2 PROGRAMAS, MÁQUINAS E COMPUTAÇÕES

2.1 Programas
2.1.1 Programa Monolítico
2.1.2 Programa Iterativo
2.1.3 Programa Recursivo

2.2 Máquinas

2.3 Computações e Funções Computadas
2.3.1 Computação
2.3.2 Função Computada

2.4 Equivalências de Programas e Máquinas
2.4.1 Equivalência Forte de Programas
2.4.2 Equivalência de Programas
2.4.3 Equivalência de Máquinas

2.5 Verificação da Equivalência Forte de Programas
2.5.1 Máquina de Traços
2.5.2 Instruções Rotuladas Compostas
2.5.3 Equivalência Forte de Programas Monolíticos

2.6 Conclusão

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.

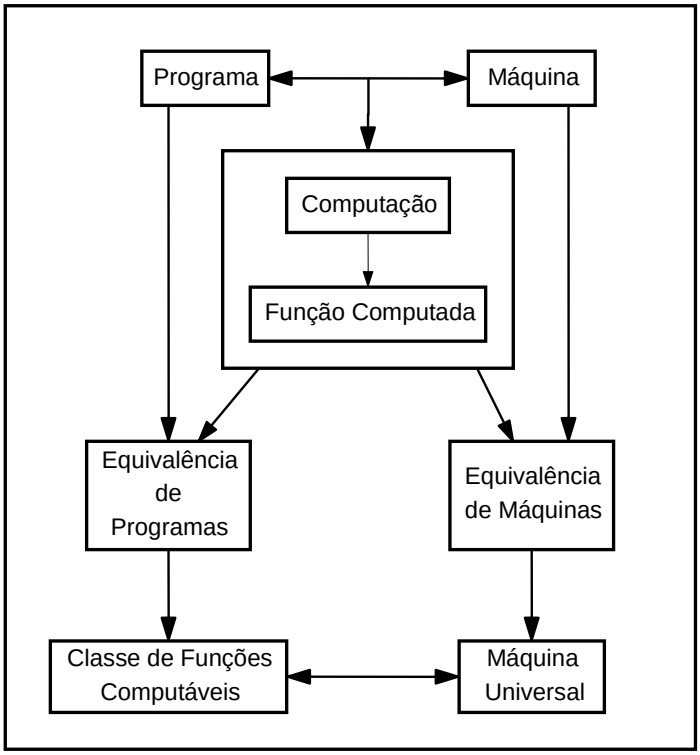
Computação e função computada.

- *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 equivalentes fortemente*: 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.

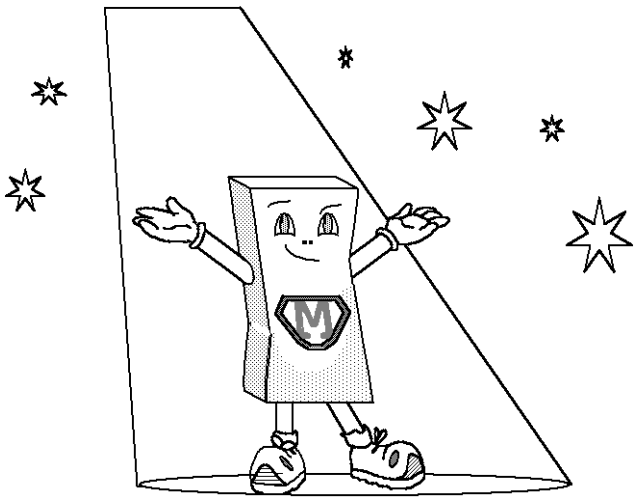
Composição de Instruções

- *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 ✓.

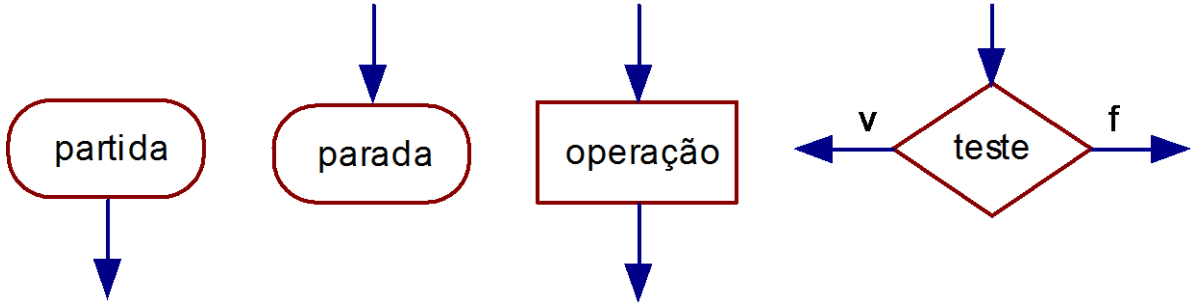
2.1.1 Programa Monolítico



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.

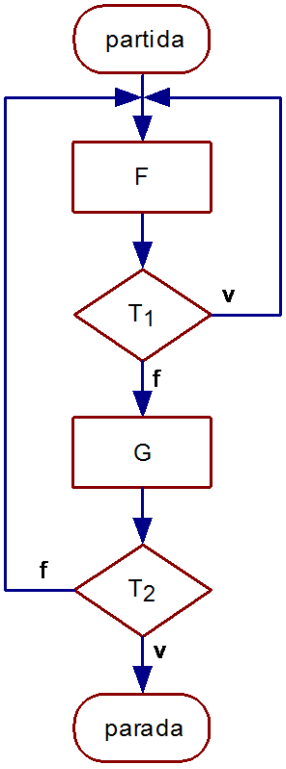
Fluxogramas

- 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



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
```

Formalização de Programa Monolítico

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 4 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*.

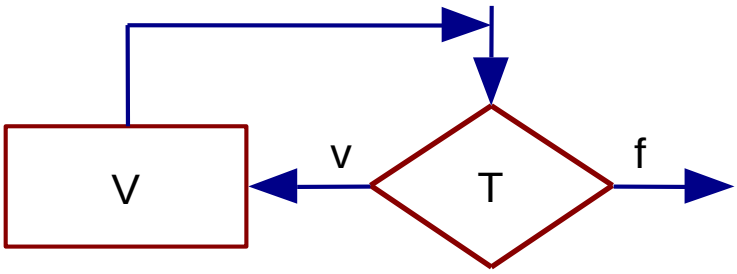
f) *Composição Até*. até T faça (V)

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)



EXEMPLOS 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   (✓))
```

Os programas tipo while de 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 é E_0 onde R_1 *def* E_1, R_2 *def* E_2, ..., R_n *def* E_n

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

- 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;S *onde*
R *def* F;(se T então R senão G;S),
S *def* (se T então ✓ senão F;R)

- 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 $n \neq 0$.

- 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)
```

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
```

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

```
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
```

- 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

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

Programa Monolítico comp_infinita

```
1: faça adiciona_b vá_para 1
```

- Para o valor inicial de memória (3, 0), qual a computação?

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 $D_0 = E_0$; ✓ 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:

$D_k = (✓;C)$

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

```
qq máquina é R onde
                R def R
```

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

$$(R, v_0)(R, v_0)(R, v_0)...$$

EXEMPLO 2.12 Computação Finita de Programa Recursivo na Máquina de Um Registrador.

- Considere 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: $N \rightarrow N$ é a função identidade em N

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

sub: $N \rightarrow N$ 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: $N \rightarrow \{verdadeiro, falso\}$ tal que, $\forall n \in N$,

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

- e o programa recursivo duplica para máquina um_reg .

