MAC 414
Autômatos, Computabilidade e Complexidade

aula 13 — 4/11/2020

Outro modelo de computação

Outro modelo de computação

Idéia: definir funções, com números naturais como argumentos, que sejam *obviamente* computáveis.

Outro modelo de computação

Idéia: definir funções, com números naturais como argumentos, que sejam *obviamente* computáveis.

Construção bottom-up: começa com coisas super simples, e combinadores que são *obviamente* computáveis.

Outro modelo de computação

Idéia: definir funções, com números naturais como argumentos, que sejam *obviamente* computáveis.

Construção bottom-up: começa com coisas super simples, e combinadores que são *obviamente* computáveis.

Kleene (1936) propôs: funções recursivas primitivas e funções μ-recursivas.

Outro modelo de computação

Idéia: definir funções, com números naturais como argumentos, que sejam *obviamente* computáveis.

Construção bottom-up: começa com coisas super simples, e combinadores que são *obviamente* computáveis.

Kleene (1936) propôs: funções recursivas primitivas e funções μ-recursivas.

L&P 4.7.

Funções básicas

São funções de $\mathbb{N}^k$ em $\mathbb{N}$, para cada $k \geq 0$.

1. Para cada $k \geq 0$, a função zero k-ária:

$$\textbf{zero}(n_1, n_2, \ldots, n_k) = 0.$$

Funções básicas

São funções de $\mathbb{N}^k$ em $\mathbb{N}$, para cada $k \geq 0$.

1. Para cada $k \geq 0$, a função zero k-ária:

   $$\mathbf{zero}(n_1, n_2, \ldots, n_k) = 0.$$

2. Para $k \geq j \geq 0$ a j-ésima k-ária identidade (projeção)

   $$\mathbf{id}_{k,j}(n_1, n_2, \ldots, n_k) = n_j.$$

Funções básicas

São funções de $\mathbb{N}^k$ em $\mathbb{N}$, para cada $k \geq 0$.

1. Para cada $k \geq 0$, a função zero k-ária:

$$\mathbf{zero}(n_1, n_2, \ldots, n_k) = 0.$$

2. Para $k \geq j \geq 0$ a j-ésima k-ária identidade (projeção)

$$\mathbf{id}_{k,j}(n_1, n_2, \ldots, n_k) = n_j.$$

3. A função sucessor

$$\mathbf{succ}(n) = n + 1.$$

Combinadores

1. Dados $k, \ell \geq 0$ sejam $g : \mathbb{N}^k \rightarrow \mathbb{N}$ uma função k-ária, e $h_1, h_2, \ldots, h_k : \mathbb{N}^\ell \rightarrow \mathbb{N}$ funções ℓ-árias. Então, a composição de g com $h_1, h_2, \ldots, h_k$ é a função ℓ-ária definida por

$$f(n_1, n_2, \ldots, n_\ell) = g(h_1(n_1, n_2, \ldots, n_\ell), h_2(n_1, n_2, \ldots, n_\ell), \ldots, h_k(n_1, n_2, \ldots, n_\ell)).$$

Combinadores

1. Dados $k, \ell \geq 0$ sejam $g : \mathbb{N}^k \to \mathbb{N}$ uma função k-ária, e $h_1, h_2, \ldots, h_k : \mathbb{N}^\ell \to \mathbb{N}$ funções ℓ-árias. Então, a composição de g com $h_1, h_2, \ldots, h_k$ é a função ℓ-ária definida por

$$f(n_1, n_2, \ldots, n_\ell) = g(h_1(n_1, n_2, \ldots, n_\ell), h_2(n_1, n_2, \ldots, n_\ell), \ldots, h_k(n_1, n_2, \ldots, n_\ell)).$$

2. Sejam $k \geq 0$, g uma função k-ária e h uma função $(k+2)$-ária. A função definida recursivamente por g e h é a função $(k+1)$-ária definida por

$$f(n_1, n_2, \ldots, n_k, 0) = g(n_1, n_2, \ldots, n_k),$$
$$f(n_1, n_2, \ldots, n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots, n_k, m))$$

Definição recursiva e recursão

```
def f(a,...,b,m):
 if m == 0:
   # g é código envolvendo a,...,b
   return g(a,...,b)
 else:
   m = m-1
   p = f(a,...,b,m)
   # Compute com a,...,b,m,p e devolva algo
```

Uma função é recursiva primitiva se pode ser obtida das funções básicas usando composição e recursão.

