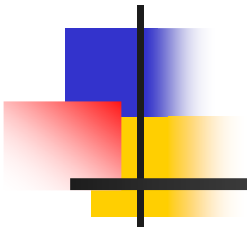
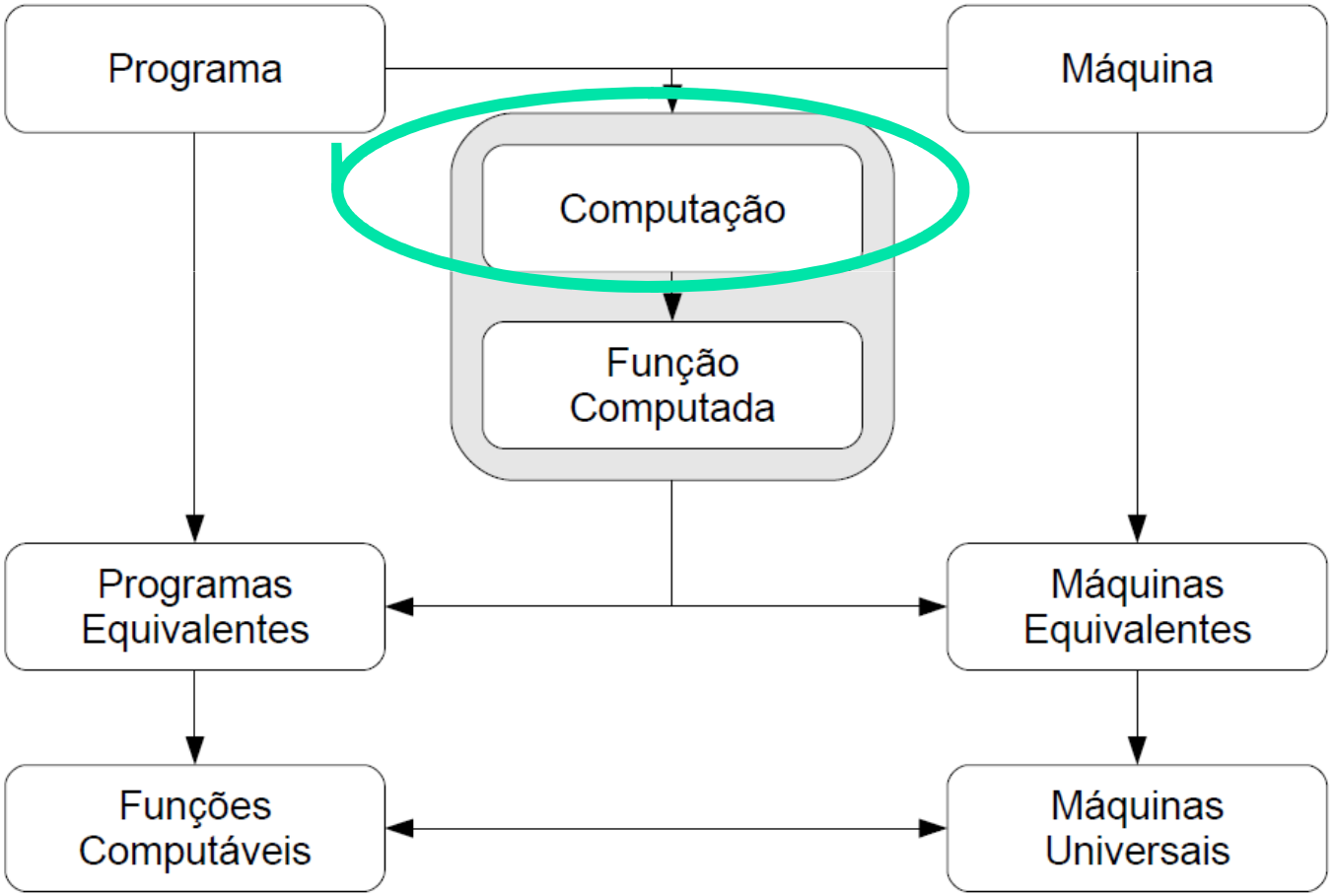


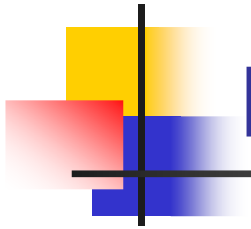
Teoria da Computação

Unidade 2 – Programas, máquinas e computação:
Equivalência de programas

Referência – Teoria da Computação (Divério, 2000)

Programas, Máquinas e Computações



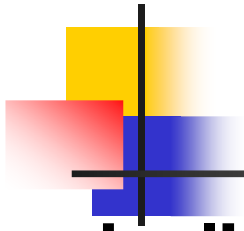


Programas, Máquinas e Computações

- Exercício - Fazer a computação e verificar se a computação é finita ou infinita
- programa Recursivo duplica

  duplica é R onde

  R def (se **zero** então √ senão **sub**; R; **ad**; **ad**)



Programas, Máquinas e Computações

duplica é R onde

R def (se **zero** então √ senão **sub**; R; **ad**; **ad**)

Computação de **duplica**

(R, √, 2)

(se **zero** então √ senão (**sub**; R; **ad**; **ad**); √; 2)

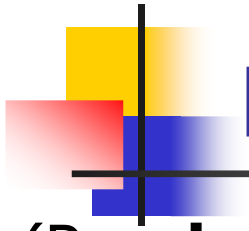
(**sub**; R; **ad**; **ad**); √; 2)

(R; **ad**; **ad**); √; 1)

(se **zero** então √ senão (**sub**; R; **ad**; **ad**); **ad**; **ad** √; 1)

(**sub**; R; **ad**; **ad**; **ad**; **ad** √; 1)

(R; **ad**; **ad**; **ad**; **ad** √; 0) ...



Programas, Máquinas e Computações

(R; **ad**; **ad**; **ad**; **ad** √; 0)

(se **zero** então √ senão (**sub**; R; **ad**; **ad**); **ad**; **ad**; **ad**; **ad** √; 0)

(√; **ad**; **ad**; **ad**; **ad** √; 0)

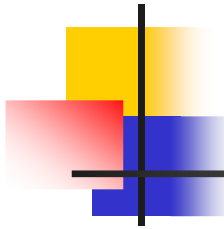
(**ad**; **ad**; **ad** √; 1)

(**ad**; **ad** √; 2)

(**ad** √; 3)

(√; 4)

A computação é finita



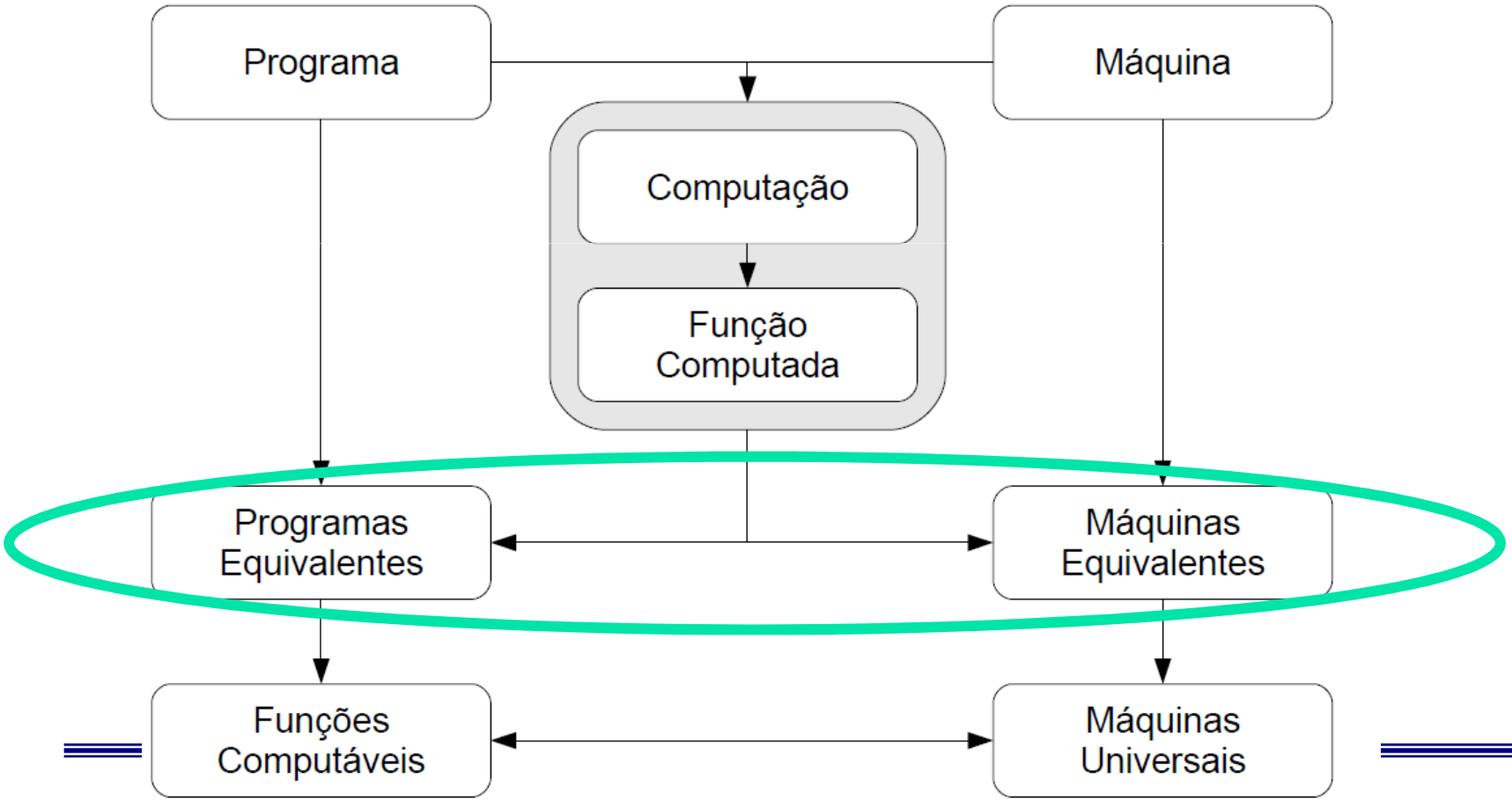
Conclusão

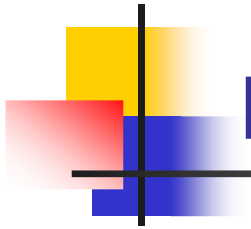
- A partir do conceito de função computada são apresentadas noções de equivalências de programas e máquinas: programas equivalentes fortemente e programas equivalentes em uma máquina;

- É apresentado um algoritmo para verificar se programas monolíticos (ou iterativos) são equivalentes fortemente. Até o momento não existe um algoritmo para programas recursivos;

- Posteriormente, será apresentado o conceito de máquina universal.