Programa Recursivo duplica

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

- 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, senão
(sub; (R;ad;ad); ✓, 3)	caso 4
((R;ad;ad); ✓, 2)	subtraiu 1 da memória
(R; (ad;ad); ✓, 2)	caso 4
((se zero então ✓ senão (sub;R;ad;ad)); (ad;ad); ✓, 2)	caso 3
((sub;R;ad;ad); (ad;ad); ✓, 2)	como n ≠ 0, senão
(sub; (R;ad;ad); (ad;ad); ✓, 2)	caso 4
((R;ad;ad); (ad;ad); ✓, 1)	subtraiu 1 da memória
(R; (ad;ad); (ad;ad); ✓, 1)	caso 4
((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, senão
(sub; (R;ad;ad); (ad;ad); (ad;ad); ✓, 1)	caso 4
((R;ad;ad); (ad;ad); (ad;ad); ✓, 0)	subtraiu 1 da memória
(R; (ad;ad); (ad;ad); (ad;ad); ✓, 0)	caso 4
((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, então
((ad;ad); (ad;ad); (ad;ad); ✓, 0)	caso 1
(ad; (ad; (ad;ad); (ad;ad); ✓), 0)	caso 4
((ad; (ad;ad); (ad;ad); ✓), 1)	adicionou 1 na memória
(ad; ((ad;ad); (ad;ad); ✓), 1)	caso 4
(((ad;ad); (ad;ad); ✓), 2)	adicionou 1 na memória
(ad; (ad; (ad;ad); ✓), 2)	caso 4
((ad; (ad;ad); ✓), 3)	adicionou 1 na memória
(ad; ((ad;ad); ✓), 3)	caso 4
(((ad;ad); ✓), 4)	adicionou 1 na memória
(ad; ((ad); ✓), 4)	caso 4
(((ad); ✓), 5)	adicionou 1 na memória
(ad; (✓), 5)	caso 4
((✓), 6)	adicionou 1 na memória
(✓, 6)	fim da recursão

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,

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

tal que, para qualquer $n \in N$, 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

$$\langle \text{comp\_infinita, dois\_reg}\rangle: N \rightarrow N,$$

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 sobre uma máquina é análoga à de um monolítico:

- a computação inicia na expressão 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 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:

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

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:

- $D_0 = E_0$; ✓ é expressão inicial de P
- $v_0 = \pi_X(x)$
- $Dn =$ ✓

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

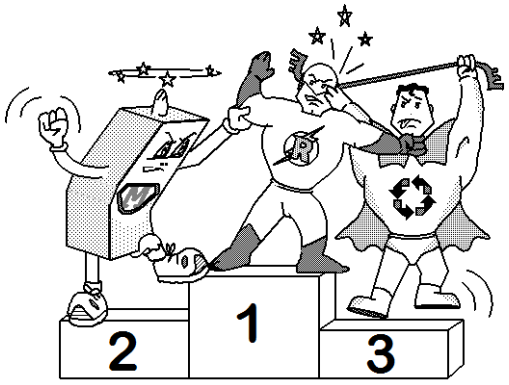
EXEMPLO 2.15 Função Computada por Programa Recursivo.

- 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$\rangle: X \rightarrow Y$ 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:
  $$\langle \text{duplica, um\_reg}\rangle: N \rightarrow N$$
  tal que, para qualquer $n \in N$:
  $$\langle \text{duplica, um\_reg}\rangle(n) = 2n.$$

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)

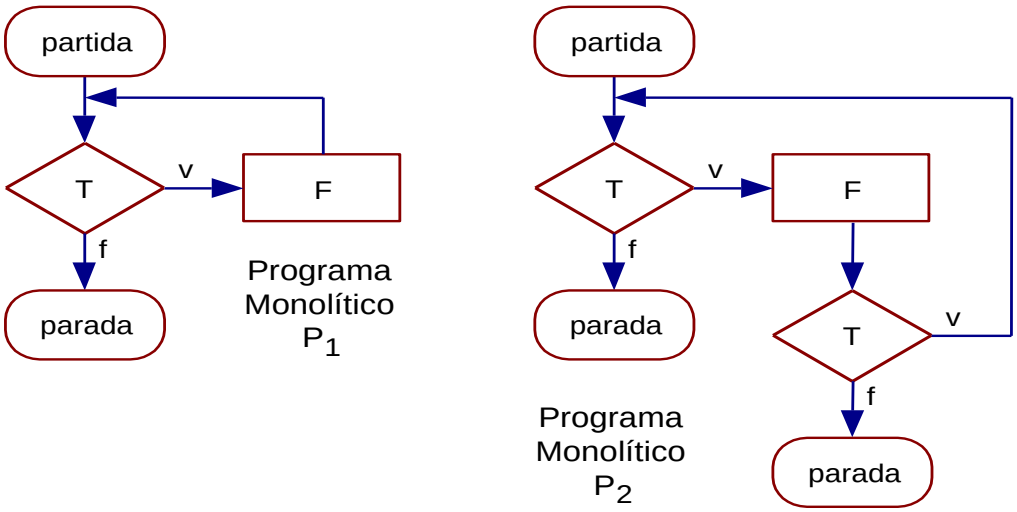
2.4.1 Equivalência Forte de Programas

Definição 2.13 Relação Equivalência Forte de Programas, Programas Equivalentes Fortemente.

- 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 Equivalentes Fortemente.*

EXEMPLO 2.17 Programas Equivalentes Fortemente.

- 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 equivalentes fortemente.



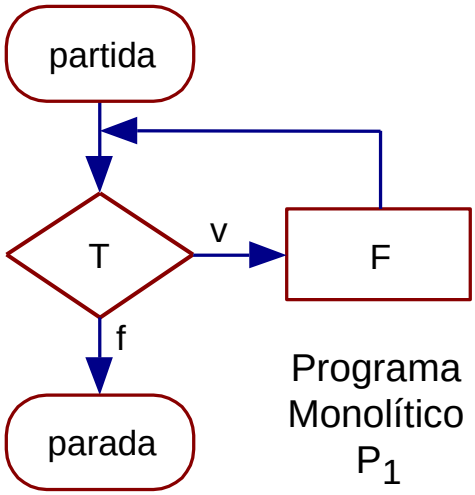
Programa Iterativo P_3

```
enquanto T
faça (F)
```

