Equivalências entre Programas e Máquinas

Teoria da Computação

INF05501

Equivalências

- A **determinação de funções computadas** permite a introdução de relações de equivalência de programas e máquinas
- Equivalências determinam **programas e máquinas que produzem uma mesma função computada**
- Com isto, podemos analisar se é possível obterem-se os **mesmos resultados** usando **diferentes combinações de programas e máquinas**

Definições Auxiliares

- Igualdade de funções parciais: duas funções parciais $f, g : X \rightarrow Y$ são ditas **iguais, ou seja,** $f = g$, sss, para cada $x \in X$, **ou** $f(x)$ **e** $g(x)$ **são ambas indefinidas ou ambas são definidas e** $f(x) = g(x)$

- Composição sucessiva de funções: para uma função $f : S \rightarrow S$, **denota-se a composição sucessiva de** f **consigo mesma usando-se um expoente**, de forma que $f^n = f \circ f \circ f \circ ... \circ f$ descreve a composição de f com ela própria n vezes

Relação de Equivalência Forte entre Programas

Sejam P e Q dois programas quaisquer, de tipos quaisquer.

O par (P, Q) pertence à relação de equivalência forte entre programas, denotada por $P \equiv Q$, sss, para qualquer máquina M, as suas funções computadas correspondentes são iguais. Isto é,

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

Neste caso, dizemos que P e Q são programas fortemente equivalentes

Relação de Equivalência Forte entre Programas (cont.)

- Esta relação é uma **relação de equivalência**, visto que, para quaisquer programas P, Q e R

$$P \equiv P$$

$$P \equiv Q \rightarrow Q \equiv P$$

$$P \equiv Q \wedge Q \equiv R \rightarrow P \equiv R$$

- Desta forma, esta relação induz uma partição do conjunto de todos os programas em classes de equivalência

Exemplo de Equivalência Forte entre Programas

Considere o **programa monolítico** P_1

```
1 : se T então vá_para 2 senão vá_para 3
2 : faça F vá_para 1
```

e uma **máquina qualquer** $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$, e que $x \in X$ tal que $\pi_X(x) = v$, onde $v \in V$

Se $\langle P_1, M \rangle$ é definida para x, a computação correspondente é a seguinte:

$(1, v)(2, v)(1, \pi_F(v))(2, \pi_F(v))(1, {\pi_F}^2(v))(2, {\pi_F}^2(v))...(1, {\pi_F}^n(v))(2, {\pi_F}^n(v))$

supondo-se que n seja o menor natural tal que $\pi_T({\pi_F}^n(v)) = \texttt{falso}$

Assim, $\langle P_1, M \rangle(x) = \pi_Y({\pi_F}^n(v))$

Exemplo de Equivalência Forte entre Programas (cont.)

Agora considere o **programa recursivo** P_2

P_2 é $\mathcal{R}$ onde
$\mathcal{R}$ def (se T então faça F;$\mathcal{R}$ senão ✓)

e uma máquina qualquer $M = (V, X, Y, \pi_X, \pi_Y, \Pi_F, \Pi_T)$, e que $x \in X$ tal que $\pi_X(x) = v$, onde $v \in V$

$(\mathcal{R}; \checkmark, v)$
$(($`se` T `então faça` F`;`$\mathcal{R}$ `senão` $\checkmark$$)$`;`$\checkmark, v)$
$(F$`;`$\mathcal{R}$`;`$\checkmark, v)$
$(\mathcal{R}; \checkmark, \pi_F(v))$
$(($`se` T `então faça` F`;`$\mathcal{R}$ `senão` $\checkmark$$)$`;`$\checkmark, \pi_F(v))$
$(F$`;`$\mathcal{R}$`;`$\checkmark, \pi_F(v))$
$(\mathcal{R}; \checkmark, {\pi_F}^2(v))$
$(($`se` T `então faça` F`;`$\mathcal{R}$ `senão` $\checkmark$$)$`;`$\checkmark, {\pi_F}^2(v))$
$(F$`;`$\mathcal{R}$`;`$\checkmark, {\pi_F}^2(v))$
...
$(\mathcal{R}; \checkmark, {\pi_F}^n(v))$
$(($`se` T `então faça` F`;`$\mathcal{R}$ `senão` $\checkmark$$)$`;`$\checkmark, {\pi_F}^n(v))$
$(\checkmark$`;`$\checkmark, {\pi_F}^n(v))$
$(\checkmark, {\pi_F}^n(v))$

Assim, $\langle P_2, M\rangle(x) = \pi_Y({\pi_F}^n(v))$

Exemplo de Equivalência Forte entre Programas (cont.)