Uma função é recursiva primitiva se pode ser obtida das funções básicas usando composição e recursão.

Vamos ver que isso equivale a ser computável (em C, por exemplo) usando só:

- Expressões aritméticas, booleanas e atribuições.

Uma função é recursiva primitiva se pode ser obtida das funções básicas usando composição e recursão.

Vamos ver que isso equivale a ser computável (em C, por exemplo) usando só:

- Expressões aritméticas, booleanas e atribuições.
- **if**

Uma função é recursiva primitiva se pode ser obtida das funções básicas usando composição e recursão.

Vamos ver que isso equivale a ser computável (em C, por exemplo) usando só:

- Expressões aritméticas, booleanas e atribuições.
- **if**
- **for**(`i = 0; i < n; i++`), onde n e `i` não são alterados durante o loop.

Uma função é recursiva primitiva se pode ser obtida das funções básicas usando composição e recursão.

Vamos ver que isso equivale a ser computável (em C, por exemplo) usando só:

- Expressões aritméticas, booleanas e atribuições.
- **if**
- **for**(`i = 0; i < n; i++`), onde n e `i` não são alterados durante o loop.
- Funçoes cujo grafo de chamadas é acíclico.

Exemplos

- Funções constantes (com qualquer número de argumentos)

Exemplos

- Funções constantes (com qualquer número de argumentos)

Exemplos

- Funções constantes (com qualquer número de argumentos)
  $5 = \mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{zero}(\cdots))))))$
- Sinal

Exemplos

- Funções constantes (com qualquer número de argumentos)
  $5 = \mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{zero}(\cdots))))))$
- Sinal

Exemplos

- Funções constantes (com qualquer número de argumentos)
  $5 = \mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{zero}(\cdots))))))$
- Sinal

$$\begin{aligned} \mathbf{sgn}(0) &= 0 \\ \mathbf{sgn}(n+1) &= 1. \end{aligned}$$

Exemplos

- $\textbf{plus2}(n) = n + 2$

Exemplos

- $\text{plus2}(n) = n + 2$

Exemplos

- $\mathbf{plus2}(n) = n + 2 = \mathbf{succ}(\mathbf{succ}(n))$

Exemplos

- $\text{plus2}(n) = n + 2 = \text{succ}(\text{succ}(n))$
- $\text{plus}(m, n) = m + n$

Exemplos

- $\text{plus2}(n) = n + 2 = \text{succ}(\text{succ}(n))$
- $\text{plus}(m, n) = m + n$

Exemplos

- $\textbf{plus2}(n) = n + 2 = \textbf{succ}(\textbf{succ}(n))$
- $\textbf{plus}(m, n) = m + n$
  Definido por recursão

Exemplos

- **plus2**$(n) = n + 2 =$ **succ**$($**succ**$(n))$
- **plus**$(m, n) = m + n$
  Definido por recursão
  Relembrando:

$$f(n_1, n_2, \ldots, n_k, 0) = g(n_1, n_2, \ldots, n_k),$$
$$f(n_1, n_2, \ldots, n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots, n_k, m))$$

Exemplos

- $\mathbf{plus2}(n) = n + 2 = \mathbf{succ}(\mathbf{succ}(n))$
- $\mathbf{plus}(m, n) = m + n$
  Definido por recursão
  Relembrando:

$$f(n_1, n_2, \ldots, n_k, 0) = g(n_1, n_2, \ldots, n_k),$$
$$f(n_1, n_2, \ldots, n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots, n_k, m))$$

$k = 1$, $f = \mathbf{plus}$, $g = \mathbf{id}_{1,1}$,
$h(m, n, p) = \mathbf{succ}(\mathbf{id}_{3,3}) = p + 1.$