Programa Recursivo P_4

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

Programa monolítico



Programa Monolítico P_1

```
1:seTentão vá para2 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

P_4 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, v)
((se T então F;R senão ✓), v)
(F;R, v)

(R, $\pi_F(v)$)
((se T então F;R senão ✓), $\pi_F(v)$)
(F;R, $\pi_F(v)$)

(R, $\pi_F^2(v)$)
((se T então F;R senão ✓), $\pi_F^2(v)$)
(F;R, $\pi_F^2(v)$)

...
(R, $\pi_F^n(v)$)
((se T então F;R senão ✓), $\pi_F^n(v)$)
(✓, $\pi_F^n(v)$)

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

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

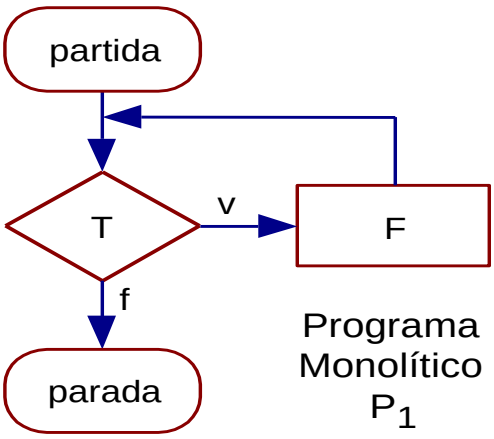
Portanto, $P_1 \equiv P_4$ pois, para qualquer máquina M, tem-se que: $\langle P_1, M\rangle = \langle P_4, M\rangle$

Relação Equivalência Forte de Programas

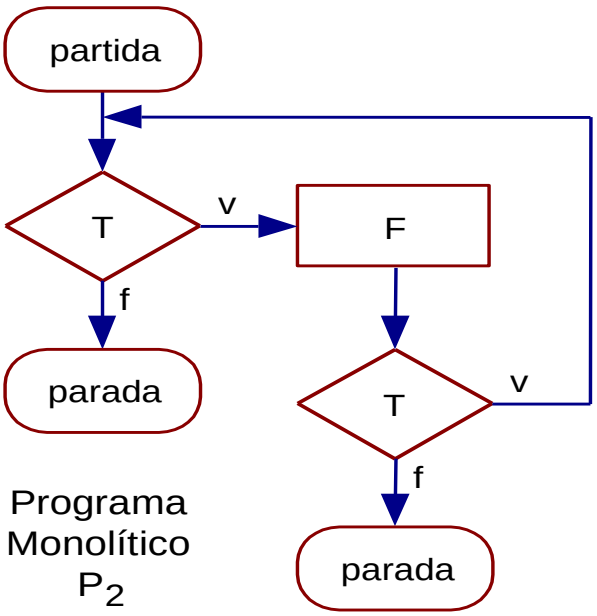
- permite identificar diferentes programas em uma mesma classe de equivalência, ou seja, identificar diferentes programas cujas funções computadas coincidem, para qualquer máquina;

- as funções computadas por programas equivalentes fortemente têm a propriedade de que os mesmos testes e as mesmas operações são efetuados na mesma ordem, independentemente do significado dos mesmos;

- 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

Hierarquia induzida pela Relação Equivalência Forte de Programas

Teorema 2.14 Equivalência Forte de Programas: Iterativo → Monolítico.

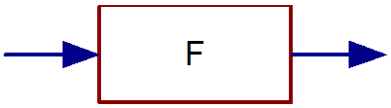
Para qualquer programa iterativo P_i, existe um programa monolítico P_m, tal que $P_i \equiv P_m$.

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:

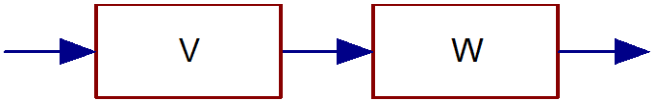
b) Para cada identificador de operação F de P_i corresponde o seguinte fluxograma elementar:

F



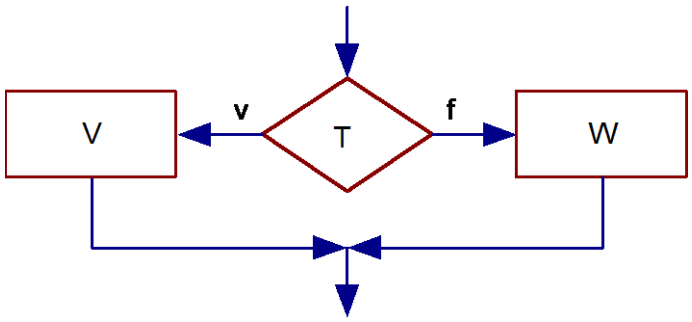
c) Composição Seqüencial.

V;W



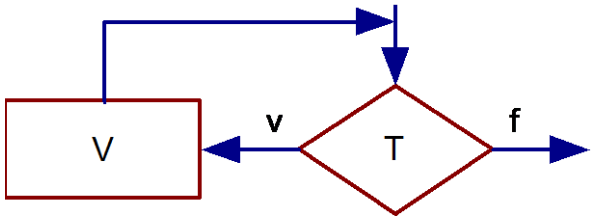
d) Composição Condicional.

(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 `rk` é da forma: `rk: faça F vá_para rk'`
então Ek é a seguinte expressão de sub-rotinas: `F;Rk'`

b) *Teste*. Se `rk` é da forma:
`rk: se T então vá_para rk' senão vá_para rk"`
então Ek 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 $n \in N$:

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

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

$$\langle \text{duplica, um\_reg} \rangle = \langle P_m, \text{um\_reg} \rangle$$

- Suponha que P_m é constituído de k operações ad.
- Suponha $n \in N$ tal que $n \geq k$.

Então,

- para que $\langle P_m, \text{um\_reg} \rangle$ (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, os mesmos testes e as mesmas operações são efetuados na mesma ordem; portanto, o programa monolítico correspondente não pode intercalar testes de controle de fim de ciclo na seqüência de operações ad (no programa recursivo, as operações ad não são intercaladas por qualquer outra operação ou teste) .
- 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:

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 ou testes 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.

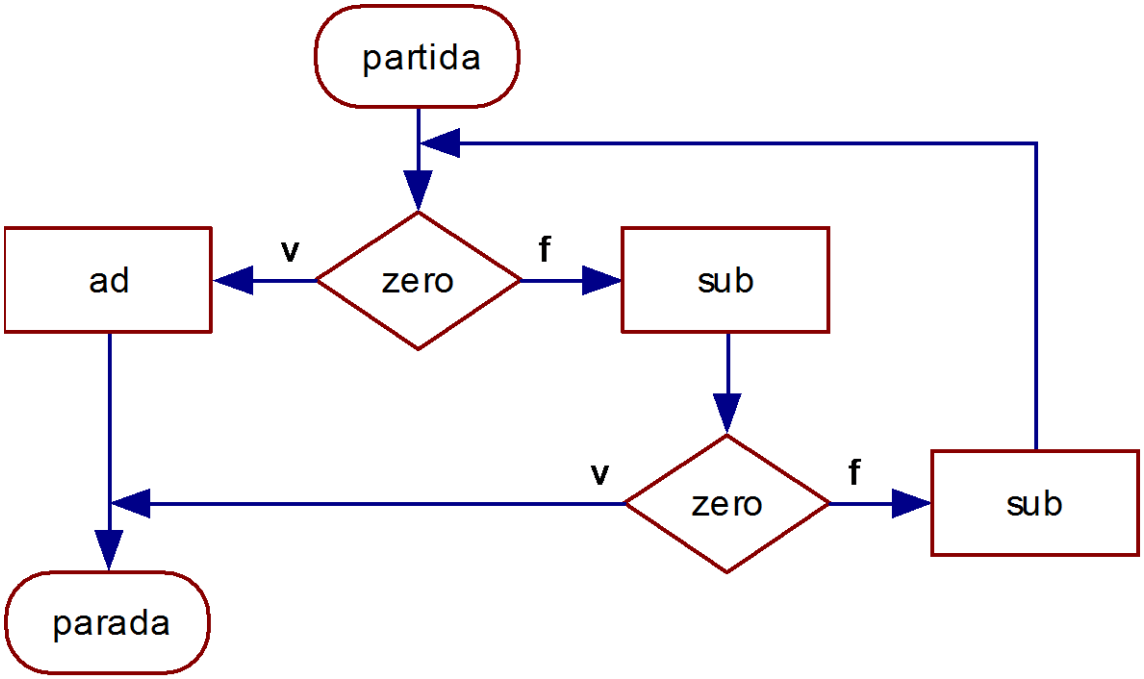
Teorema 2.18 Equivalência Forte de Programas: Monolítico ↛ Iterativo.

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



- 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 $n \geq k$.

- Então, é necessário que P_i execute n vezes a operação sub.
- Mas, como $n \geq k$, 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 Equiv 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 Verificação da Equivalência Forte de Programas

- Mostra-se que existe um algoritmo para decidir a equivalência forte entre programas monolíticos.
- Como todo programa iterativo possui um programa monolítico equivalente fortemente, o mesmo pode ser afirmado para programas iterativos.

A verificação de que dois programas monolíticos são equivalentes fortemente usa os seguintes conceitos:

a) *Máquina de Traços*. Produz um rastro ou histórico (denominado *traço*) da ocorrência das operações do programa. Neste contexto, dois programas são equivalentes fortemente se são equivalentes em qualquer máquina de traços.

b) *Programa Monolítico com Instruções Rotuladas Compostas*. Instruções rotuladas compostas constituem uma forma alternativa de definir programas monolíticos. Basicamente, uma instrução rotulada composta é um teste da seguinte forma :

r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3

- De fato, usando máquinas de traços, é fácil verificar que, para um dado fluxograma, é possível construir um programa equivalente fortemente, usando instruções rotuladas compostas.
- Instruções rotuladas compostas induzem a noção de rótulos equivalentes fortemente, a qual é usada para determinar se dois programas são ou não equivalentes fortemente.

2.5.1 Máquina de Traços

- Uma máquina de traços não executa as operações propriamente ditas, mas apenas produz um histórico ou rastro da ocorrência destas, denominado de *traço*.
- Essas máquinas são de grande importância para o estudo da equivalência de programas, pois, se dois programas são equivalentes em qualquer máquina de traços, então são equivalentes fortemente.

Definição 2.24 Máquina de Traços.

Uma *Máquina de traços* é uma máquina

$$M = (Op^*, Op^*, Op^*, id_{Op}^*, id_{Op}^*, \Pi_F, \Pi_T)$$

