemente;
d) Para todo programa recursivo, existe um iterativo equivalente fortemente.
e) Para todo programa recursivo, existe um equivalente monolítico na máquina Norma.

Exercício 2.20

Dado uma máquina qualquer e considerando a análise dos diversos tipos de programa nesta máquina, marque a alternativa correta:

a) Qualquer programa iterativo pode ser traduzido para fluxograma;
b) Qualquer fluxograma pode ser traduzido para programa iterativo;
c) Qualquer programa recursivo “faz alguma coisa”, já que a operação vazia não constitui um programa recursivo.
d) Dado um programa monolítico, não necessariamente existe um correspondete programa monolítico fortemente equivalente;
e) Qualquer programa iterativo que possuir Composição Enquanto, denotada por "enquanto X faça Y", não possui um correspondente programa recursivo fortemente equivalente.

Exercício 2.21

Marque a alternativa correta:

a) Desvios incondicionais são permitidos em um programa iterativo;
b) Um programa descreve completamente uma computação, visto que contém a natureza das operações ou dos testes.
c) Para todo programa monolítico, existe necessariamente um programa iterativo fortemente equivalente;
d) Duas máquinas são equivalentes se uma pode simular a outra e vice-versa;
e) Dois programas são fortemente equivalentes se suas funções computadas coincidem para uma determinada máquina.

Exercício 2.22

Sobre Equivalência Forte de Programas, analise as afirmações abaixo:

I. Dois programas são fortemente equivalentes se e somente se os dois são do mesmo tipo e suas funções computadas são iguais;
II. Nem todo programa recursivo possui um monolítico fortemente equivalente pelo fato de que o primeiro é mais genérico que o segundo;
III. As funções computadas por programas fortemente equivalentes possuem a propriedade de que as mesmas operações podem ser efetuadas em qualquer ordem, independente do significado das mesmas, pois a saída é a mesma.

Marque a alternativa correta:

a) Apenas I;
b) Apenas II;
c) Apenas III;
d) Apenas II e III;
e) I, II e III

Exercício 2.23

Analise as afirmações abaixo:

I. A palavra "Elεevaεdor" pode ser formada a partir do alfabeto $\sum$ = {a, d, e, l, o, r, v, w};
II. Sejam P e Q, dois programas. Se para uma maquina M, $\langle P,M\rangle \neq \langle Q,M\rangle$, então os programas não são equivalentes nessa máquina;
III. Um programa pode ser representado através de um fluxograma que contenha $\surd$ como instrução de parada;
IV. Cabe a máquina suprir o significado, ou seja, dar semântica, aos identificadores das operações e dos testes de um programa.

Marque a alternativa *incorreta*:

a) Apenas II e III estão incorretas;
b) Apenas IV e II estão corretas;
c) Ou I e IV estão corretas ou III está correta;
d) Ou I e II estão corretas ou III e IV estão incorretas;
e) Ou I esta correta ou a III está correta.

Exercício 2.24

No contexto da equivalência forte, marque a alternativa correta:

a) Para qualquer programa monolítico, existe um programa iterativo equivalente;
b) Para qualquer programa recursivo, existe um programa monolítico equivalente;
c) Para qualquer programa recursivo, existe um programa iterativo equivalente;
d) Para qualquer programa monolítico, existe um programa recursivo equivalente;
e) Nenhuma das alternativas anteriores está correta.

Exercício 2.25

A Relação de Equivalência Forte entre programas permite:

a) Identificar programas iguais em diferentes classes de equivalência;
b) Identificar programas iguais cujas funções computadas são iguais;
c) Afirmar que todo programa monolítico tem um iterativo fortemente equivalente;
d) Identificar diferentes programas em uma mesma classe de equivalência;
e) Afirmar que funções computadas por programas equivalentes fortemente tem a propriedade de que as mesmas operações são efetuadas numa ordem muito semelhante.