Programas, Máquinas e Computações



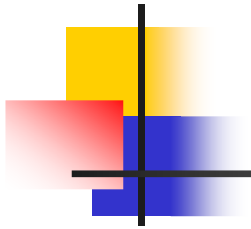


Equivalência de Programas e Máquinas

1. **Relação Equivalência Forte de Programas:** Um par de programas pertence a relação se as correspondentes funções computadas coincidem para *qualquer* máquina

2. **Relação Equivalência de Programas em uma Máquina:** Um par de programas pertence a relação se as correspondentes funções computadas coincidem para uma *dada* máquina

3. **Relação Equivalência de Máquinas:** Um par de máquinas pertence a relação se as máquinas podem se simular mutuamente. A simulação de uma máquina por outra pode ser feita usando programas diferentes.

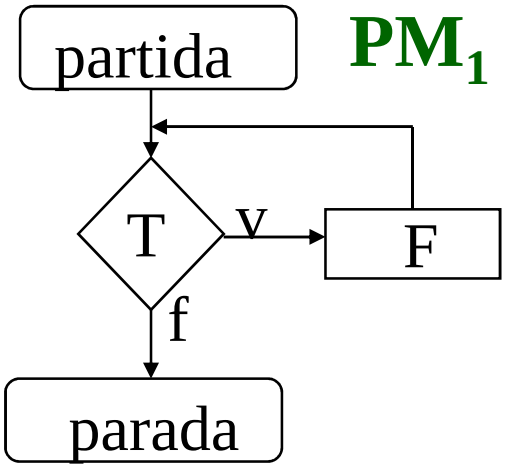


Equivalência Forte de Programas

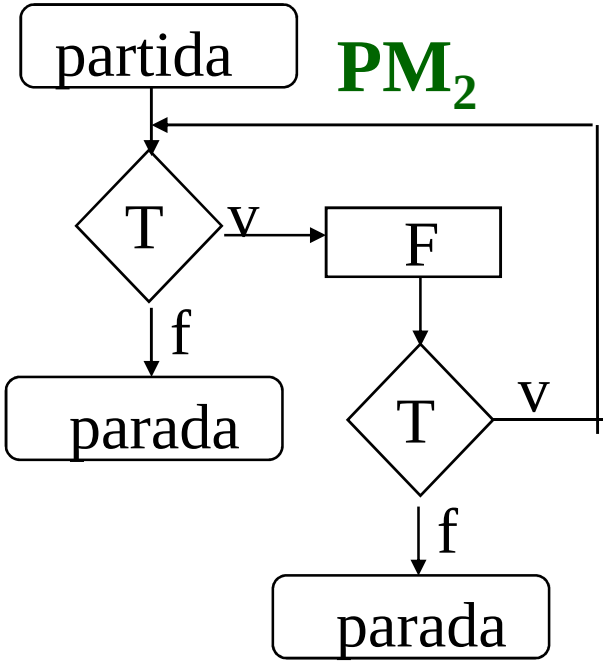
1. Relação Equivalência Forte de Programas, Programas Equivalentes Fortemente
   - Sejam P e Q dois programas. Então o par (P, Q) está na *Relação Equivalência Forte de Programas*, denotado por: P ≡ Q
     - Se, e somente se, para qualquer máquina M as correspondentes funções computadas são iguais, ou seja:
       - <P, M> = <Q, M>
   - Neste caso, P e Q são ditos *Programas Equivalentes Fortemente*

Equivalência Forte de Programas

1. **Programas Equivalentes Fortemente:**



PM_1



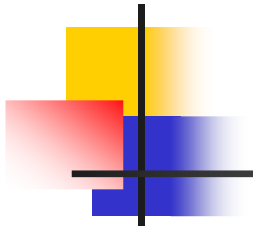
PM_2

PI_3

enquanto T
faça (F)

PR_4

P_4 é R onde
R def (se T então F;R senão√)



Equivalência Forte de Programas

- **É importante considerar a relação Equivalência Forte de Programas por várias razões, como:**

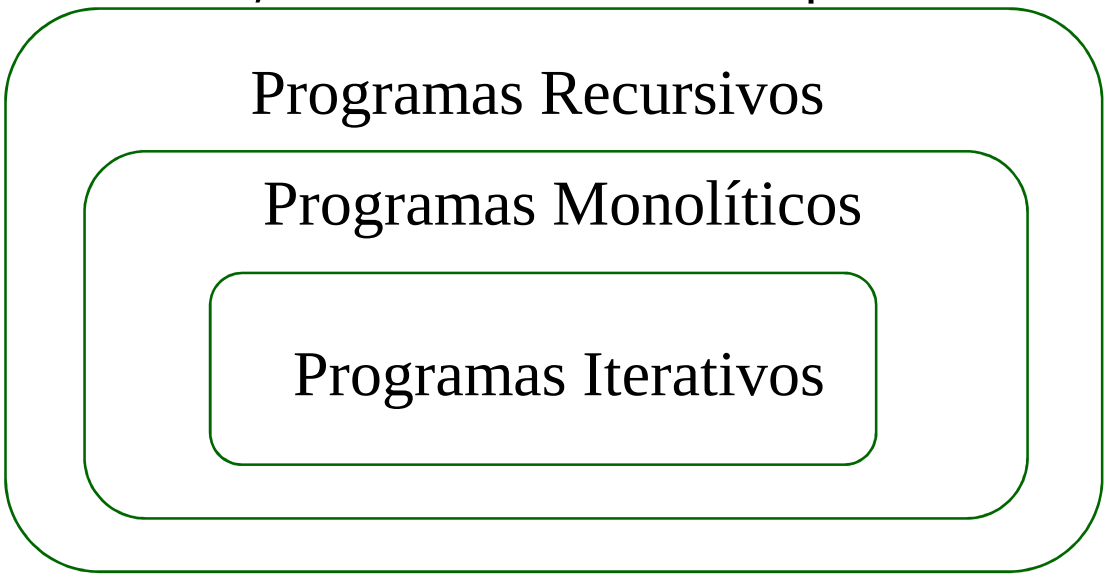
  - Permite identificar diferentes programas em uma mesma classe de equivalência, ou seja, identificar diferentes programas cujas funções computadas coincidem, para qualquer máquina;

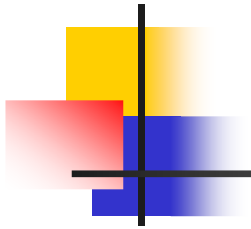
  - *As funções computadas por programas equivalentes fortemente têm a propriedade de que os mesmos testes e as mesmas operações são efetuados na mesma ordem, independentemente do significado dos mesmos*

  - Fornece subsídios para analisar a complexidade estrutural de programas. Por exemplo, analisando os programas monolíticos equivalentes fortemente PM_1 e PM_2, pode-se concluir que PM_1 é estruturalmente *mais otimizado* que PM_2, pois contém um teste a menos.

Equivalência Forte de Programas

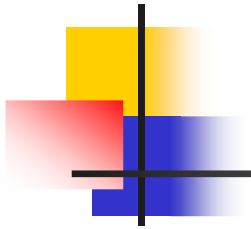
- Verifica-se que:
  - Para todo iterativo, existe um monolítico equivalente fortemente
  - Para todo monolítico, existe um recursivo equivalente fortemente
  - Para todo iterativo, existe um recursivo equivalente fortemente





Equivalência Forte de Programas

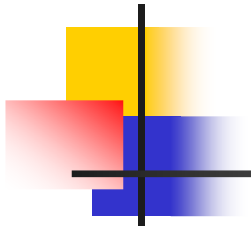
- **Equivalência Forte de Programas**
  - **Iterativo $\rightarrow$ Monolítico**
  - **Monolítico $\rightarrow$ Recursivo**
  - **Iterativo $\rightarrow$ Recursivo**
  - **Recursivo $\neq$ Monolítico**
  - **Monolítico $\neq$ Iterativo**



Equivalência Forte de Programas

- **Equivalência Forte de Programas**
  - **Iterativo $\rightarrow$ Monolítico**
  - **Monolítico $\rightarrow$ Recursivo**
  - **Iterativo $\rightarrow$ Recursivo**

  - **Recursivo $\neq$ Monolítico**
  - **Monolítico $\neq$ Iterativo**



Equivalência Forte de Programas

- Iterativo → Monolítico

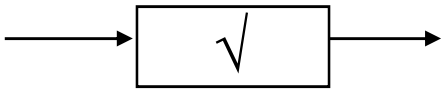
Justificativa: a obtenção de um programa monolítico a partir de um iterativo é direta, a partir do mapeamento das construções elementares de um programa iterativo em sequências de construções equivalentes em um programa monolítico. Como as mesmas operações são executadas na mesma ordem em ambos os programas, as funções computadas são idênticas

Equivalência Forte de Programas

- Iterativo → Monolítico

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

a) Para a operação vazia corresponde ao fluxograma elementar:



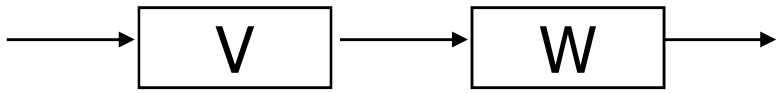
b) Para cada identificador de operação F de P_i

Equivalência Forte de Programas

- Iterativo → Monolítico

c) Suponha que T é um identificador de teste e que V, W são programas iterativos usados na construção de P_i. Então, para cada um dos seguintes tipos de composição é apresentado o correspondente fluxograma de P_m.