onde:

Op^* *conjunto de palavras de operações* onde $Op = \{F, G, ...\}$ o qual corresponde, simultaneamente, aos conjuntos de valores de memória, de entrada e de saída;

id_{Op}^* *função identidade* em Op^*, a qual corresponde, simultaneamente, às funções de entrada e saída;

Π_F *conjunto de interpretações de operações* onde, para cada identificador de operação F de Op, a interpretação π_F: $Op^* \rightarrow Op^*$ é tal que, para qualquer $w \in Op^*$, $\pi_F(w)$ resulta na concatenação do identificador F à direita de w, ou seja:

$$\pi_F(w) = wF$$

Π_T *conjunto de interpretações de testes,* tal que, para cada identificador de teste T:

π_T: $Op^* \rightarrow \{verdadeiro, falso\}$ é função de Π_T

- efeito de cada operação interpretada por uma máquina de traços é simplesmente o de acrescentar o identificador da operação à direita do valor atual da memória.
- a função computada consiste em um histórico das operações executadas durante a computação.

Definição 2.25 Função Induzida por um Traço em uma Máquina

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

$Op = \{F, G, H, ...\}$ conj. de operações interpretadas em Π_F e

$w = FG...H$ um traço possível de M, ou seja, $w \in Op^*$

A *Função Induzida pelo Traço* w *na Máquina* M, denotada por:

$$[w, M]: X \rightarrow V$$

é a função (total):

$$[w, M] = \pi_H \circ ... \circ \pi_G \circ \pi_F \circ \pi_X$$

A função [w, M] aplicada a uma entrada $x \in X$ é denotada por:

$$[wx, M] = \pi_H \circ ... \circ \pi_G \circ \pi_F \circ \pi_X(x)$$

- Portanto, a função induzida por um traço nada mais é do que a função resultante da composição das interpretações das diversas operações que constituem o traço.

Teorema 2.26 Equivalência Forte de Programas ⇔ Equivalência de Programas em Máquinas de Traços

Sejam P e Q dois programas arbitrários, não necessariamente do mesmo tipo. Então: $P \equiv Q$ se, e somente se, para qualquer máquina de traços M, $P \equiv_M Q$.

Corolário 2.27 Equivalência Forte de Programas ⇔ Equivalência de Programas em Máquinas de Traços.

Sejam P e Q dois programas arbitrários, não necessariamente do mesmo tipo. Então: $P \equiv Q$ se, e somente se, para qualquer máquina de traços M, $\langle P, M\rangle(\varepsilon) = \langle Q, M\rangle(\varepsilon)$.

2.5.2 Instruções Rotuladas Compostas

- Instruções rotuladas compostas possuem somente um formato, ao contrário das instruções rotuladas, as quais podem ter dois formatos: de operação e de teste.

Definição 2.28 Instrução Rotulada Composta.

- Uma *Instrução Rotulada Composta* é uma seqüência de símbolos da seguinte forma :

r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3

  - r_2 e r_3 são ditos *rótulos sucessores* de r_1;
  - r_1 é dito *rótulo antecessor* de r_2 e r_3.

Definição 2.29 Programa Monolítico com Instruções Rotuladas Compostas

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

- Adicionalmente, relativamente ao conjunto I, tem-se que:
  - 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*.

- Para simplificar, iremos considerar somente o caso particular em que os programas possuem um *único* identificador de teste, denotado por T.
- Assim, uma instrução rotulada composta da forma:

r_1:se T então faça F vá_para r_2 senão faça G vá_para r_3

pode ser abreviada simplesmente por:

r_1: (F, r_2), (G, r_3)

Definição 2.30 Algoritmo:

Fluxograma → Rotuladas Compostas.

- Entrada: um fluxograma de um programa monolítico P
- Saída: um programa monolítico P' constituído por instruções rotuladas compostas
- *Nós* - componentes elementares de partida, parada e operação de um fluxograma

a) *Rotulação de Nós.* Rotula-se cada nó do fluxograma. Suponha que existe um único componente elementar de parada, ao qual é associado o identificador ε (palavra vazia). O rótulo correspondente ao nó partida é o Rótulo Inicial do programa P'.

b) *Instruções Rotuladas Compostas*. A construção de uma instrução rotulada composta parte do nó partida e segue o caminho do fluxograma.

b.1) *Teste.* Para um teste, a correspondente instrução rotulada composta é: r_1: (F, r_2), (G, r_3)

b.2) *Operação.* Para uma operação, a correspondente instrução rotulada composta é:

$$r_1: (F, r_2), (F, r_2)$$

b.3) *Parada*. Para uma parada, a correspondente instrução rotulada composta é como segue:

$$r: (\text{parada}, \varepsilon), (\text{parada}, \varepsilon)$$

b.4) *Testes Encadeados*. No caso de testes encadeados, segue-se o fluxo até que seja encontrado um nó, resultando na seguinte instrução rotulada composta:

$$r_1: (F, r_2), (G, r_3)$$

b.5) *Testes Encadeados em Ciclo Infinito*. Para um ciclo infinito determinado por testes encadeados, a correspondente instrução rotulada composta é:

$$r_1: (F, r_2), (\text{ciclo}, \omega)$$

Neste caso, deve ser incluída, adicionalmente, uma instrução rotulada composta correspondente ao ciclo infinito:

$$\omega: (\text{ciclo}, \omega), (\text{ciclo}, \omega)$$

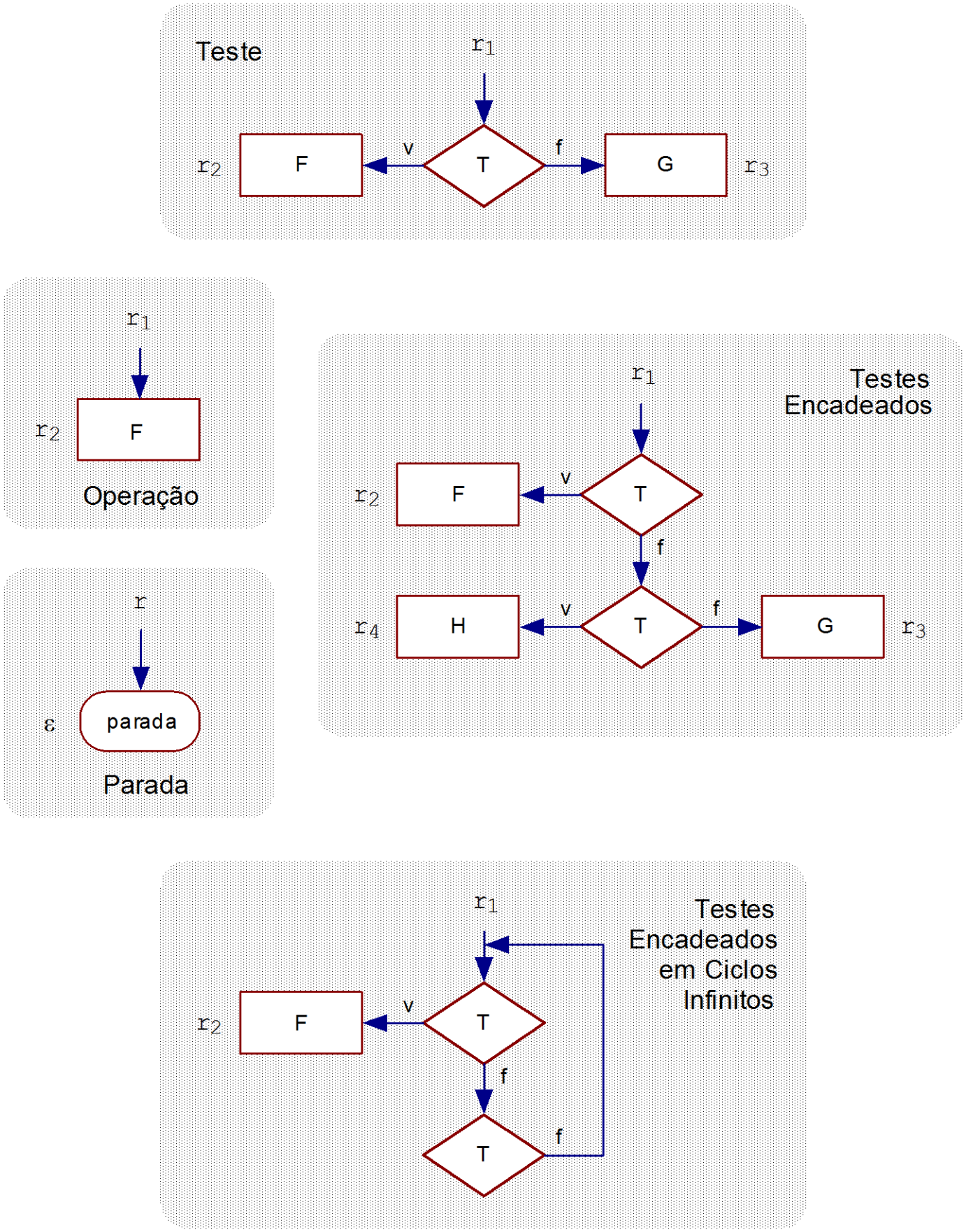
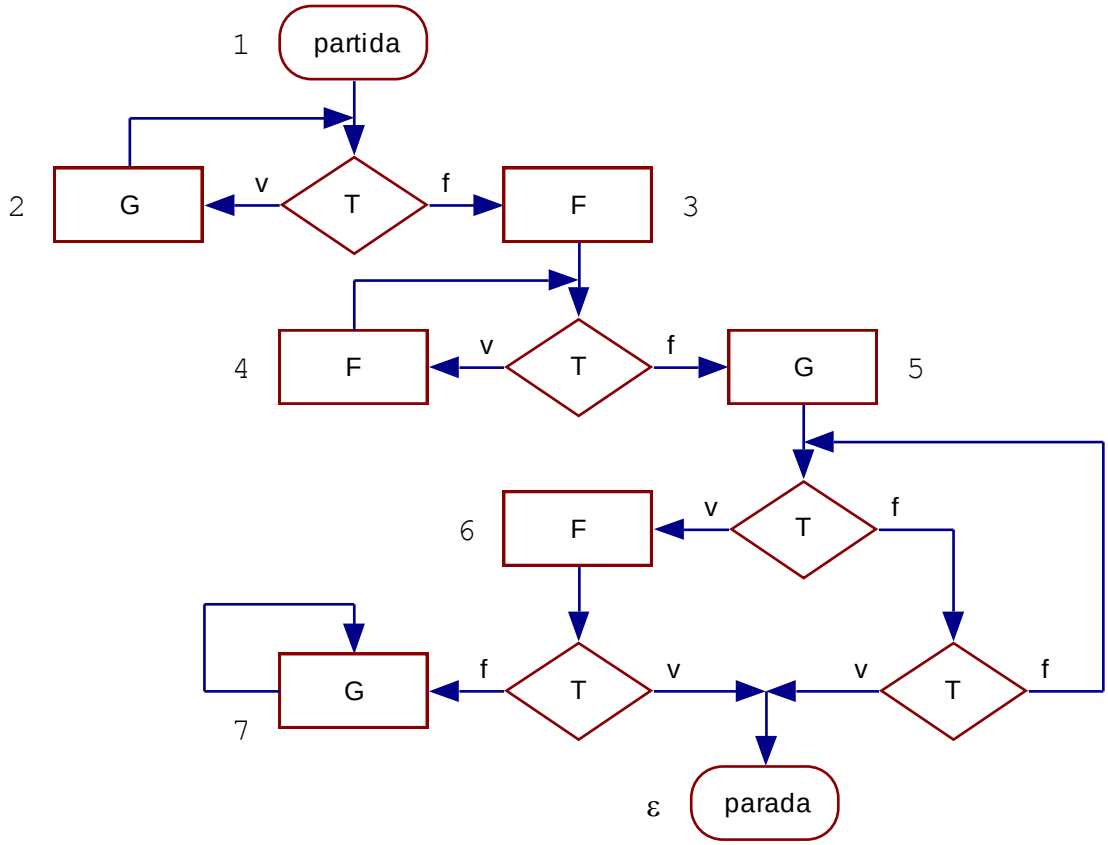


Figura 2.23 Instruções rotuladas compostas

EXEMPLO 2.18 Algoritmo Fluxograma → Rotuladas Compostas.

- Considere o programa monolítico especificado na forma de fluxograma cujos nós já estão rotulados:



- correspondente programa com instruções rotuladas compostas é representado abaixo supondo que 1 é o rótulo inicial.