Exercício 2.26

Sobre a definição de programas recursivos, analise as seguintes afirmações:

I. O programa tem, em sua definição, uma expressão inicial, indicando qual a primeira rotina a ser computada.
II. Um programa recursivo é capaz de algoritmizar um conjunto de problemas maior do que os programas iterativos são capazes.
III. A instrução representa um teste em um programa recursivo: *R def se T então √senão F;*
IV. Um programa recursivo termina quando há uma chamada de sub-rotina que não está definida.
V. A representação de um programa recursivo pode ser reduzida, substituindo-se as chamadas de sub-rotinas pelas expressões executadas por estas, desde que essas sub-rotinas não sejam testes.

Marque a alternativa correta:

a) Apenas I e III estão corretas;
b) Apenas I e V estão corretas;
c) Apenas I, III e V estão corretas;
d) Apenas I, II e V estão corretas;
e) Apenas II, III e IV estão corretas.

Exercício 2.27

Sobre a Relação Equivalência Forte de Programas, marque a alternativa errada:

a) As funções computadas por programas equivalentes fortemente têm a propriedade de que as mesmas operações são efetuadas na mesma ordem, independentemente do significado dos mesmos;
b) Analisando programas equivalentes fortemente, é possível dizer qual o "mais otimizado" é estruturalmente;
c) Como para qualquer programa iterativo existe um monolítico equivalente fortemente, a classe de programas monolíticos tem maior poder computacional que os iterativos;
d) Para qualquer programa iterativo, existe um programa recursivo equivalente fortemente;
e) Permite identificar diferentes programas cujas funções computadas coincidem, para qualquer máquina.

Exercício 2.28

Sobre a equivalência de programas e máquinas marque a alternativa correta:

a) A relação de equivalência entre duas máquinas ocorre quando ao menos uma delas é capaz de simular todas as funções da outra.
b) A relação de equivalência forte entre um programa monolítico Pm e outro iterativo Pi pode ser verificada, transformando-se cada um deles em programas recursivos fortemente equivalentes (Pr,m = Pm e Pr,i = Pi) então, aplicando-se um método conhecido para testar equivalência entre Pr,m e Pr,i a fim de provar a relação entre todos (Pr,m = Pm = Pr, i = Pi).
c) Não existe qualquer método capaz de verificar a equivalência forte entre dois programas iterativos;
d) Se dois programas P e Q são equivalentes em uma máquina de traços M, então P e Q são fortemente equivalentes;
e) A simulação forte entre máquinas deve ser feita usando programas iguais;

Exercício 2.29

Com relação à equivalência de programas, marque a alternativa que descreve corretamente a igualdade de funções parciais, tendo-se duas funções parciais f, g: $X \rightarrow Y$ para cada $x \in X$:

a) Ambas as funções são definidas e f(x) = g(x);
b) Ou ambas são definidas e f(x)=g(x) ou ambas f(x) e g(x) são indefinidas;
c) Ou ambas são definidas e f(x)=g(x) ou pelo menos uma delas f(x) ou g(x) é indefinida;
d) Funções parciais iguais são encontradas apenas em programas iterativos, pois são os únicos que admitem funções;
e) Nenhuma das alternativas anteriores está correta.

Exercício 2.30

Sobre equivalência de programas e maquinas, analise as seguintes afirmações:

I. Se um programa P_1 é equivalente a um programa P_2 em uma máquina de rastros, então eles são fortemente equivalentes;
II. Seja R um programa recursivo, M um programa monolítico e I um programa iterativo. *Não* é verdade que M é mais geral que I, que por sua vez é mais geral que R;
III. Sabendo que a máquina M_1 pode simular a máquina M_2, então M_1 é equivalente a M_2.

Marque a alternativa correta:

a) Apenas I está correta;
b) Apenas I e II estão corretas;
c) Apenas I e III estão corretas;
d) Apenas III está correta;
e) Apenas II e III estão corretas.