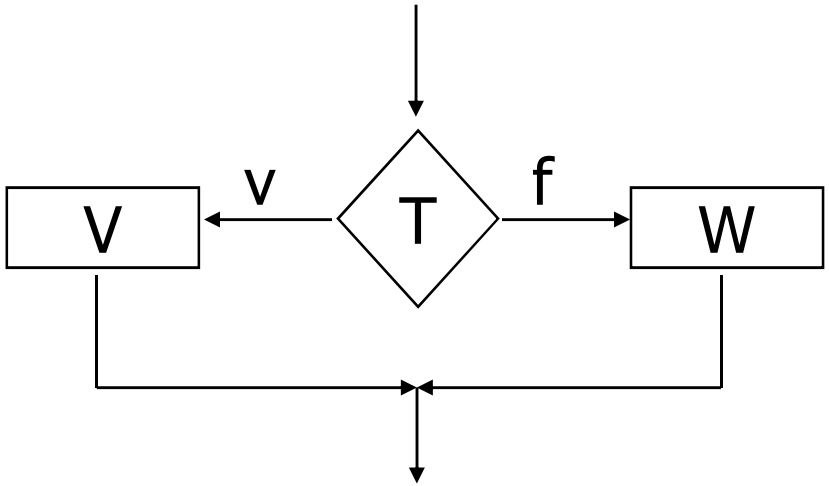
c.1) Composição sequencial. V;W



Equivalência Forte de Programas

- Iterativo → Monolítico

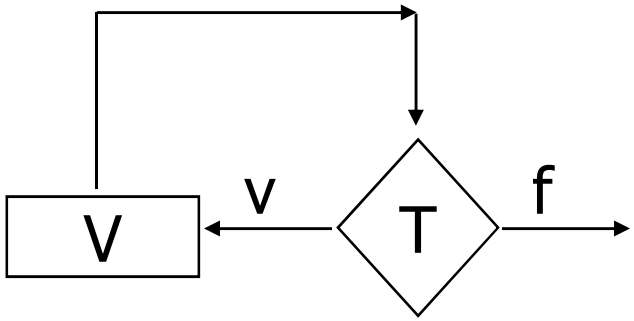
c.2) Composição condicional. (Se T então V senão W)



Equivalência Forte de Programas

- Iterativo → Monolítico

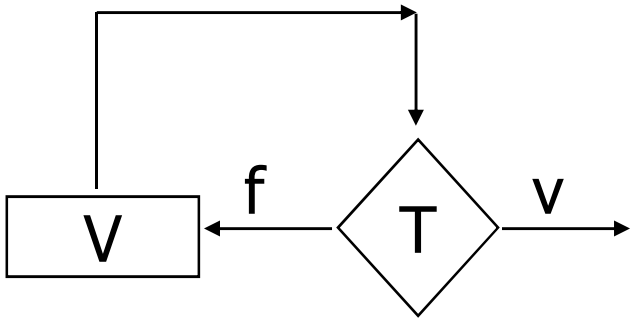
c.3) Composição enquanto. (Enquanto T faça (V))

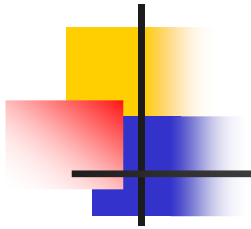


Equivalência Forte de Programas

- Iterativo → Monolítico

c.4) Composição até. (Até T faça (V))





Equivalência Forte de Programas

- Iterativo → Monolítico

Exemplo: faça o mapeamento do programa iterativo para um programa monolítico

até a_zero faça (subtrai_a; adiciona_b)

Iterativo

até a_zero faça (subtrai_a; adiciona_b)

Equivalência Forte de Programas

- Iterativo → Monolítico

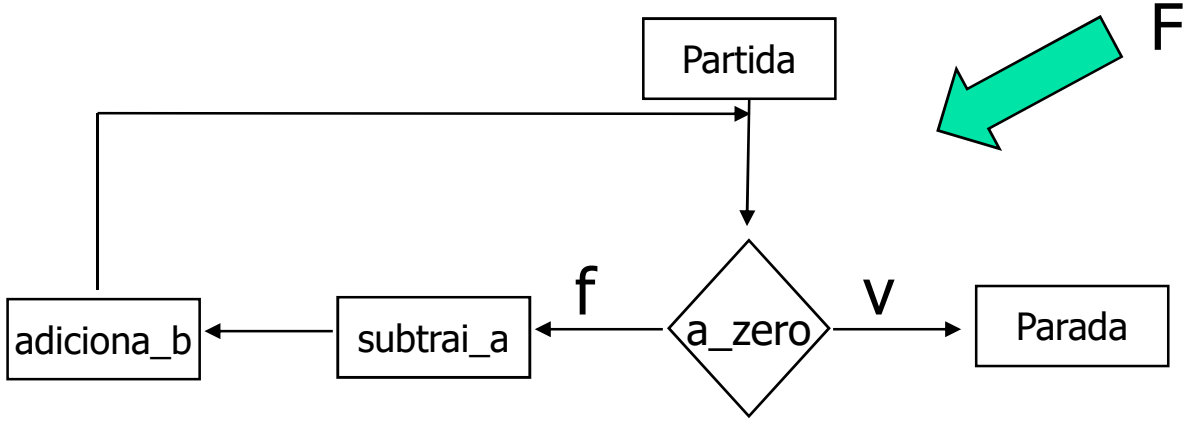
1: se a_zero então vá_para 4 senão vá_para 2

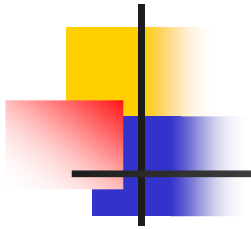
2: faça subtrai_a vá_para 3

3: faça adiciona_b vá_para 1

Instruções rotuladas

Fluxograma

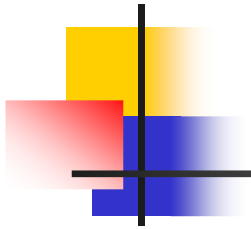




Equivalência Forte de Programas

- **Equivalência Forte de Programas**
  - **Iterativo $\rightarrow$ Monolítico**
  - **Monolítico $\rightarrow$ Recursivo**
  - **Iterativo $\rightarrow$ Recursivo**

  - **Recursivo $\neq$ Monolítico**
  - **Monolítico $\neq$ Iterativo**



Equivalência Forte de Programas

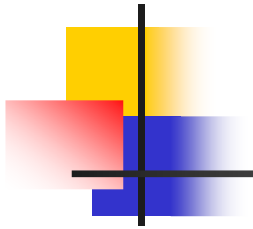
- Monolítico → Recursivo

Para qualquer programa monolítico P_m, existe um programa recursivo P_r, tal que $P_m \equiv P_r$

- Seja P_m um programa monolítico qualquer onde $L = \{r_1, r_2, \ldots, r_n\}$ é o correspondente conjunto de rótulos. Suponha que de 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:

$$P_r \text{ é } R_1 \text{ onde } R_1 \text{ def } E_1,\ R_2 \text{ def } E_2, \ldots,\ R_n \text{ def } \surd$$

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



Equivalência Forte de Programas

- Monolítico → Recursivo

a) *Operação*. Se r_k é da forma:

r_k: faça F vá_para r_k'

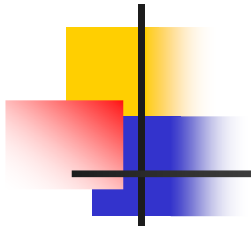
então E_k é a seguinte expressão de sub-rotinas: $F;R_k'$

b) *Teste*. Se r_k é da forma:

r_k: se T então vá_para r_k' senão vá_para r_k''

então E_k é a seguinte expressão de sub-rotinas:

(se T então R_k' senão R_k'')



Equivalência Forte de Programas

- Monolítico → Recursivo

Exemplo: faça o mapeamento do programa monolítico (instruções rotuladas) para recursivo

1: se a_zero então vá_para 4 senão vá_para 2

2: faça subtrai_a vá_para 3

3: faça adiciona_b vá_para 1

Equivalência Forte de Programas

- Monolítico → Recursivo

Exemplo: faça o mapeamento do programa monolítico para recursivo

R é R_1 onde

R_1 def (se a_zero então R_4 senão R_2)

R_2 def (faça subtrai_a; R_3)

R_3 def (faça adiciona_b; R_1)

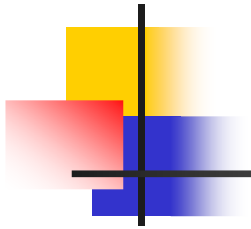
R_4 def $\sqrt{}$

Monolítico

1: se a_zero então vá_para 4 senão vá_para 2

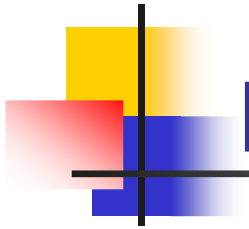
2: faça subtrai_a vá_para 3

3: faça adiciona_b vá_para 1



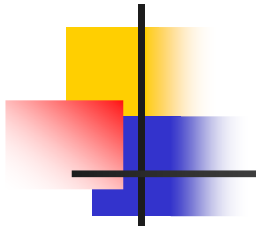
Equivalência Forte de Programas

- **Equivalência Forte de Programas**
  - **Iterativo $\rightarrow$ Monolítico**
  - **Monolítico $\rightarrow$ Recursivo**
  - **Iterativo $\rightarrow$ Recursivo**
  - **Recursivo $\neq$ Monolítico**
  - **Monolítico $\neq$ Iterativo**



Programas, Máquinas e Computações

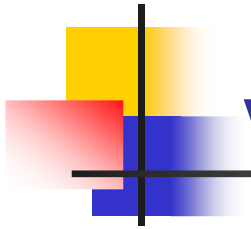
- Exercícios:
  - 2.12 ao 2.14
    - Enviar por email (celso.olivete@unesp.br) até 12/03



Equivalência Forte de Programas

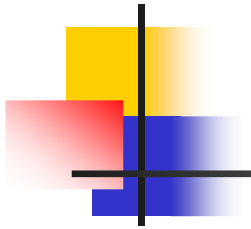
Recursivo ≠ Monolítico

- Dado um programa recursivo qualquer, não necessariamente existe um programa monolítico
- 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;



Verificação de Equivalência Forte de Programas

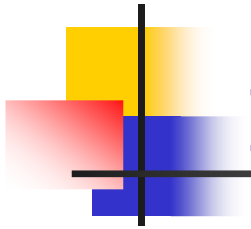
- **A verificação de que dois programas monolíticos são equivalentes fortemente usa os seguintes conceitos:**
  - *Máquina de Traços*: Produz um rastro ou histórico (denominado *traço*) da ocorrência das operações do programa. Neste contexto, dois programas são equivalente fortemente se são equivalentes em qualquer máquina de traços.
  - *Programa Monolítico com Instruções Rotuladas Compostas*: Instruções rotuladas compostas constituem uma forma alternativa de definir programas monolíticos. Ex.:
    - r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3



Instruções Rotuladas Compostas

- A verificação da equivalência forte de dois programas monolíticos pode ser realizada usando uma representação baseada em conjuntos de instruções rotuladas compostas.

- Instruções rotuladas compostas possuem somente uma única forma, ao contrário das instruções rotuladas, as quais podem ser de duas formas: operação ou teste. De fato, uma instrução rotulada composta combina ambas em uma única forma.