```
1: (G, 2), (F, 3)
2: (G, 2), (F, 3)
3: (F, 4), (G, 5)
4: (F, 4), (G, 5)
5: (F, 6), (ciclo, ω)
6: (parada, ε), (G, 7)
7: (G, 7), (G, 7)
ω: (ciclo, ω), (ciclo, ω)
```

Note-se que:

- o rótulo 2 é sucessor dele mesmo. O mesmo ocorre com os rótulos 4, 7 e ω;
- existem dois caminhos no fluxograma que atingem o nó parada. Só um é representado no conjunto de instruções rotuladas compostas.
- na instrução rotulada por 7, ocorre um ciclo infinito.

2.5.3 Equivalência Forte de Programas Monolíticos

- A *união disjunta* de conjuntos garante que todos os elementos dos conjuntos componentes constituem o conjunto resultante, mesmo que possuam a mesma identificação.
- considera-se que os elementos são distintos, mesmo que possuam a mesma identificação.
- Exemplo: para os conjuntos $A = \{a, x\}$ e $B = \{b, x\}$, o conjunto resultante da união disjunta é: $\{a_A, x_A, b_B, x_B\} = \{a, x_A, b, x_B\}$

Corolário 2.32 Equivalência Forte: União Disjunta.
Sejam:

- $Q = (I_Q, q)$ e $R = (I_R, r)$ dois programa monolíticos especificados usando instruções rotuladas compostas
- $P_q = (I, q)$ e $P_r = (I, r)$ programas monolíticos onde I é o conjunto resultante da união disjunta de I_Q e I_R.
- Então: $P_q \equiv P_r$ se, e somente se, $Q \equiv R$

O algoritmo para verificação da equivalência forte de Q e R resume-se à verificação se P_q e P_r são equivalentes fortemente.

- *cadeia de conjunto*: seqüência de conjuntos ordenada pela relação de inclusão;
- *programa monolítico simplificado*: instruções rotuladas compostas que determinam ciclos infinitos são excluídas (excetuando-se a instrução rotulada por ω, se existir). A simplificação baseia-se em cadeia de conjuntos;
- *rótulos equivalentes fortemente*: o algoritmo de verificação se P_q e P_r são equivalentes fortemente baseia-se em rótulos equivalentes fortemente de programas simplificados.

Definição 2.33 Cadeia de Conjuntos, Cadeia Finita de Conjuntos, Limite de uma Cadeia Finita de Conjuntos.

Uma seqüência de conjuntos $A_0A_1...$ é dita:

a) uma *Cadeia de Conjuntos* se, para qualquer $k \geq 0$: $A_k \subseteq A_{k+1}$

b) uma *Cadeia Finita de Conjuntos* é uma cadeia de conjuntos onde existe n, para todo $k \geq 0$, tal que: $A_n = A_{n+k}$

c) o *Limite da Cadeia Finita de Conjuntos* é: $lim\ A_k = A_n$

Lema 2.34 Identificação de Ciclos Infinitos em Programa Monolítico.

- *lema fornece um algoritmo para determinar se existem ciclos infinitos em um conjunto de instruções rotuladas compostas.*
- *A idéia básica é partir da instrução parada, rotulada por ε, determinando os seus antecessores.*
- *Por exclusão, um instrução que não é antecessor da parada determina um ciclo infinito.*

➢ Seja I um conjunto de n instruções rotuladas compostas.

➢ Seja $A_0A_1...$ uma seqüência de conjuntos de rótulos indutivamente definida como segue:

$A_0 = \{\varepsilon\}$

$A_{k+1} = A_k \cup \{r \mid r$ é rótulo de instrução antecessora de alguma instrução rotulada por $A_k\}$

➢ Então $A_0A_1...$ é uma cadeia finita de conjuntos, e, para qualquer rótulo r de instrução de I, tem-se que

$(I, r) \equiv (I, \omega)$ se, e somente se, $r \notin lim\ A_k$

O Lema acima proporciona uma maneira fácil de determinar se algum rótulo caracteriza ciclos infinitos.

EXEMPLO 2.19 Identificação de Ciclos Infinitos em Programa Monolítico.

- Considere o conjunto I de instruções rotuladas compostas representado na Figura abaixo