Exemplos

- $\mathbf{plus2}(n) = n + 2 = \mathbf{succ}(\mathbf{succ}(n))$
- $\mathbf{plus}(m, n) = m + n$
  Definido por recursão
  Relembrando:

$$f(n_1, n_2, \ldots, n_k, 0) = g(n_1, n_2, \ldots, n_k),$$
$$f(n_1, n_2, \ldots, n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots, n_k, m))$$

$k = 1$, $f = \mathbf{plus}$, $g = \mathbf{id}_{1,1}$,
$h(m, n, p) = \mathbf{succ}(\mathbf{id}_{3,3}) = p + 1.$

Exemplos

- $\mathbf{plus2}(n) = n + 2 = \mathbf{succ}(\mathbf{succ}(n))$
- $\mathbf{plus}(m, n) = m + n$
  Definido por recursão
  Relembrando:

$$f(n_1, n_2, \ldots, n_k, 0) = g(n_1, n_2, \ldots, n_k),$$
$$f(n_1, n_2, \ldots, n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots, n_k, m))$$

$k = 1$, $f = \mathbf{plus}$, $g = \mathbf{id}_{1,1}$,
$h(m, n, p) = \mathbf{succ}(\mathbf{id}_{3,3}) = p + 1.$

$$\mathbf{plus}(m, 0) = m,$$
$$\mathbf{plus}(m, n+1) = \mathbf{succ}(\underbrace{\mathbf{plus}(m, n)}_{m+n}).$$

n+1

Exemplos

- $\textbf{mult}(m, n) = mn$

Exemplos

- $\mathbf{mult}(m, n) = mn$

Exemplos

- $\mathbf{mult}(m, n) = mn$

$$\begin{aligned} \mathbf{mult}(m, 0) &= \mathbf{zero}(m), \\ \mathbf{mult}(m, n+1) &= \mathbf{plus}(m, \mathbf{mult}(m, n)) \\ &= \mathbf{plus}(\mathbf{id}_{1,1}, \mathbf{id}_{3,3})(m, n, p). \end{aligned}$$

$m(n+1) = mn + m$

- $\mathbf{exp}(m, n) = m^n$

Exemplos

- $\mathbf{mult}(m,n) = mn$

$$\begin{aligned} \mathbf{mult}(m,0) &= \mathbf{zero}(m), \\ \mathbf{mult}(m,n+1) &= \mathbf{plus}(m,\mathbf{mult}(m,n)) \\ &= \mathbf{plus}(\mathbf{id}_{1,1},\mathbf{id}_{3,3})(m,n,p). \end{aligned}$$

- $\mathbf{exp}(m,n) = m^n$

Exemplos

- $\mathbf{mult}(m,n) = mn$

$$\begin{aligned} \mathbf{mult}(m,0) &= \mathbf{zero}(m), \\ \mathbf{mult}(m,n+1) &= \mathbf{plus}(m,\mathbf{mult}(m,n)) \\ &= \mathbf{plus}(\mathbf{id}_{1,1},\mathbf{id}_{3,3})(m,n,p). \end{aligned}$$

- $\mathbf{exp}(m,n) = m^n$

$$\begin{aligned} \mathbf{exp}(m,0) &= \mathbf{succ}(\mathbf{zero}(m)), \\ \mathbf{exp}(m,n+1) &= \mathbf{mult}(m,\mathbf{exp}(m,n)) \end{aligned}$$

1

Exemplos

- $\mathbf{mult}(m,n) = mn$

$$\begin{aligned}\mathbf{mult}(m,0) &= \mathbf{zero}(m),\\ \mathbf{mult}(m,n+1) &= \mathbf{plus}(m,\mathbf{mult}(m,n))\\ &= \mathbf{plus}(\mathbf{id}_{1,1},\mathbf{id}_{3,3})(m,n,p).\end{aligned}$$

- $\mathbf{exp}(m,n) = m^n$

$$\begin{aligned}\mathbf{exp}(m,0) &= \mathbf{succ}(\mathbf{zero}(m)),\\ \mathbf{exp}(m,n+1) &= \mathbf{mult}(m,\mathbf{exp}(m,n))\end{aligned}$$