- Portanto, podemos concluir que $P_1 \equiv P_2$, pois, para qualquer máquina M

$$\langle P_1, M \rangle = \langle P_2, M \rangle$$

- Note que, como descrito na definição, **a relação de equivalência forte entre programas não requer que os programas sejam do mesmo tipo**, o quê foi demonstrado por este exemplo

Consequências da Relação de Equivalência Forte entre Programas

- Podemos identificar **diferentes programas que pertencem a uma mesma classe de equivalência** (i.e., possuem as mesmas funções computadas para qualquer máquina)

- Funções computadas de programas fortemente equivalentes têm a propriedade de que as **mesmas operações são efetuadas na mesma ordem, independentemente do significado das mesmas**

- Podemos obter **subsídios para analisar a complexidade estrutural de programas** e identificar qual dos programas equivalentes é estruturalmente "mais otimizado" (por exemplo, possui menos testes)

Teoremas sobre Equivalência Forte de Programas

- A partir da definição da relação de equivalência forte entre programas, podemos apresentar alguns **teoremas sobre os tipos de programas envolvidos**

- Tais teoremas visam ao **estabelecimento formal da relação entre os tipos de programas e suas equivalências**

Teorema 1: Iterativo $\Rightarrow$ Monolítico

Teorema 1. *"Para todo programa iterativo P_i existe um programa monolítico P_m tal que $P_i \equiv P_m$."*

- Para provar este teorema, precisamos demonstrar que **cada componente de um programa iterativo possui uma representação correspondente em um programa monolítico**

- Para facilitar tal demonstração, podemos usar **fluxogramas para descrever componentes de um programa monolítico**, visto que ambos são equivalentes

- A prova do Teorema 1 é apresentada a seguir

Teorema 1: Iterativo $\Rightarrow$ Monolítico

Prova:

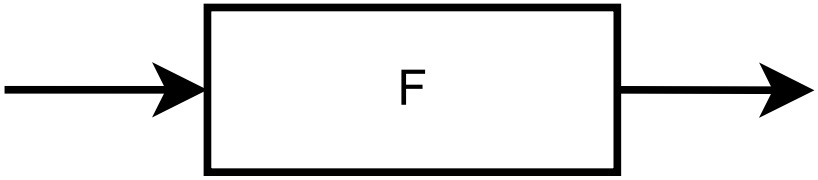
Seja P_i um programa iterativo qualquer. Podemos construir um programa monolítico P_m fortemente equivalente a P_i de forma indutiva da seguinte maneira:

- A **operação vazia** ✓ corresponde ao fluxograma elementar:

OU

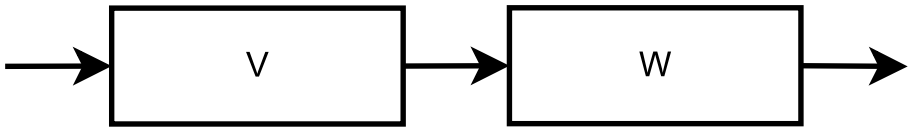
Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Para cada **identificador de operação** F de P_i, temos um fluxograma elementar correspondente do tipo:



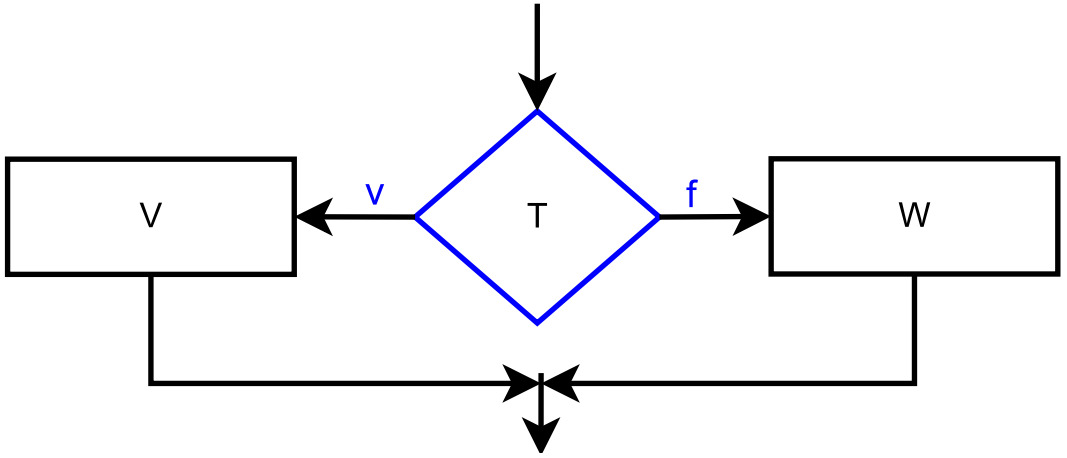
Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Sendo T um **identificador de teste** e V e W **programas iterativos** usados na construção de P_i, então, para cada possível composição correspondem os seguintes fluxogramas elementares:
  - *Composição sequencial*: $V\,;W$



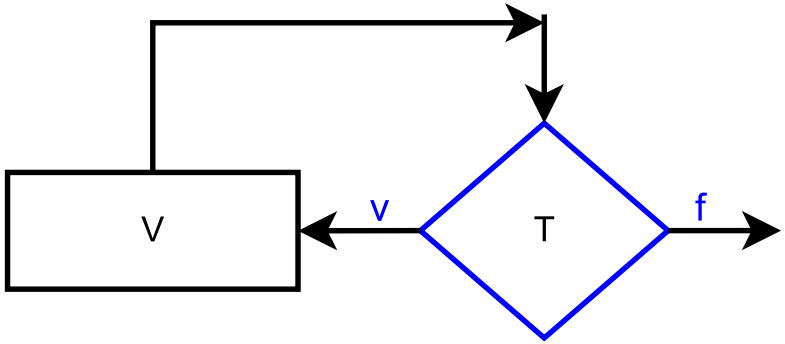
Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Sendo T um **identificador de teste** e V e W **programas iterativos** usados na construção de P_i, então, para cada possível composição correspondem os seguintes fluxogramas elementares:
  - *Composição condicional*: `(se` T `então` V `senão` W`)`



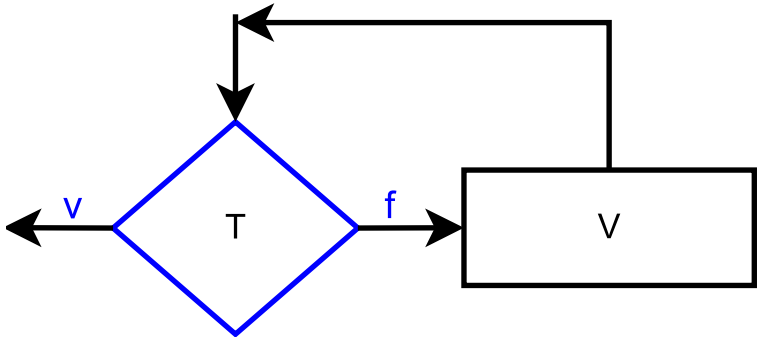
Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Sendo T um **identificador de teste** e V e W **programas iterativos** usados na construção de P_i, então, para cada possível composição correspondem os seguintes fluxogramas elementares:
  - *Composição Enquanto*: `enquanto` T `faça (`V`)`



Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Sendo T um **identificador de teste** e V e W **programas iterativos** usados na construção de P_i, então, para cada possível composição correspondem os seguintes fluxogramas elementares:
  - *Composição Até*: `até` T `faça (`V`)`



Teorema 1: Iterativo $\Rightarrow$ Monolítico

- Adicionalmente, os tratamentos de início e de fim de um programa iterativo correspondem aos fluxogramas elementares de partida e de parada, respectivamente

- Assim, toda construção de um programa iterativo P_i possui uma construção correspondente em um fluxograma que equivale a um programa monolítico P_m e, portanto, $P_i \equiv P_m$.