```
1: (G, 2), (F, 3)
2: (G, 2), (F, 3)
3: (F, 4), (G, 5)
4: (F, 4), (G, 5)
5: (F, 6), (ciclo, ω)
6: (parada, ε), (G, 7)
7: (G, 7), (G, 7)
ω: (ciclo, ω), (ciclo, ω)
```

- A correspondente cadeia finita de conjuntos é:

$A_0 = \{\varepsilon\}$

$A_1 = \{6, \varepsilon\}$

$A_2 = \{5, 6, \varepsilon\}$

$A_3 = \{3, 4, 5, 6, \varepsilon\}$

$A_4 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

$A_5 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

- Logo: $lim\ A_k = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

- Simplificação de ciclos infinitos: $(I, 7) \equiv (I, \omega)$, pois $7 \notin lim\ A_k$.

Portanto, pode-se simplificar um conjunto de instruções rotuladas compostas eliminando-se qualquer instrução de rótulo $r \neq \omega$ que determine um ciclo infinito.

Definição 2.35 Algoritmo de Simplificação de Ciclos Infinitos.
Seja I um conjunto finito de instruções rotulas compostas.
O *Algoritmo de Simplificação de Ciclos Infinitos* é:

a) determina-se a correspondente cadeia finita de conjuntos $A_0A_1...$ como no lema 2.34;

b) para qualquer rótulo r de instrução de I tal que $r \notin lim\ A_k$, tem-se que:
   - a instrução rotulada por r é excluída;
   - toda referência a pares da forma (F, r) em I é substituída por (ciclo, ω);
   - $I = I \cup \{\omega: (ciclo, \omega), (ciclo, \omega)\}$.

EXEMPLO 2.20:.