- $\mathbf{torre}(m,n) = m^{\underbrace{m^{m^{\cdot^{\cdot^{\cdot^{m}}}}}}_{n \text{ vezes}}}$

Exemplos

- $\mathbf{mult}(m,n) = mn$

$$\begin{aligned}\mathbf{mult}(m,0) &= \mathbf{zero}(m),\\ \mathbf{mult}(m,n+1) &= \mathbf{plus}(m,\mathbf{mult}(m,n))\\ &= \mathbf{plus}(\mathbf{id}_{1,1},\mathbf{id}_{3,3})(m,n,p).\end{aligned}$$

- $\mathbf{exp}(m,n) = m^n$

$$\begin{aligned}\mathbf{exp}(m,0) &= \mathbf{succ}(\mathbf{zero}(m)),\\ \mathbf{exp}(m,n+1) &= \mathbf{mult}(m,\mathbf{exp}(m,n))\end{aligned}$$

- $\mathbf{torre}(m,n) = m^{\underbrace{m^{m^{\cdot^{\cdot^{\cdot^{m}}}}}}_{n \text{ vezes}}}$

Exemplos

- $\mathbf{mult}(m,n) = mn$

$$\begin{aligned}\mathbf{mult}(m,0) &= \mathbf{zero}(m),\\ \mathbf{mult}(m,n+1) &= \mathbf{plus}(m,\mathbf{mult}(m,n))\\ &= \mathbf{plus}(\mathbf{id}_{1,1},\mathbf{id}_{3,3})(m,n,p).\end{aligned}$$

- $\mathbf{exp}(m,n) = m^n$

$$\begin{aligned}\mathbf{exp}(m,0) &= \mathbf{succ}(\mathbf{zero}(m)),\\ \mathbf{exp}(m,n+1) &= \mathbf{mult}(m,\mathbf{exp}(m,n))\end{aligned}$$

- $\mathbf{torre}(m,n) = m^{\underbrace{m^{m^{\cdot^{\cdot^{\cdot^{m}}}}}}_{n \text{ vezes}}}$

$$\begin{aligned}\mathbf{torre}(m,0) &= \mathbf{succ}(\mathbf{zero}(m)),\\ \mathbf{torre}(m,n+1) &= \mathbf{exp}(m,\mathbf{torre}(m,n))\end{aligned}$$

Polinômios e outros

Polinômios e outros

$25n + 47 + 32n^3 + 44^n$ é primitiva recursiva

Polinômios e outros

$25n + 47 + 32n^3 + 44^n$ é primitiva recursiva

E subtração? Divisão?

Monus

Monus

Subtração não é definida em $\mathbb{N}$, mas é util a operação monus

$$m \dot{-} n = \max(m - n, 0).$$

Monus

Subtração não é definida em $\mathbb{N}$, mas é util a operação monus

$$m \dot{-} n = \max(m - n, 0).$$

Primeiro definimos o predessessor

$$\mathbf{pred}(0) = 0,$$
$$\mathbf{pred}(m+1) = m.$$

Monus

Subtração não é definida em $\mathbb{N}$, mas é util a operação monus

$$m \dot{-} n = \max(m - n, 0).$$

Primeiro definimos o predessessor

$$\begin{aligned} \mathbf{pred}(0) &= 0, \\ \mathbf{pred}(m+1) &= m. \end{aligned}$$

E agora monus:

$$\begin{aligned} \mathbf{monus}(m, 0) &= m \\ \mathbf{monus}(m, n+1) &= \mathbf{pred}(\mathbf{monus}(m, n)) \\ [m \dot{-} (n+1) &= \mathbf{pred}(m \dot{-} n)] \end{aligned}$$

Predicados

Predicados

São funções com imagem em $\{0,1\}$.