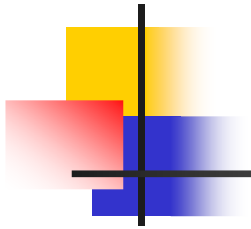


Instrução Rotulada Composta

- É uma sequência de símbolos da seguinte forma:

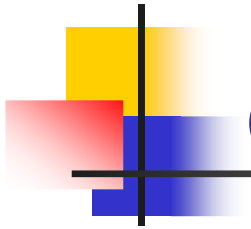
- r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3

- Adicionalmente:
- r_2 e r_3 são ditos *rótulos sucessores* de r_1
- r_1 é dito *rótulo antecessor* de r_2 e r_3



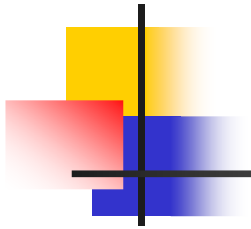
Programa Monolítico com Instruções Rotuladas Compostas

- 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* o qual 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 *Rótulo Final.*



Programa Monolítico com Instruções Rotuladas Compostas

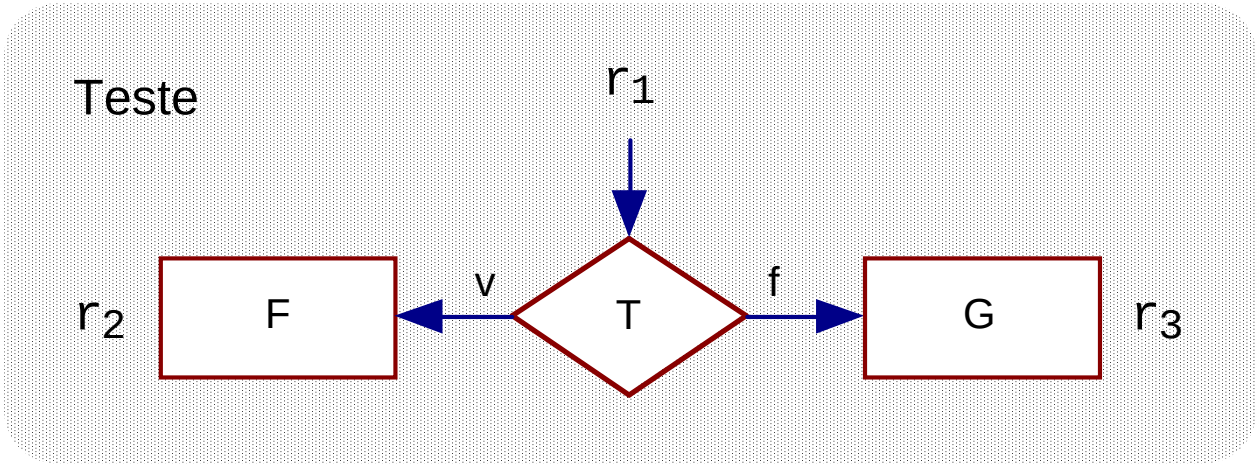
- Considerando um único identificador de teste, uma instrução rotulada composta da forma:
  - r_1: se T então faça F vá_para r_2 senão faça G vá_para r_3
- pode ser abreviada simplesmente por:
  - r_1: (F, r_2), (G,r_3)



Algoritmo: Fluxograma→Rotuladas Compostas

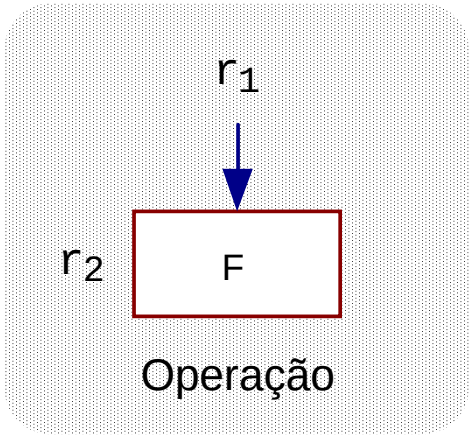
- *Rotulação de Nós.* Rotula-se cada nó do fluxograma. Existe uma parada associada ao identificador ε (palavra vazia). Partida é o Rótulo Inicial do programa P´.
- *Instruções Rotuladas Compostas.*
  - *Teste*: r_1: (F, r_2), (G, r_3)
  - *Operação:* r_1: (F, r_2), (F, r_2)
  - *Parada:* r: (parada,ε), (parada,ε)
  - *Testes Encadeados:* r_1: (F, r_2), (G, r_3)
  - *Testes Encadeados em Ciclo Infinito.* r_1: (F, r_2), (ciclo, ω)

Algoritmo: Fluxograma→Rotuladas Compostas



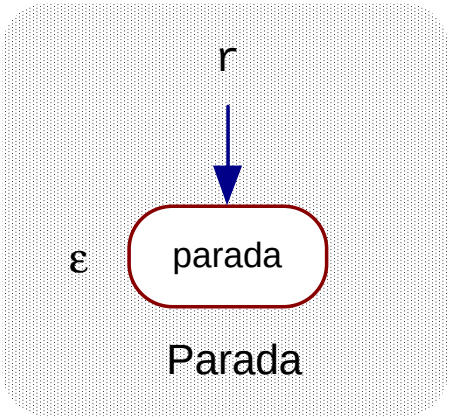
- *Teste*: r_1: (F, r_2), (G, r_3)

Algoritmo: Fluxograma→Rotuladas Compostas



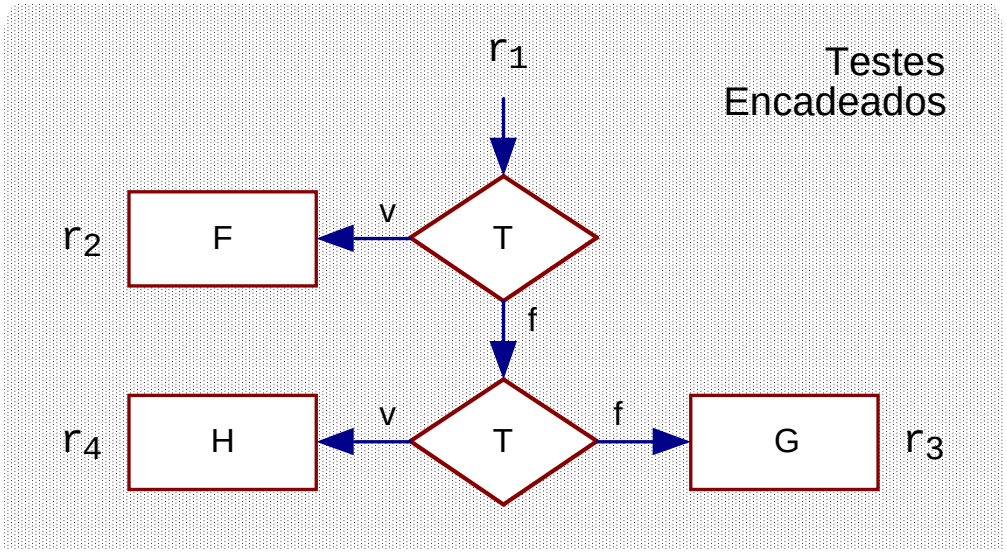
- *Operação:* r_1: (F, r_2), (F, r_2)

Algoritmo: Fluxograma→Rotuladas Compostas



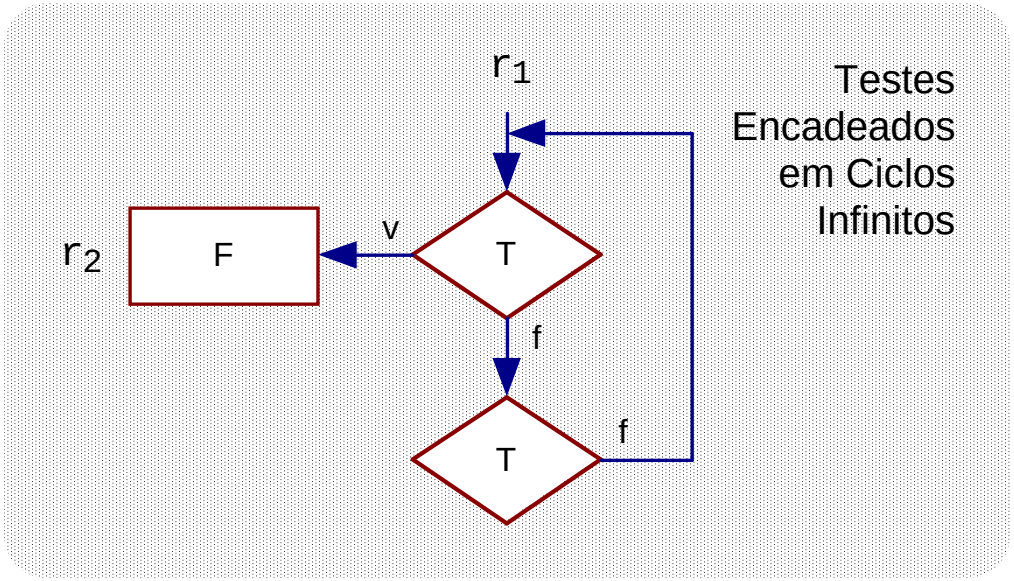
- *Parada:* r: (parada,ε), (parada,ε)

Algoritmo: Fluxograma→Rotuladas Compostas

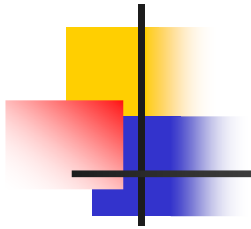


- *Testes Encadeados:* r_1: (F, r_2), (G, r_3)

Algoritmo: Fluxograma→Rotuladas Compostas

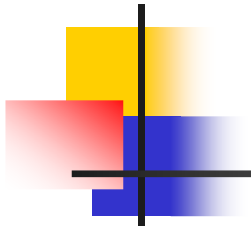


- *Testes Encadeados em Ciclo Infinito.* r_1: (F, r_2), (ciclo, ω)



Algoritmo de Equivalência Forte de Programas Monolíticos

- Passos para verificação:
  - a) Especificação do programa usando instruções rotuladas compostas;
  - b) Conjunto de instruções rotuladas compostas;
  - c) Identificação dos ciclos infinitos;
  - d) Simplificação dos ciclos infinitos;
  - e) União disjunta dos conjuntos I_Q e I_R, excetuando-se a instrução rotulada ω;
  - f) Verificar se $Q \equiv R$