Fim da Prova

Teorema 2: Monolítico $\Rightarrow$ Recursivo

Teorema 2. *"Para todo programa monolítico P_m existe um programa recursivo P_r tal que $P_m \equiv P_r$."*

- Para esta prova, temos de demonstrar que **toda operação e todo teste de P_m corresponde a alguma expressão de sub-rotina em P_r**
- Esta demonstração é apresentada a seguir

Teorema 2: Monolítico $\Rightarrow$ Recursivo

Prova:

Seja P_m um programa monolítico qualquer, onde $L = \{r_1, r_2, ..., r_n\}$ é o seu conjunto de rótulos. Suponha-se que r_n é o único rótulo final de P_m, F é um identificador de operação e T é um identificador de teste. Então, P_r é um programa recursivo construído a partir de P_m tal que

$$P_r \text{ é } \mathcal{R}_1 \text{ def } E_1, \mathcal{R}_2 \text{ def } E_2, \ldots, \mathcal{R}_n \text{ def } \checkmark$$

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

Teorema 2: Monolítico $\Rightarrow$ Recursivo

- *Operação*: Se r_k é da forma rk: `faça` F `vá_para` $r_k{}'$, então E_k é a seguinte expressão de sub-rotina:

  $F; \mathcal{R}_k{}'$

- *Teste*: Se r_k é da forma rk: `se` T `então vá_para` $r_k{}'$ `senão vá_para` $r_k{}''$, então E_k é a seguinte expressão de sub-rotina:

  `(se` T `então` $\mathcal{R}_k{}'$ `senão` $\mathcal{R}_k{}''$`)`

Logo, $P_m \equiv P_r$.

Fim da Prova

Corolário 1: Iterativo $\Rightarrow$ Recursivo

Dados os resultados dos Teoremas 1 e 2, e sabendo-se que a relação de equivalência forte entre programas é uma relação de equivalência, pela transitividade, temos:

Corolário 1. *"Para todo programa iterativo P_i existe um programa recursivo P_r tal que $P_i \equiv P_r$."*

Equivalências Eventuais

- Vimos que, **obrigatoriamente**:
  - Todo programa iterativo possui um programa monolítico fortemente equivalente a ele
  - Todo programa monolítico possui um programa recursivo fortemente equivalente a ele
  - Todo programa iterativo possui um programa recursivo fortemente equivalente a ele

- No entanto, **existem equivalências fortes que são eventuais**

- Isto é, que **não são necessariamente verdadeiras para todos os programas de um tipo**

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

Teorema 3. *"Dado um programa recursivo P_r qualquer, nem sempre existe um programa monolítico P_m tal que $P_r \equiv P_m$."*

- Para provar esta afirmação, é suficiente apresentar **um programa recursivo que, para uma determinada máquina, não possua um programa monolítico fortemente equivalente**

- A prova do Teorema 3 é apresentada a seguir

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

Prova:

- Considere o **programa recursivo** $duplica$ e a máquina um_reg
- A função computada $\langle duplica, um\_reg \rangle : \mathbb{N} \to \mathbb{N}$, para todo $n \in \mathbb{N}$, é:

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

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

- Suponha que:
  - Existe um **programa monolítico** P_m que computa a mesma função, ou seja, que $\langle P_m, um\_reg\rangle : \mathbb{N} \to \mathbb{N}$ e:

$$\langle duplica, um\_reg\rangle = \langle P_m, um\_reg\rangle$$

  - P_m é constituído de k operações ad
  - $n \in \mathbb{N}$ tal que $n \geq k$

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

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

- Mas, como $n \geq k$, então **pelo menos uma das ocorrências de ad será executada mais de uma vez**; ou seja, existe um ciclo em P_m