```
1: (G, 2), (F, 3)
2: (G, 2), (F, 3)
3: (F, 4), (G, 5)
4: (F, 4), (G, 5)
5: (F, 6), (ciclo, ω)
6: (parada, ε), (ciclo, ω)
ω: (ciclo, ω), (ciclo, ω)
```

Lema 2.36 Rótulos Consistentes.

- Seja I um conjunto finito de instruções rotuladas compostas e simplificadas.
- Sejam r e s dois rótulos de instruções de I, ambos diferentes de ε.
- Suponha que as instruções rotuladas por r e s são da seguinte forma, respectivamente:

  r: $(F_1, r_1), (F_2, r_2)$

  s: $(G_1, s_1), (G_2, s_2)$

  Então:

  r e s são consistentes se, e somente se, $F_1 = G_1$ e $F_2 = G_2$

Definição 2.37 Rótulos Equivalentes Fortemente.

- Seja I um conjunto finito de instruções rotuladas compostas e simplificadas.
- Sejam r e s dois rótulos de instruções de I.
- Suponha que as instruções rotuladas por r e s são da seguinte forma, respectivamente:

$r: (F_1, r_1), (F_2, r_2)$
$s: (G_1, s_1), (G_2, s_2)$

Então, r e s são *Rótulos Equivalentes Fortemente* se, e somente se:

- ou $r = s = \varepsilon$;
- ou r e s são ambos diferentes de ε e consistentes, ou seja

$$F_1 = G_1 \text{ e } F_2 = G_2$$

Teorema 2.38 Determinação de Rótulos Equivalentes Fortemente.

- Seja I um conjunto de n instruções compostas e simplificadas.
- Sejam r e s dois rótulos de instruções de I.
- Define-se, indutivamente, a seqüência de conjuntos $B_0B_1...$ por:

$B_0 = \{(r, s)\}$
$B_{k+1} = \{(r'', s'') \mid r''$ e s'' são rótulos sucessores de r' e s', respectivamente, $(r', s') \in B_k$ e $(r'', s'') \notin B_i$, $(0 \leq i \leq k)\}$

- Então $B_0B_1...$ é uma seqüência que converge para o conjunto vazio, e r, s são rótulos equivalentes fortemente se, e somente se, qualquer par de B_k é constituído por rótulos consistentes.

Definição 2.39
Algoritmo de Verificação da Equivalência Forte de Programas Monolíticos.

- Sejam $Q = (I_Q, q)$ e $R = (I_R, r)$ dois programas monolíticos especificados usando instruções rotuladas compostas e simplificado.
- *Algoritmo de Verificação da Equivalência Forte de Programas Monolíticos* Q *e* R é determinado pelos passos:

Passo 1. Sejam $P_q = (I, q)$ e $P_r = (I, r)$ programas monolíticos onde I é o conjunto resultante da união disjunta de I_Q e I_R, excetuando-se a instrução rotulada ω, se existir, a qual ocorre, no máximo, uma vez em I.

Passo 2. Se q e r são rótulos equivalentes fortemente, então $B_0=\{(q, r)\}$. Caso contrário, Q e R *não são equivalentes fortemente,* e o algoritmo termina.

Passo 3. Para $k \geq 0$, define-se o conjunto B_{k+1}, contendo somente os pares (q'', r'') de rótulos sucessores de cada $(q', r') \in B_k$, tais que:

- $q' \neq r'$;
- q' e r' são ambos diferentes de ε;
- os pares sucessores (q'', r'') não são elementos de B_0, B_1, ... , B_k.

Passo 4. Dependendo de B_{k+1}, tem-se que:

a) $B_{k+1} = \varnothing$: Q *e* R ***são*** *equivalentes fortemente,* e o algoritmo termina;

b) $B_{k+1} \neq \varnothing$: se todos os pares de rótulos de B_{k+1} são equivalentes fortemente, então vá para o *Passo 3*; caso contrário, Q e R ***não são*** *equivalentes fortemente,* e o algoritmo termina.

EXEMPLO 2.21 Algoritmo de Verificação da Equivalência Forte de Programas Monolíticos.

Considere os programas monolíticos especificados na forma de fluxograma Q (*Exemplo 2.18 – Figura 2.25*) e R (*dado abaixo*):

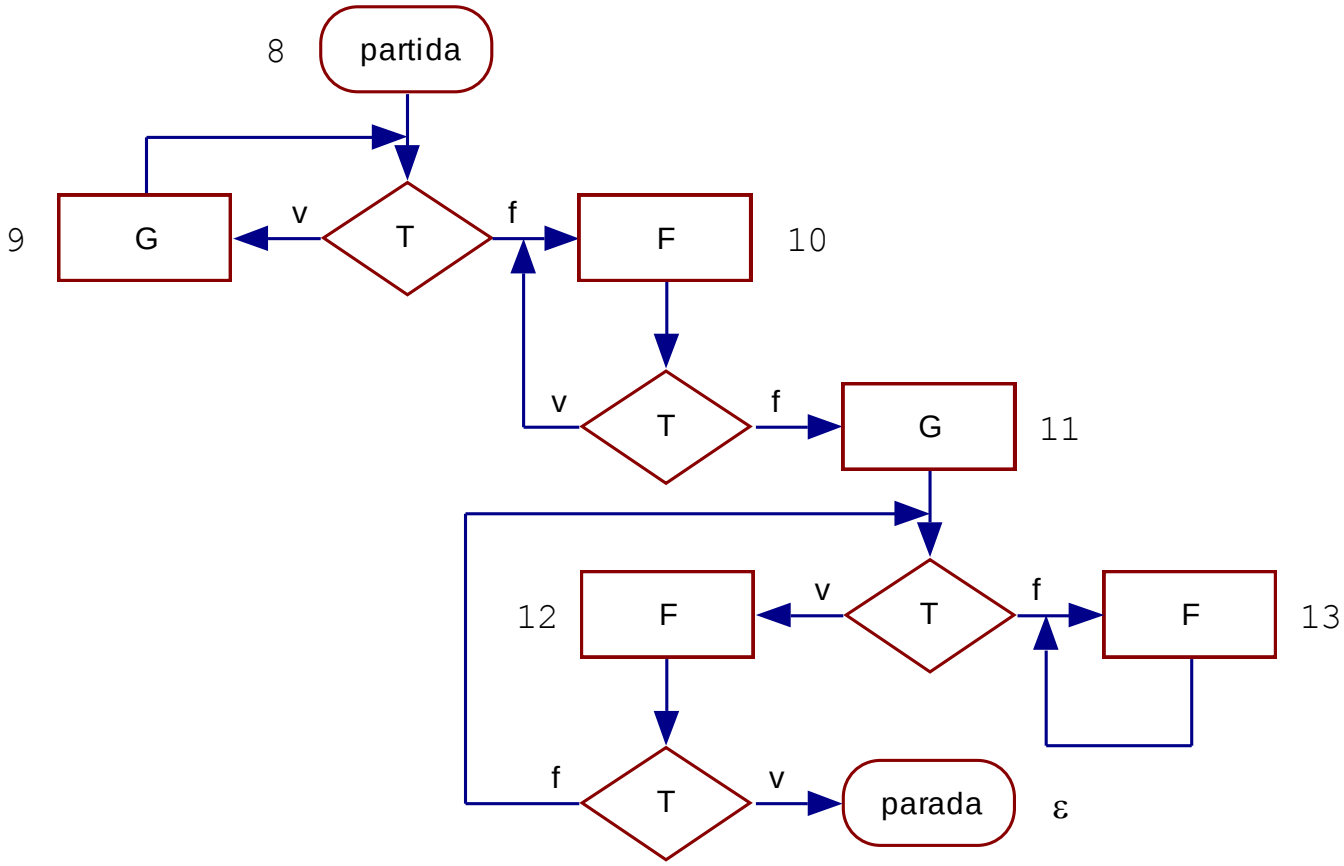


Figura 2.27 - Programa Monolítico R

a) A especificação do programa Q usando instruções rotuladas compostas, já simplificado, já foi feita na Figura 2.26.

```
1: (G, 2), (F, 3)
2: (G, 2), (F, 3)
3: (F, 4), (G, 5)
4: (F, 4), (G, 5)
5: (F, 6), (ciclo, ω)
6: (parada, ε), (ciclo, ω)
ω: (ciclo, ω), (ciclo, ω)
```

b) Em relação ao programa R, tem-se que:

b.1) ***Conjunto de instruções rotuladas compostas.***

```
8:      (G, 9), (F, 10)
9:      (G, 9), (F, 10)
10:     (F, 10), (G, 11)
11:     (F, 12), (F, 13)
12:     (parada, ε), (F, 13)
13:     (F, 13), (F, 13)
```

b.2) Identificação de ciclos infinitos.

$A_0 = \{\varepsilon\}$

$A_1 = \{12, \varepsilon\}$
$A_2 = \{11, 12, \varepsilon\}$
$A_3 = \{10, 11, 12, \varepsilon\}$
$A_4 = \{8, 9, 10, 11, 12, \varepsilon\}$
$A_5 = \{8, 9, 10, 11, 12, \varepsilon\}$

Portanto:

$lim\ A_k = \{8, 9, 10, 11, 12, \varepsilon\}$

$(I_R, 13) \equiv (I, \omega)$, pois $13 \notin lim\ A_k$.

b.3) Simplificação de ciclos infinitos.