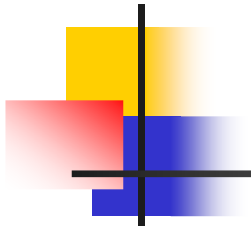


Equivalência Forte de Programas Monolíticos

a) Especificação do programa utilizando instruções rotuladas compostas

1. Representar o programa monolítico na forma de fluxograma
2. Atribuir rótulos numéricos para todas as operações
3. Atribuir o rótulo ε para a instrução "parada" → deve ser único
4. Considerar que os rótulos seguem os nós
5. A cada rótulo numérico *i*, criar *i: (F, i′) (G, i″)* se:
   - A condição verdadeira para o teste *T* implica na execução de *F*
   - Após a execução de *F* o próximo rótulo atingido é *i′*
   - A condição falsa para o teste *T* implica na execução de *G*
   - Após a execução de *G* o próximo rótulo atingido é *i″*

- Se a execução de uma operação levar a um loop infinito, deve-se usar (ciclo, *w*) e acrescentar a instrução rotulada composta *w: (ciclo, w), (ciclo, w)* ao programa



Equivalência Forte de Programas Monolíticos

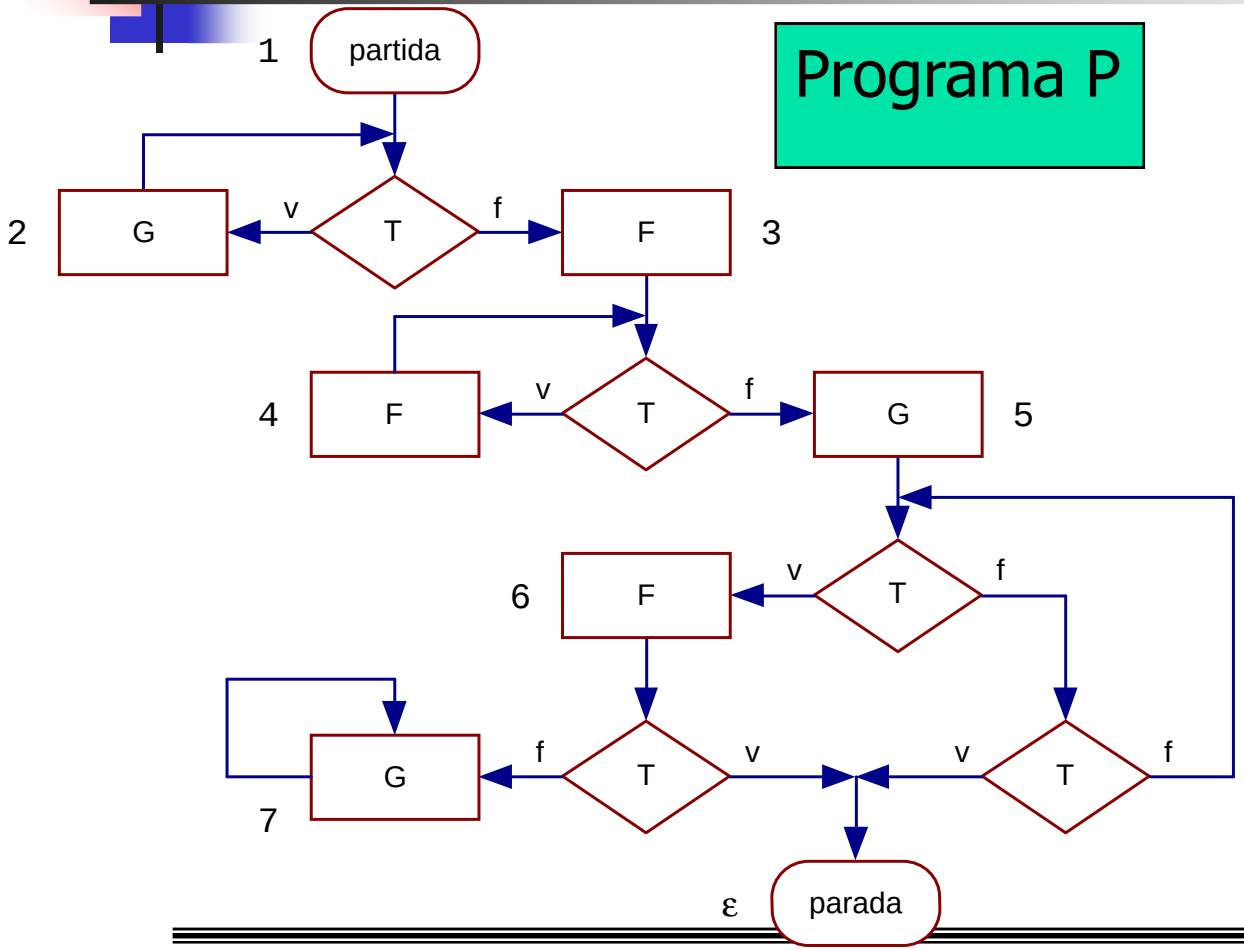
a) Especificação do programa utilizando instruções rotuladas compostas

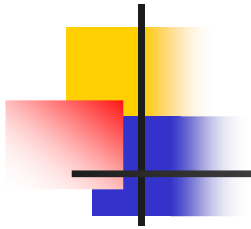
1. **Representar o programa monolítico na forma de fluxograma**
2. **Atribuir rótulos numéricos para todas as operações**
3. **Atribuir o rótulo ε para a instrução "parada" → deve ser único**
4. **Considerar que os rótulos seguem os nós**
5. A cada rótulo numérico *i*, criar *i: (F, i′) (G, i″)* se:
   - A condição verdadeira para o teste *T* implica na execução de *F*
   - Após a execução de *F* o próximo rótulo atingido é *i′*
   - A condição falsa para o teste *T* implica na execução de *G*
   - Após a execução de *G* o próximo rótulo atingido é *i″*

- Se a execução de uma operação levar a um loop infinito, deve-se usar (ciclo, *w*) e acrescentar a instrução rotulada composta *w: (ciclo, w), (ciclo, w)* ao programa

Equivalência Forte de Programas Monolíticos

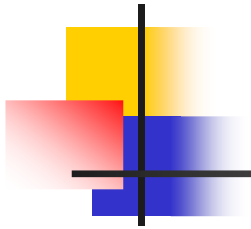
Exemplo: fluxograma rotulado e instruções rotuladas compostas





Algoritmo de Equivalência Forte de Programas Monolíticos

- Passos para verificação:
  - a) Especificação do programa usando instruções rotuladas compostas;
  - b) Conjunto de instruções rotuladas compostas;
  - c) Identificação dos ciclos infinitos;
  - d) Simplificação dos ciclos infinitos;
  - e) União disjunta dos conjuntos I_Q e I_R, excetuando-se a instrução rotulada ω;
  - f) Verificar se $Q \equiv R$



Equivalência Forte de Programas Monolíticos

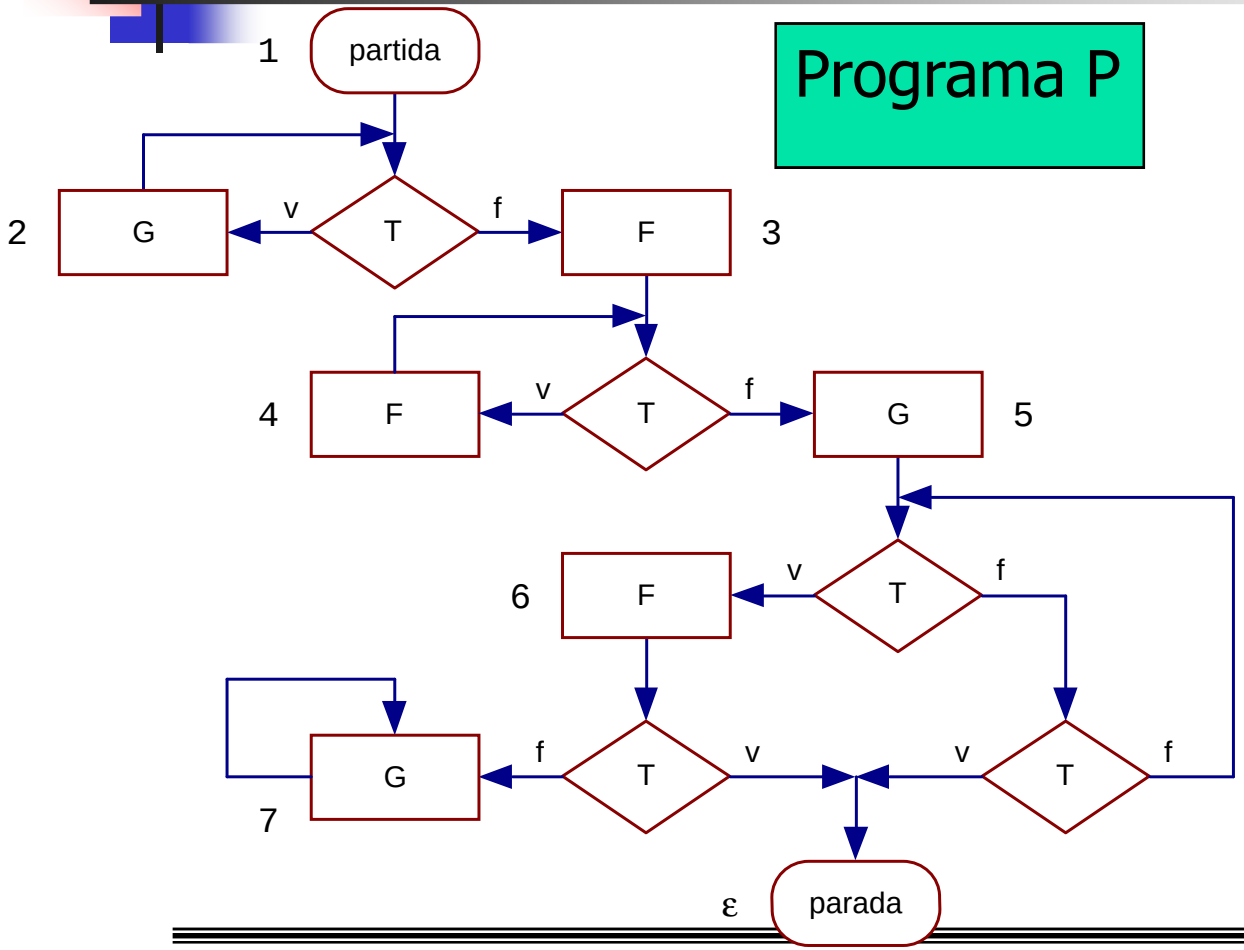
a) Especificação do programa utilizando instruções rotuladas compostas