- Na função computada por dois programas fortemente equivalentes, as **mesmas operações são efetuadas na mesma ordem**; portanto, o programa monolítico correspondente não pode intercalar testes de controle de fim de ciclo na sequência de operações ad

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

- Desse modo, **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 fortemente equivalentes

- Logo, não existe um programa monolítico fortemente equivalente ao programa recursivo $duplica$

Fim da Prova

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

- Entendendo o resultado do Teorema 3:
  - Um programa de qualquer tipo **não pode ser modificado dinamicamente** durante uma computação
  - Um programa, para ser fortemente equivalente a outro, não pode conter ou usar facilidades adicionais, como memória auxiliar ou operações extras
  - Para que um **programa monolítico possa simular uma recursão sem um número finito e predefinido** de quantas vezes a recursão pode ocorrer, seriam **necessárias infinitas opções de ocorrências das diversas operações ou testes** envolvidos na recursão em questão

Teorema 3: Recursivo $\not\Rightarrow$ Monolítico

- Entendendo o resultado do Teorema 3:
  - **Infinitas opções implicam um programa infinito**, o quê **contradiz a definição de programa monolítico**, o qual é constituído por um conjunto finito de instruções rotuladas
  - Na máquina só existe um registrador, assim o programa tenta utilizar esse registrador tanto para controlar o ciclo como para acumular o resultado, o quê resulta em um **programa com ciclo infinito**

Teorema 4: Monolítico $\not\Rightarrow$ Iterativo

Teorema 4. *"Dado um programa monolítico P_m qualquer, nem sempre existe um programa iterativo P_i tal que $P_m \equiv P_i$."*

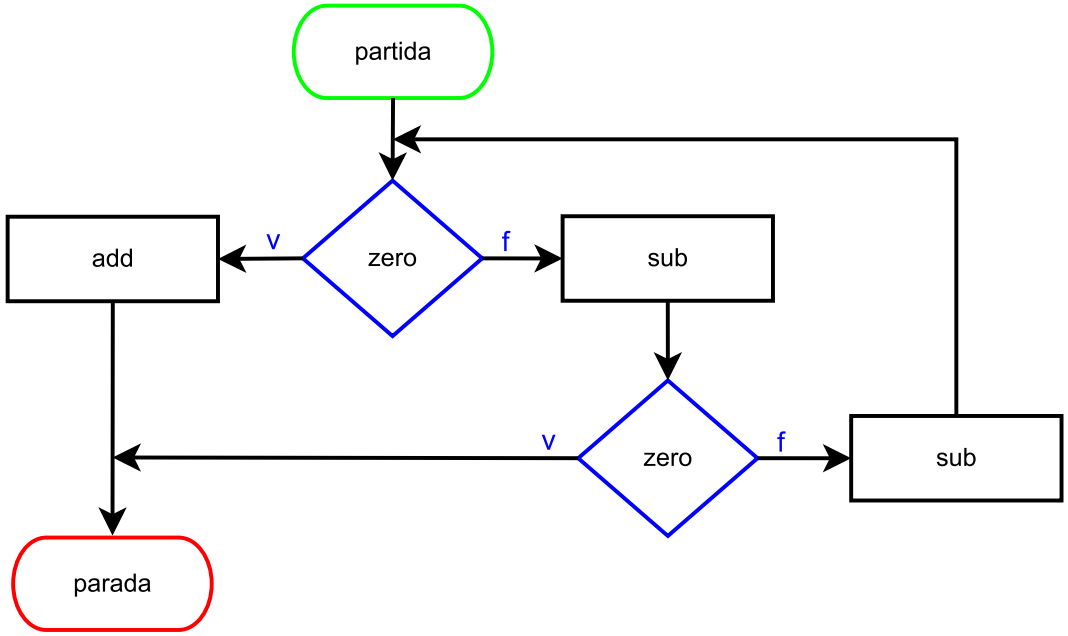
- Para provar esta afirmação, é suficiente apresentar **um programa monolítico que, para uma determinada máquina, não possua um programa iterativo fortemente equivalente**