Predicados

São funções com imagem em $\{0, 1\}$.
Um predicado recursivo primitivo importante:

$$\mathbf{iszero}(0) = 1, \qquad \mathbf{iszero}(m+1) = 0.$$

$\mathrm{iszero}(n) = 1 \dot{-} \mathrm{sgn}(n)$

Predicados

São funções com imagem em $\{0,1\}$.
Um predicado recursivo primitivo importante:

$$\textbf{iszero}(0) = 1, \qquad \textbf{iszero}(m+1) = 0.$$

Outros

$$m \geq n : \textbf{iszero}(n \dot{-} m)$$
$$m = n : \textbf{iszero}(n \dot{-} m + m \dot{-} n)$$

Predicados

São funções com imagem em $\{0,1\}$.
Um predicado recursivo primitivo importante:

$$\mathbf{iszero}(0) = 1, \qquad \mathbf{iszero}(m+1) = 0.$$

Outros

$$m \geq n : \mathbf{iszero}(n \dot{-} m)$$
$$m = n : \mathbf{iszero}(n \dot{-} m + m \dot{-} n)$$

Se p, q são predicados recursivos primitivos, com a mesma aridade, também são recursivos primitivos

$$!p(a,\ldots,b) = 1 \dot{-} p(a,\ldots,b)$$
$$(p||q)(a,\ldots,b) = \mathbf{sign}(p(a,\ldots,b) + q(a,\ldots,b))$$
$$(p\&\&q)(a,\ldots,b) = p(a,\ldots,b)q(a,\ldots,b)$$

Definição por casos

Definição por casos

g, h funções, p predicados, todos r.p., mesma aridade.
Também é r.p.:

$$f(a,\ldots,b) = \begin{cases} g(a,\ldots,b) & \text{se } p(a,\ldots,b) \\ h(a,\ldots,b) & \text{caso contrário.} \end{cases}$$

Definição por casos

g, h funções, p predicados, todos r.p., mesma aridade.
Também é r.p.:

$$f(a,\ldots,b) = \begin{cases} g(a,\ldots,b) & \text{se } p(a,\ldots,b) \\ h(a,\ldots,b) & \text{caso contrário.} \end{cases}$$

$$f = p \cdot g + !p \cdot h.$$

Definição por casos

g, h funções, p predicados, todos r.p., mesma aridade.
Também é r.p.:

$$f(a,\ldots,b) = \begin{cases} g(a,\ldots,b) & \text{se } p(a,\ldots,b) \\ h(a,\ldots,b) & \text{caso contrário.} \end{cases}$$

$$f = p \cdot g + !p \cdot h.$$

$$0\%n = 0$$

$$(m+1)\%n = \begin{cases} 0 & \text{se } m\%n = \mathbf{pred(n)}, \\ m\%n + 1 & \text{caso contrário.} \end{cases}$$

Definição por casos

g, h funções, p predicados, todos r.p., mesma aridade.
Também é r.p.:

$$f(a,\ldots,b) = \begin{cases} g(a,\ldots,b) & \text{se } p(a,\ldots,b) \\ h(a,\ldots,b) & \text{caso contrário.} \end{cases}$$

$$f = p \cdot g + !p \cdot h.$$

$$0\%n = 0$$

$$(m+1)\%n = \begin{cases} 0 & \text{se } m\%n = \mathbf{pred(n)}, \\ m\%n + 1 & \text{caso contrário.} \end{cases}$$

Divisão inteira: exercício.

Computáveis?

Computáveis?

Funções recursivas primitivas são claramente computáveis (por programas).

Computáveis?

Funções recursivas primitivas são claramente computáveis (por programas).

Existem funções computáveis que não são r.p.

Computáveis?

Funções recursivas primitivas são claramente computáveis (por programas).

Existem funções computáveis que não são r.p.

Considere um alfabeto Σ suficiente para descrever funções r.p. Enumere Σ^* em ordem comprimento-lex, verificando se a palavra define uma função r.p. Isso determina

$f_n = n$-ésima função r.p. achada.

Computáveis?

Funções recursivas primitivas são claramente computáveis (por programas).

Existem funções computáveis que não são r.p.

Considere um alfabeto Σ suficiente para descrever funções r.p. Enumere Σ^* em ordem comprimento-lex, verificando se a palavra define uma função r.p. Isso determina

f_n = n-ésima função r.p. achada.