1. Representar o programa monolítico na forma de fluxograma
2. Atribuir rótulos numéricos para todas as operações
3. Atribuir o rótulo ε para a instrução "parada" → deve ser único
4. Considerar que os rótulos seguem os nós
5. A cada rótulo numérico *i*, criar *i: (F, i′) (G, i″)* se:
   - A condição verdadeira para o teste *T* implica na execução de *F*
   - Após a execução de *F* o próximo rótulo atingido é *i′*
   - A condição falsa para o teste *T* implica na execução de *G*
   - Após a execução de *G* o próximo rótulo atingido é *i″*

- Se a execução de uma operação levar a um loop infinito, deve-se usar (ciclo, *w*) e acrescentar a instrução rotulada composta *w: (ciclo, w), (ciclo, w)* ao programa

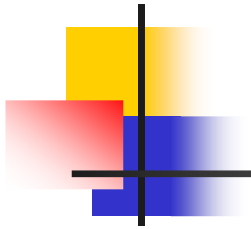
Equivalência Forte de Programas Monolíticos

Exemplo: fluxograma rotulado e instruções rotuladas compostas



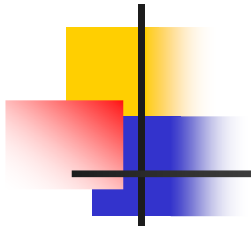
b) Instruções rotuladas compostas

```
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)
```



Algoritmo de Equivalência Forte de Programas Monolíticos

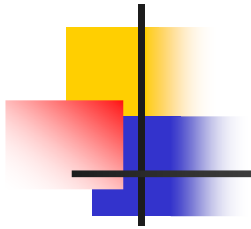
- Passos para verificação:
  - a) Especificação do programa usando instruções rotuladas compostas;
  - b) Conjunto de instruções rotuladas compostas;
  - c) Identificação dos ciclos infinitos;
  - d) Simplificação dos ciclos infinitos;
  - e) União disjunta dos conjuntos I_Q e I_R, excetuando-se a instrução rotulada ω;
  - f) Verificar se $Q \equiv R$



Equivalência Forte de Programas Monolíticos

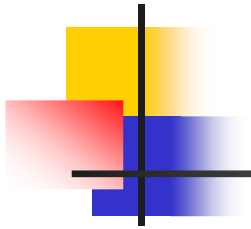
c) Identificação dos ciclos infinitos;

- A *união disjunta* de conjuntos garante que todos os elementos dos conjuntos componentes constituem o conjunto resultante, mesmo que possuam a mesma identificação. Neste caso, considera-se que os elementos são distintos, mesmo que possuam a mesma identificação. Por exemplo, para os conjuntos A={a, x} e B={b,x}, o conjunto resultante da união disjunta é:
  - $\{a_A, x_A, b_B, x_B\}$
- Quando a identificação for única, o índice pode ser omitido
  - $\{a, x_A, b, x_B\}$



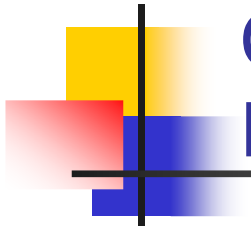
Equivalência Forte de Programas Monolíticos

- **Equivalência Forte: União Disjunta.**
- Sejam:
- $Q = (I_Q, q)$ e $R = (I_R, r)$ dois programas 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, $Q \equiv R$



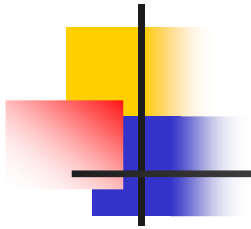
Equivalência Forte de Programas Monolíticos

- O algoritmo para verificação da equivalência forte de Q e R resume-se à verificação se P_q e P_r são equivalentes fortemente. Para tanto, é necessário considerar:
  - *cadeia de conjuntos*: sequência de conjuntos ordenada pela relação de inclusão;
  - *programa monolítico simplificado*: instruções rotuladas compostas que determinam ciclos infinitos a serem excluídos (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.



Cadeia de conjuntos, Cadeia finita de Conjuntos, Limite de uma Cadeia Finita de Conjuntos.

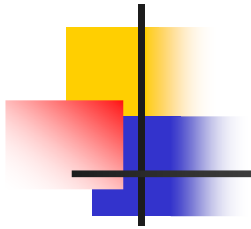
- Uma cadeia de conjuntos $A_0A_1...$ é dita:
  - Uma *Cadeia de Conjuntos* se, para qualquer $k \geq 0$, $A_k \subseteq A_{k+1}$
  - Uma *Cadeia Finita de Conjuntos* é uma cadeia de conjuntos onde existe n, para todo $k \geq 0$, tal que: $A_n = A_{n+k}$
- Neste caso, define-se o *Limite da Cadeia Finita de Conjuntos* como segue:
  - $\lim A_k = A_n$



Identificação de Ciclos Infinitos em Programa Monolítico

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

Identificação de Ciclos Infinitos em Programa Monolítico



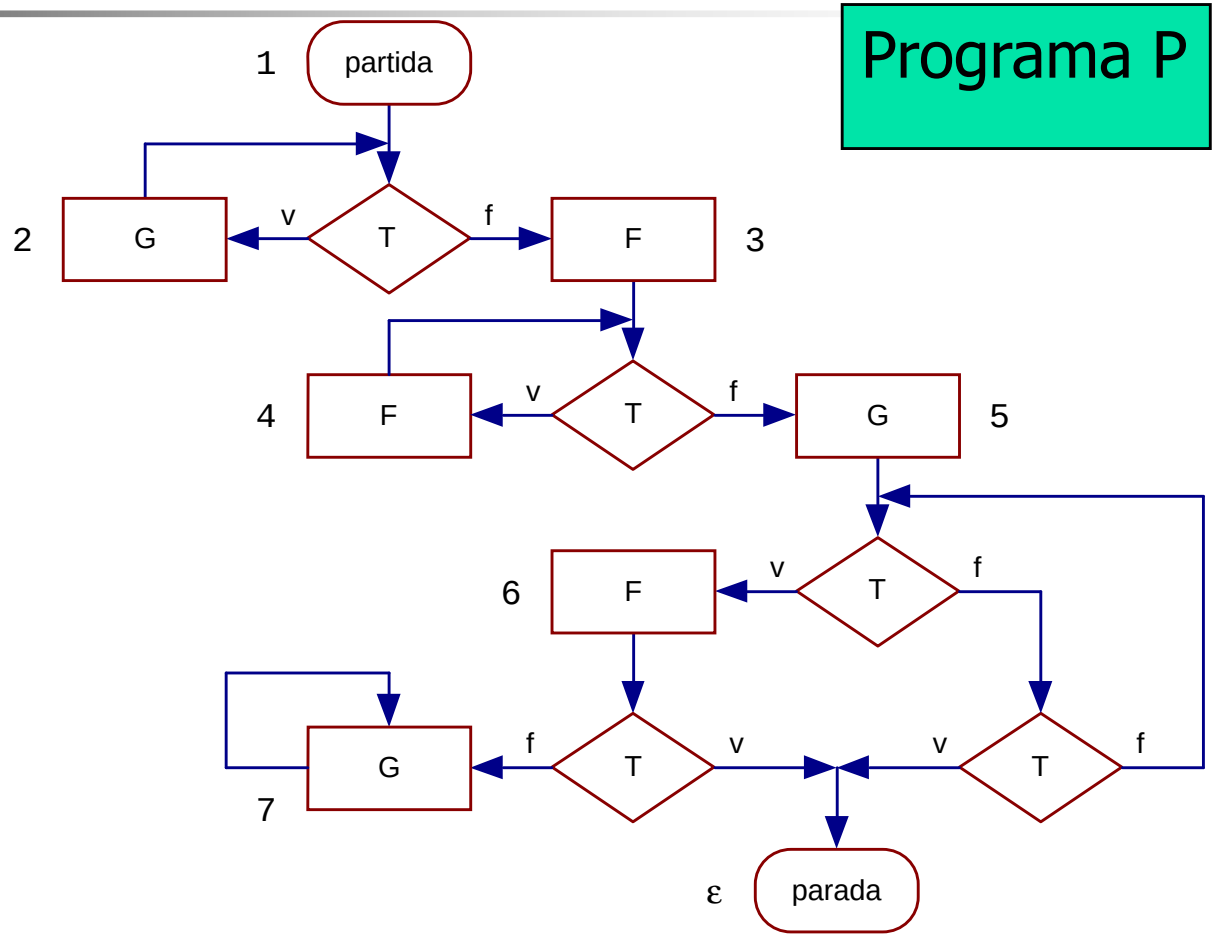
- Se I é um conjunto de n instruções rotuladas compostas. Seja A_0A_1...uma sequência de conjuntos de rótulos definida como segue
  - $A_0=\{\varepsilon\}$
  - $A_1= A_k \cup \{r \mid r$ é rótulo de instrução antecessora de alguma instrução rotulada $A_k\}$
  - Então A_0A_1... é uma cadeia finita de conjuntos e, para qualquer rótulo r de instrução I, tem-se que
    - $(I, r) \equiv (I, \omega)$ se, e somente se, $r \notin \lim A_k$
    - *Proporciona uma maneira fácil de determinar se algum rótulo caracteriza ciclos infinitos*