- A prova do Teorema 4 é apresentada a seguir

Teorema 4: Monolítico $\not\Rightarrow$ Iterativo

Prova:

- Considere o programa monolítico par, apresentado como um fluxograma:



Teorema 4: Monolítico $\not\Rightarrow$ Iterativo

- Considerando-se a máquina um_reg, a função computada $\langle par, um\_reg \rangle$: $\mathbb{N} \to \mathbb{N}$ é tal que, para todo $n \in \mathbb{N}$:

  $\langle par, um\_reg \rangle(n) = 1$, se n é par

  $\langle par, um\_reg \rangle(n) = 0$, se n é ímpar

Teorema 4: Monolítico $\not\Rightarrow$ Iterativo

- Suponha que:
  - Existe um **programa iterativo** P_i que computa a mesma função, de forma que $\langle P_i, um\_reg \rangle : \mathbb{N} \rightarrow \mathbb{N}$, tal que

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

  - O programa P_i em questão é constituído de k operações sub
  - $n \in \mathbb{N}$, tal que $n \geq k$

Teorema 4: Monolítico $\not\Rightarrow$ Iterativo

- 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
- **Independentemente de o valor ser par ou ímpar, o ciclo terminará sempre na mesma condição**, sendo a computação resultante **incapaz de distinguir entre os dois casos**, o que é um absurdo, pois é suposto que os dois programas são fortemente equivalentes
- Logo, não existe um programa iterativo fortemente equivalente ao programa monolítico par

Fim da Prova

Poder Computacional de Programas

- Os teoremas vistos podem levar à **conclusões incorretas** sobre o **poder computacional das diferentes classes de programas**

- A Relação de Equivalência Forte entre Programas considera a **coincidência de funções computadas** por dois programas distintos **em qualquer máquina**

- No entanto, é possível, por exemplo, dado **um programa recursivo qualquer em qualquer máquina**, encontrar-se **um programa monolítico em uma dada máquina** que possui a **mesma função computada**

Poder Computacional de Programas (cont.)

- Na verdade, as três classes de programas possuem o mesmo poder computacional

- Portanto, 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 sua ordem de execução) dos programas**

Equivalência de Programas

- Em certas situações, podemos querer analisar uma **noção de equivalência de programas mais fraca**

- Neste caso, podemos restringir à análise a verificar a **equivalência** de dois programas **em uma dada máquina**

Definição Formal de Equivalência de Programas

Sejam P e Q dois programas quaisquer, não necessariamente do mesmo tipo, e M uma máquina qualquer. O par (P, Q) está na Relação de Equivalência de Programas na Máquina M, denotado por $P \equiv_M Q$, sss, a suas correspondentes funções parciais computadas são iguais. Ou seja:

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

Neste caso, diz-se que P e Q são programas equivalentes na máquina M ou, simplesmente, que P e Q são programas M-equivalentes

Definição Formal de Equivalência de Programas (cont.)

- Desta forma, podemos analisar se **dois programas computam a mesma função quando executados em uma mesma máquina**

- Entretanto, há máquinas para as quais não se pode provar a existência de um algoritmo para determinar se, dados dois programas, eles são ou não M-equivalentes

Equivalência de Máquinas

- Assim como podemos analisar a equivalência entre dois programas, também podemos verificar se **duas máquinas são equivalentes**

- Dizemos que duas máquinas são equivalentes quando uma pode **simular** a outra e vice-versa

Simulação Forte entre 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 quaisquer.* N *simula fortemente* M *sss, para qualquer programa* P *para* M*, existe um programa* Q *para* N *tal que as funções parciais computadas coincidem. Isto é,*

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

Simulação Forte entre Máquinas (cont.)

- Note-se que a análise pode ser feita usando-se **programas diferentes**
- 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

Simulação entre 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 quaisquer.* N *simula* M *sss, 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:

$$\langle P, M \rangle = d \circ \langle Q, N \rangle \circ c$$

Relação de Equivalência entre Máquinas

Sejam M e N duas máquinas quaisquer. O par (M, N) pertence à Relação de Equivalência entre Máquinas sss

M simula N **E** N simula M