Diagonalize: seja $g(n) = f_n(n) + 1$. Então g não é r.p.

$g = f_m \quad g(m) = f_m(m) + 1 = g(m) + 1$

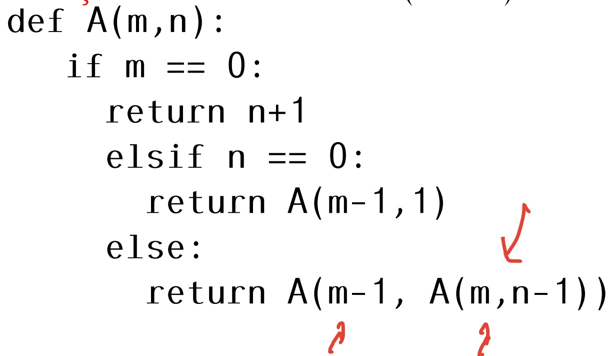
Um exemplo concreto

Um exemplo concreto

A função de Ackermann (1928):

Um exemplo concreto

A função de Ackermann (1928):



```
def A(m,n):
    if m == 0:
      return n+1
      elsif n == 0:
        return A(m-1,1)
      else:
        return A(m-1, A(m,n-1))
```

Um exemplo concreto

A função de Ackermann (1928):

```
def A(m,n):
    if m == 0:
      return n+1
      elsif n == 0:
        return A(m-1,1)
      else:
        return A(m-1, A(m,n-1))
```

Se $f(n_1,\ldots,n_k)$ é p.r., então existe $t \in \mathbb{N}$ tal que
$f(n_1,\ldots,n_k) < A(t, \max_i n_i)$.

Um exemplo concreto

A função de Ackermann (1928):

```
def A(m,n):
    if m == 0:
      return n+1
      elsif n == 0:
        return A(m-1,1)
      else:
        return A(m-1, A(m,n-1))
```

Se $f(n_1,\ldots,n_k)$ é p.r., então existe $t \in \mathbb{N}$ tal que
$f(n_1,\ldots,n_k) < A(t, \max_i n_i)$.

Se A fosse p.r., existiria t para ela e $A(t,t) < A(t,t)$.

Minimalização

Seja g uma função $(k+1)$-ária, $k > 0$. A minimalização de g é a função k-ária f definida por:

$$f(n_1, \ldots, n_k) = \begin{cases} \text{o menor } m \text{ tal que } g(n_1, \ldots, n_k, m) = 1, \\ \quad\quad \text{se existir,} \\ 0 \quad\quad \text{caso contrário} \end{cases}$$

$f(n) = \begin{cases} n/2 & \text{se par} \\ \frac{3n+1}{2} & \text{se ímpar} \end{cases}$ $\quad g(n, m) = f^{(m)}(n) = f(f(\ldots f(n)))$

Minimalização

Seja g uma função $(k+1)$-ária, $k > 0$. A minimalização de g é a função k-ária f definida por:

$$f(n_1,\ldots,n_k) = \begin{cases} \text{o menor } m \text{ tal que } g(n_1,\ldots,n_k,m) = 1, \\ \qquad \text{se existir,} \\ 0 \qquad \text{caso contrário} \end{cases}$$

Notação: $\mu m[g(n_1,\ldots,n_k,m) = 1]$

Minimalização

Seja g uma função $(k+1)$-ária, $k > 0$. A minimalização de g é a função k-ária f definida por:

$$f(n_1,\ldots,n_k) = \begin{cases} \text{o menor } m \text{ tal que } g(n_1,\ldots,n_k,m) = 1, \\ \qquad \text{se existir,} \\ 0 \qquad \text{caso contrário} \end{cases}$$

Notação: $\mu m[g(n_1,\ldots,n_k,m) = 1]$

g é minimalizável se para todos $n_1,\ldots,n_k$ existe m tal que $g(n_1,\ldots,n_k,m) = 1$.