Identificação de Ciclos Infinitos em Programa Monolítico

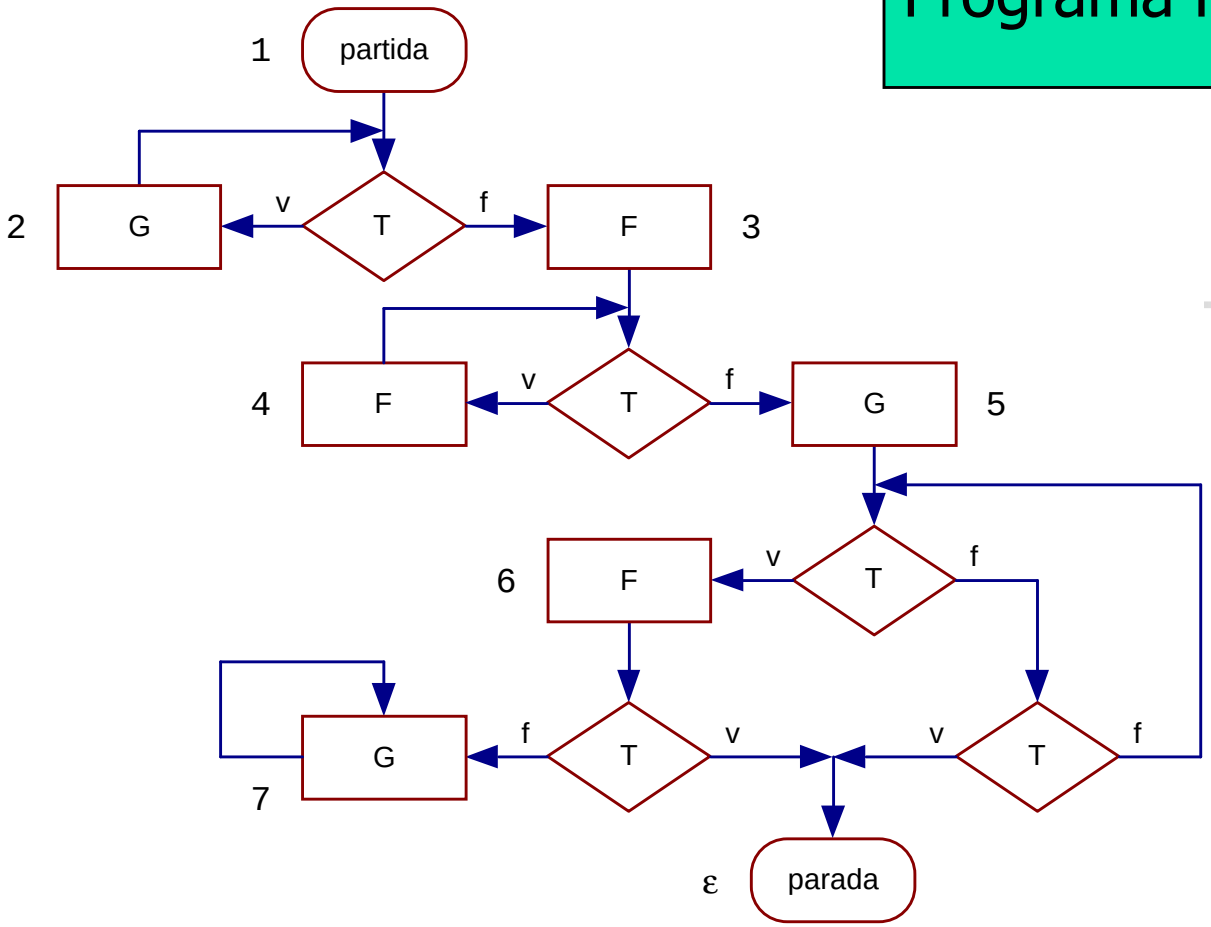
- Exemplo:

Instruções rotuladas compostas

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)



Programa P



- A correspondente cadeia finita de conjuntos é:

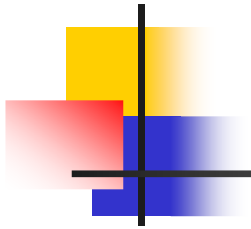
$A_0 = \{\varepsilon\}$
$A_1 = \{6, \varepsilon\}$
$A_2 = \{5, 6, \varepsilon\}$
$A_3 = \{3, 4, 5, 6, \varepsilon\}$
$A_4 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$
$A_5 = \{1, 2, 3, 4, 5, 6, \varepsilon\}$

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

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

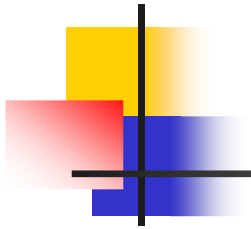
Portanto, pode-se simplificar um conjunto de instruções rotuladas compostas eliminando-se qualquer instrução de rótulo $r \neq \omega$ que determine um ciclo infinito.

A ideia básica é partir da instrução parada, rotulada por ε, determinando os seus antecessores.



Algoritmo de Equivalência Forte de Programas Monolíticos

- Passos para verificação:
  - a) Especificação do programa usando instruções rotuladas compostas;
  - b) Conjunto de instruções rotuladas compostas;
  - c) Identificação dos ciclos infinitos;
  - d) Simplificação dos ciclos infinitos;
  - e) União disjunta dos conjuntos I_Q e I_R, excetuando-se a instrução rotulada ω;
  - f) Verificar se $Q \equiv R$



Algoritmo de Equivalência Forte de Programas Monolíticos

- Passos para verificação:
  - d) Simplificação de ciclos infinitos
    - Seja I um conjunto finito de instruções rotuladas compostas. Algoritmo:
      - Identificação de ciclos infinitos
      - Para qualquer rótulo r de instrução I tal que $r \notin \lim A_k$
        - A instrução rotulada por r é excluída
        - Toda referência a pares da forma (F, r) é substituída por (ciclo, ω)

Programa P

Algoritmo de Equivalência Forte de Programas Monolíticos

- 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.

Instruções rotuladas compostas

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)

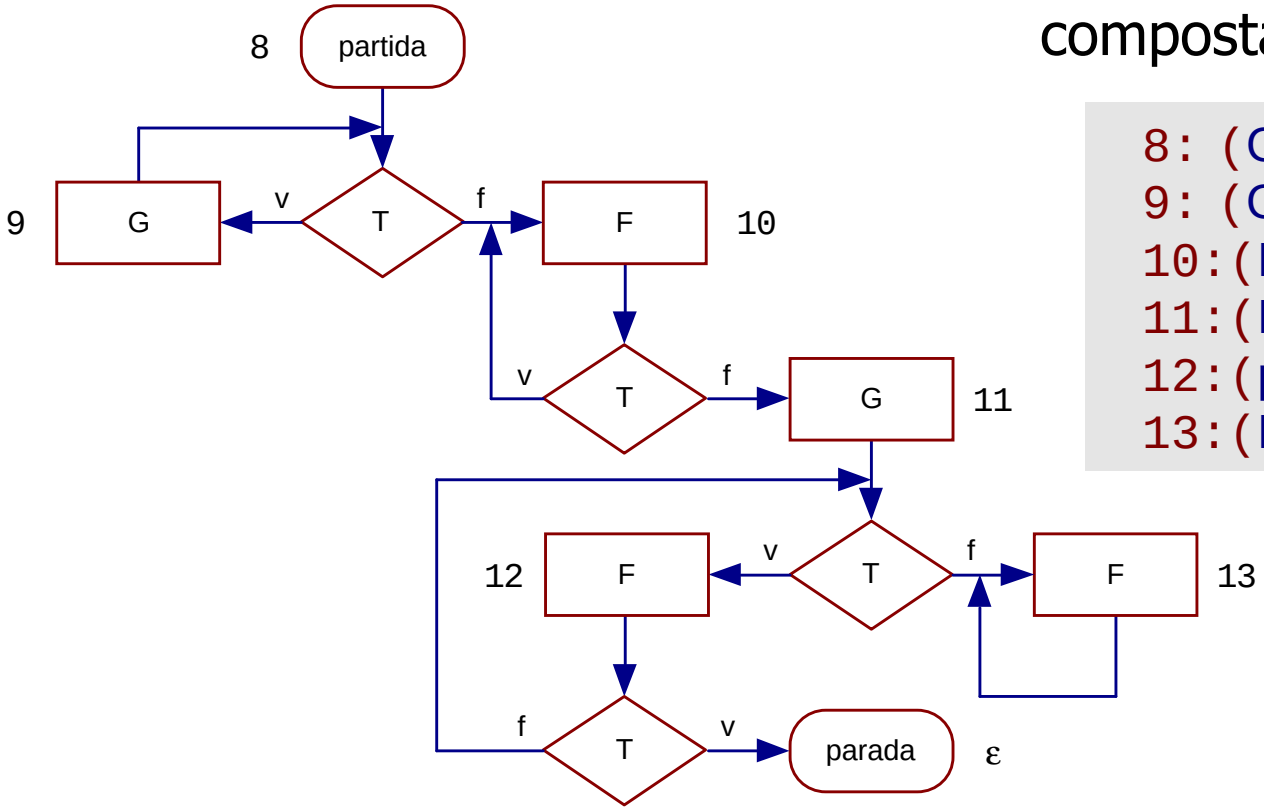
Instruções rotuladas compostas e **simplificadas**

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)

Programa Q

Algoritmo de Equivalência Forte de Programas Monolíticos

a) Especificação do programa usando instruções rotuladas compostas;



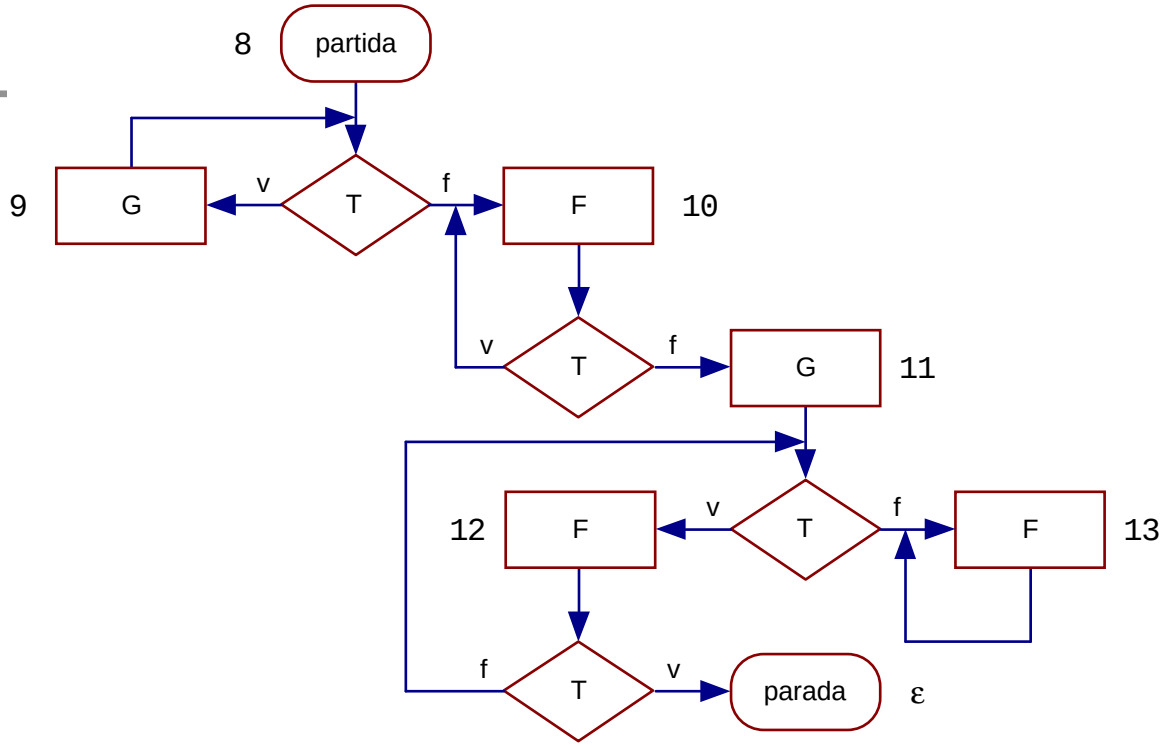

```
8: (G, 9), (F, 10)
9: (G, 9), (F, 10)
10:(F, 10), (G, 11)
11:(F, 12), (F, 13)
12:(parada, ε), (F, 13)
13:(F, 13), (F, 13)
```