```
8:      (G, 9), (F, 10)
9:      (G, 9), (F, 10)
10:     (F, 10), (G, 11)
11:     (F, 12), (ciclo, ω)
12:     (parada, ε), (ciclo, ω)
ω:      (ciclo, ω), (ciclo, ω)
```

c) Relativamente à aplicação do algoritmo, tem-se que:

Passo 1. Seja I a união disjunta dos conjuntos I_Q e I_R, excetuando-se a instrução rotulada ω, como segue:

```
1: (G, 2), (F, 3)
2: (G, 2), (F, 3)
3: (F, 4), (G, 5)
4: (F, 4), (G, 5)
5: (F, 6), (ciclo, ω)
6: (parada, ε), (ciclo, ω)
8: (G, 9), (F, 10)
9: (G, 9), (F, 10)
10:(F, 10), (G, 11)
11:(F, 12), (ciclo, ω)
12:(parada, ε), (ciclo, ω)
ω: (ciclo, ω), (ciclo, ω)
```

Para verificar se $Q \equiv R$, é suficiente verificar se $(I, 1) \equiv (I, 8)$.

Passo 2. Como 1 e 8 são rótulos equivalentes fortemente, então:

$$B_0 = \{(1, 8)\}$$

Passos 3 e 4. Para $k \geq 0$, construção de B_{k+1} é como segue:

$B_1 = \{(2, 9), (3, 10)\}$ pares de rótulos equivalentes fortemente

$B_2 = \{(4, 10), (5, 11)\}$ pares de rótulos equivalentes fortemente

$B_3 = \{(6, 12), (\omega, \omega)\}$ pares de rótulos equivalentes fortemente

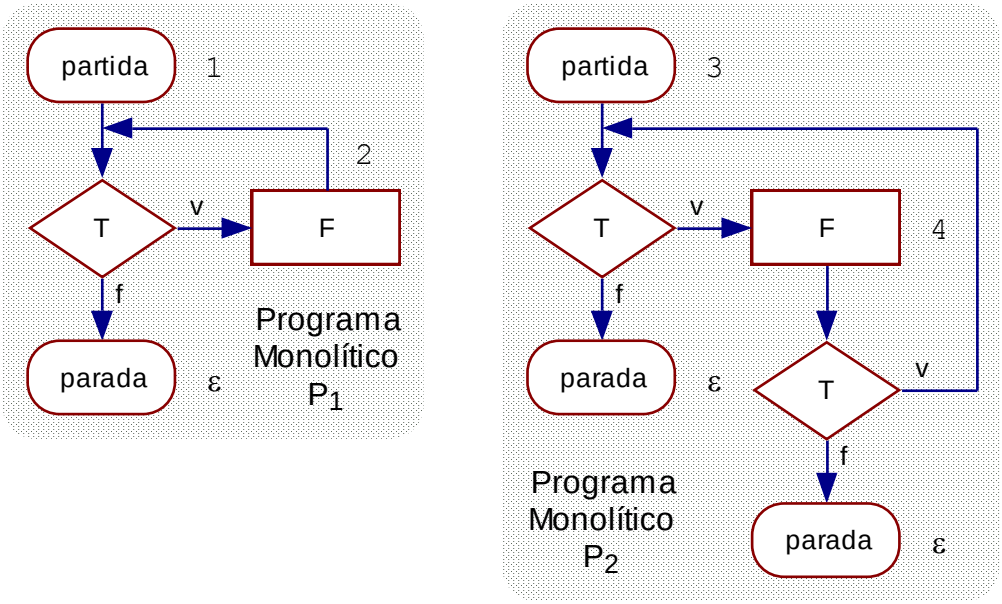
$B_4 = \{(\varepsilon, \varepsilon)\}$ pares de rótulos equivalentes fortemente

$B_5 = \varnothing$

Logo $(I, 1) \equiv (I, 8)$, e, portanto, $Q \equiv R$. □

EXEMPLO 2.22 Algoritmo de Verificação da Equivalência Forte de Programas Monolíticos.

- Os programas monolíticos (fluxogramas) reproduzidos na Figura 2.28 com os nós rotulados são equivalentes fortemente.



a) A especificação do programa P_1 usando instruções rotuladas compostas (e já simplificadas) é:

```
1: (F, 2), (parada, ε)
2: (F, 2), (parada, ε)
```

b) A especificação de P_2 usando instruções rotuladas compostas (e já simplificadas) é:

```
3: (F, 4), (parada, ε)
4: (F, 4), (parada, ε)
```

- os correspondentes conjuntos de instruções rotuladas compostas são iguais, a menos dos rótulos.
- aplicação do algoritmo: é fácil verificar que (I, 1) ≡ (I, 3).
- a relação equivalência forte fornece subsídios para analisar a complexidade estrutural de programas.
- No caso, P_1 é estruturalmente "mais otimizado" que P_2.

2.6 Conclusão

- 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
- Mostrou-se a existência de um algoritmo para verificar se programas monolíticos (ou iterativos) são fortemente equivalentes.

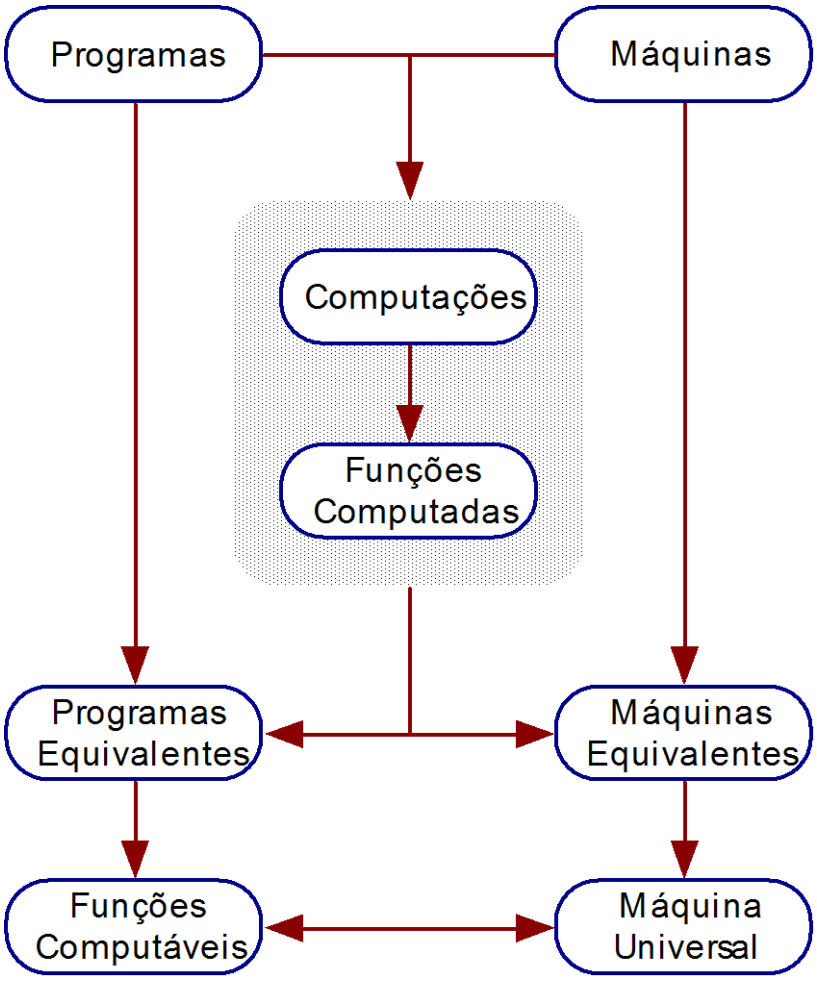


Figura 2.29 Relação entre os conceitos