Funções μ-recursivas

Funções μ-recursivas

Uma função é μ-recursiva se pode ser obtida a partir das funções básicas por meio de composição, recursão e minimalização de funções minimalizáveis.

Funções μ-recursivas

Uma função é μ-recursiva se pode ser obtida a partir das funções básicas por meio de composição, recursão e minimalização de funções minimalizáveis.

"Código" para função p.r. obviamente define uma função. Para μ-recursivas isso não é claro em geral.

Funções μ-recursivas

Uma função é μ-recursiva se pode ser obtida a partir das funções básicas por meio de composição, recursão e minimalização de funções minimalizáveis.

"Código" para função p.r. obviamente define uma função. Para μ-recursivas isso não é claro em geral.

Não-exercício: a função de Ackermann é μ-recursiva.

μ-recursivas e Turing

Teorema

Uma função é μ-recursiva sse ela for recursiva (computável por máquina de Turing).

μ-recursivas e Turing

Teorema

Uma função é μ-recursiva sse ela for recursiva (computável por máquina de Turing).

μ-recursivas e Turing

Teorema

Uma função é μ-recursiva sse ela for recursiva (computável por máquina de Turing).

Dem: Só se: vamos mostrar que funções μ-recursivas são recursivas.
Básicas são óbvias, e composição também.

μ-recursivas e Turing

Teorema

Uma função é μ-recursiva sse ela for recursiva (computável por máquina de Turing).

Dem: Só se: vamos mostrar que funções μ-recursivas são recursivas.
Básicas são óbvias, e composição também.
Para o que segue, vamos lembrar que MTs conseguem implementar `while`.

while cond
corpo

Só se, cont.

Minimalização: $f(a,\ldots,b) = \mu m[g(a,\ldots,b,m) = 1]$

Só se, cont.

Minimalização: $f(a,\ldots,b) = \mu m[g(a,\ldots,b,m) = 1]$

```
def f(a,...,b):
   m = 0
   while g(a,...,b,m) != 1:
     m = m + 1
   return m
```

Só se, cont.

Minimalização: $f(a,\ldots,b) = \mu m[g(a,\ldots,b,m) = 1]$

```
def f(a,...,b):
    m = 0
    while g(a,...,b,m) != 1:
        m = m + 1
    return m
```

Definição recursiva:

Só se, cont.

Minimalização: $f(a,\ldots,b) = \mu m[g(a,\ldots,b,m) = 1]$

```
def f(a,...,b):
    m = 0
    while g(a,...,b,m) != 1:
      m = m + 1
    return m
```

Definição recursiva:

```
def f(a,...,b,m):
    v = g(a,...,b)
    for i in range(m):
      v = h(a,...,b,i,v)
    return v
```

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.
- Codificar configuração como número: análogo.

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.
- Codificar configuração como número: análogo.
- Difícil: mostrar que $\vdash_M$ é μ-recursiva. (é r.p.)

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.
- Codificar configuração como número: análogo.
- Difícil: mostrar que $\vdash_M$ é μ-recursiva. (é r.p.)
- Construir $\mathbf{iscomp}(m, n) = m$ codifica a sequência de configurações de (M, n).

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.
- Codificar configuração como número: análogo.
- Difícil: mostrar que $\vdash_M$ é μ-recursiva. (é r.p.)
- Construir $\textbf{iscomp}(m, n) = m$ codifica a sequência de configurações de (M, n).
- $\textbf{comp}(n) = \mu m[\textbf{iscomp}(m, n) = 1$
  $\&\&\text{estado final da computação é H}]$

Se

Se f é computada por uma MT M, então f é μ-recursiva.

- Precisa codificar M como um número: considerar uma descrição textual de M como representação em alguma base.
- Codificar configuração como número: análogo.
- Difícil: mostrar que $\vdash_M$ é μ-recursiva. (é r.p.)
- Construir $\mathbf{iscomp}(m, n)$ = m codifica a sequência de configurações de (M, n).
- $\mathbf{comp}(n) = \mu m[\mathbf{iscomp}(m, n) = 1$
  $\&\&\text{estado final da computação é H}]$
- Devolver fita final da computação m.