Programa Q

Algoritmo de Equivalência Forte de Programas Monolíticos

- 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 $\notin \lim A_k$



Algoritmo de Equivalência Forte de Programas Monolíticos

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)
```

$\lim A_k = \{8, 9, 10, 11, 12, \varepsilon\}$
$(I_r, 13) \equiv (I, \omega)$, pois $\notin \lim A_k$

Instruções rotuladas compostas **simplificadas**

```
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)
```

Algoritmo de Equivalência Forte de Programas Monolíticos

- Relativamente à aplicação do algoritmo, tem-se que:
- *Passo 1:* Seja I a união disjunta dos conjuntos I_Q e I_R, executando-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)

- Seja I um conjunto de n instruções compostas e simplificadas.
- Sejam r e s dois rótulos de instruções de I.
- Define-se, indutivamente, a sequência de conjuntos $B_0B_1\ldots$ 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 $(r", s") \notin B_i$, $(0 \le i \le k)\}$

- Então $B_0B_1\ldots$ é uma sequência 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.

•Sejam r e s dois rótulos de instruções de I.
$B_0 = \{(r, s)\}$
$B_{k+1} = \{(r", s") \mid r"$ e $s"$ são rótulos sucessores de r' e s', respectivamente, $(r', s') \in B_k$ e $(r", s") \notin B_i, (0 \le i \le k)\}$
•Então $B_0B_1...$ é uma sequência 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.

- Para verificar se Q ≡ R é suficiente verificar se (I, 1) ≡ (I, 8)
- Passo 2: como 1 e 8 são rótulos equivalentes fortemente, então:

  $B_0 = \{(1, 8)\}$. Caso contrário, não são equivalentes e o algoritmo para.
- Passos 3 e 4. Para k >=0, a 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
$B_5 = \emptyset$ → converge para o conjunto vazio - Q e R são equivalentes

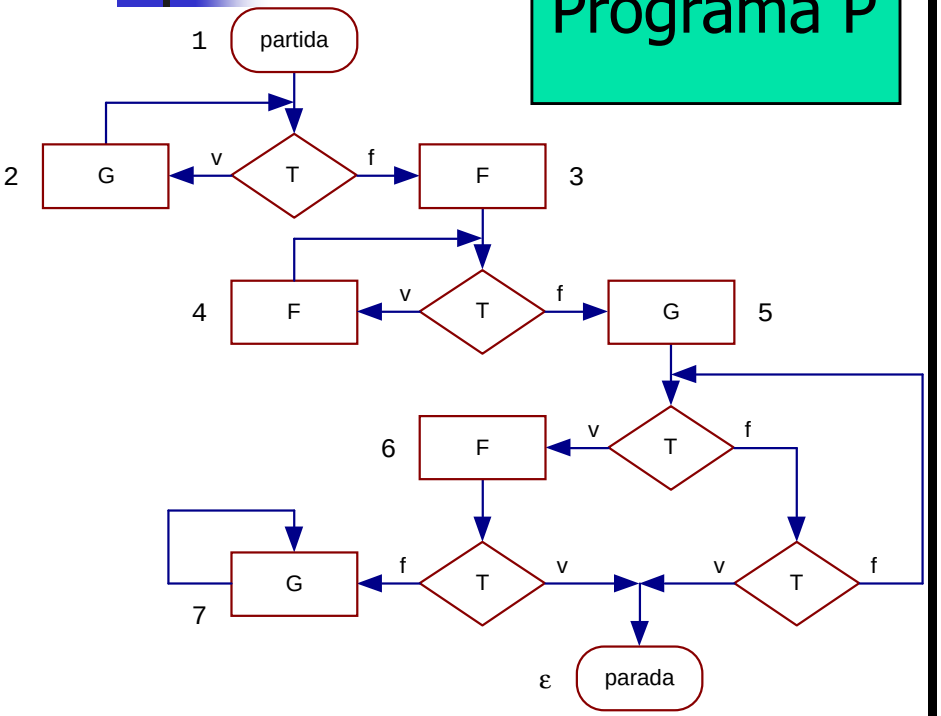
Logo (I, 1) ≡ (I, 8) e, portanto, Q ≡ R

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)

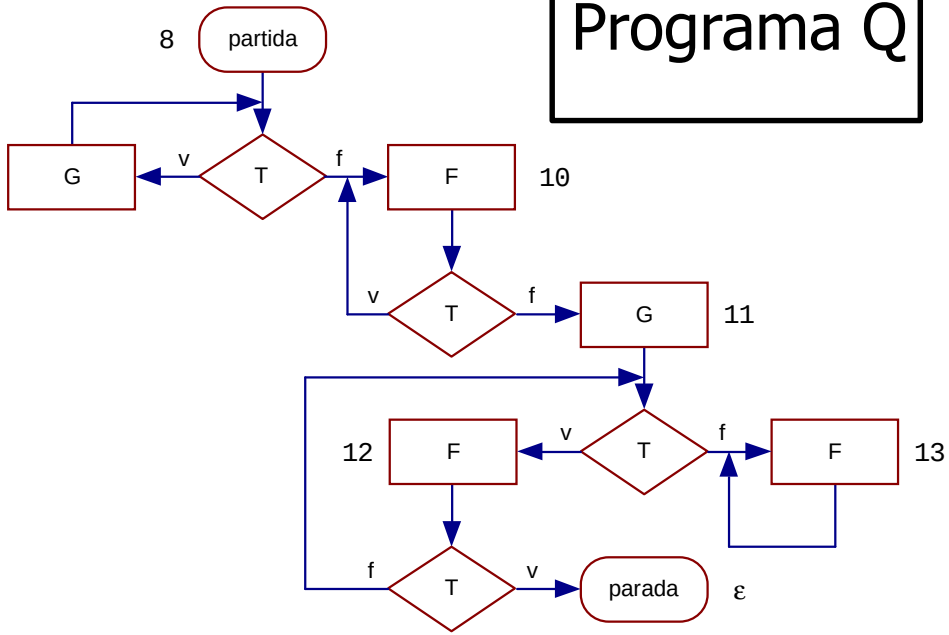
a relação equivalência forte fornece subsídios para analisar a complexidade estrutural de programas.
No caso, Q é estruturalmente "mais otimizado" que P – contem um teste a menos

Algoritmo de Equivalência Forte de Programas Monolíticos

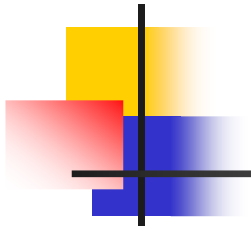
Programa P



Programa Q



a relação equivalência forte fornece subsídios para analisar a complexidade estrutural de programas.
No caso, Q é estruturalmente "mais otimizado" que P – contem um teste a menos



Passos do Algoritmo de Equivalência Forte de Programas Monolíticos

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.

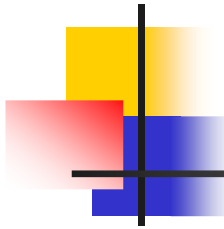
Passo 3. Para $k \geq 0$, define-se o conjunto B_{k+1}, contendo somente os pares (q'', r'') de rótulos sucessores de cada $(q', r') \in B_k$, tais que:

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

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.



Conclusão

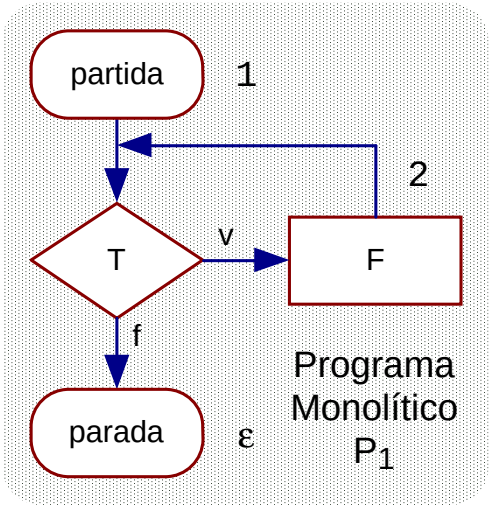
- A partir do conceito de função computada são apresentadas noções de equivalências de programas e máquinas: programas equivalentes fortemente e programas equivalentes em uma máquina;

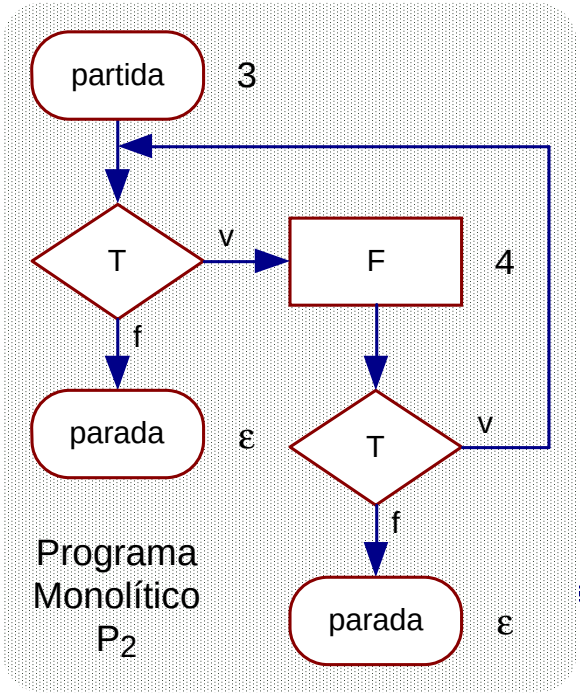
- É apresentado um algoritmo para verificar se programas monolíticos (ou iterativos) são equivalentes fortemente. Até o momento não existe um algoritmo para programas recursivos;

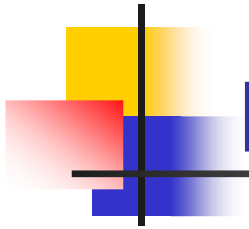
- Posteriormente, será apresentado o conceito de máquina universal.

Programas, Máquinas e Computações

- Aplique o Algoritmo de Verificação da Equivalência Forte de Programas Monolíticos para os programas abaixo







Programas, Máquinas e Computações

- Exercícios:
  - 2.17 ao 2.25
    - Enviar por email (celso.olivete@unesp.br) até o dia 18/03