Maquinas de Turing

Rodrigo Gabriel Ferreira Soares

DEINFO - UFRPE

Julho, 2014

Motivação

- Autômatos finitos nem autômatos de pilha não são modelos gerais de computação
- Não são capazes de reconhecer $L = \{a^n b^n c^n : n > 0\}$
- Uma Maquina de Turing (MT) são capazes de reconhecer essa e outras linguagens mais complexas.

Definição

- Uma MT consiste de um controle finito, uma fita e um cabeçote que pode ser utilizado para efetuar leituras e gravações na fita.
- MTs não são suplantadas por outro modelo. Por mais elementares que essas maquinas pareçam ser, nenhuma tentativa de fortalece-las se mostra eficaz.
- Qualquer computação que pode ser realizada em uma maquina mais sofisticada também poderá ser realizada com uma MT.
- A computabilidade de funções numéricas (tal como $x + 2^x$) também esta ligada a MTs.

Definição

A unidade de controle opera em passos discretos, em cada um realiza duas operações

1. Levar unidade de controle para um novo estado
2. 1. Gravar um simbolo na celula apontada pelo cabeçote, substituindo algum simbolo la encontrado ou
   2. Mover o cabeçote de leitura/gravação para apontar uma celula a esquerda ou a direita na fita em relação a posição atual.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.
- A cadeia de entrada e gravada na previamente nas celulas mais a esquerda da fita, logo apos $\triangleright$.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.
- A cadeia de entrada e gravada na previamente nas celulas mais a esquerda da fita, logo apos $\triangleright$.
- O restante da fita e preenchido com espaços em branco $\sqcup$.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.
- A cadeia de entrada e gravada na previamente nas celulas mais a esquerda da fita, logo apos $\triangleright$.
- O restante da fita e preenchido com espaços em branco $\sqcup$.
- A MT e livre para modificar o conteúdo da fita, bem como para gravar nas celulas em branco.

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.
- A cadeia de entrada e gravada na previamente nas celulas mais a esquerda da fita, logo apos $\triangleright$.
- O restante da fita e preenchido com espaços em branco $\sqcup$.
- A MT e livre para modificar o conteúdo da fita, bem como para gravar nas celulas em branco.
- Dado que a MT pode apenas mover uma celula por vez, conclui-se que, apos uma computação finita, apenas um

Definição

- A fita e delimitada fisicamente a esquerda, mas estende-se indefinidamente para a direita
- A extremidade a esquerda e marcada com $\triangleright$.
- Sempre que a MT encontrar $\triangleright$, sua posição sera movida imediatamente a direita.
- Os símbolos $\leftarrow$ e $\rightarrow$ denotam movimentos para a esquerda e direita, respectivamente. E não fazem parte de nenhum alfabeto.
- A cadeia de entrada e gravada na previamente nas celulas mais a esquerda da fita, logo apos $\triangleright$.
- O restante da fita e preenchido com espaços em branco $\sqcup$.
- A MT e livre para modificar o conteúdo da fita, bem como para gravar nas celulas em branco.
- Dado que a MT pode apenas mover uma celula por vez, conclui-se que, apos uma computação finita, apenas um

Definição

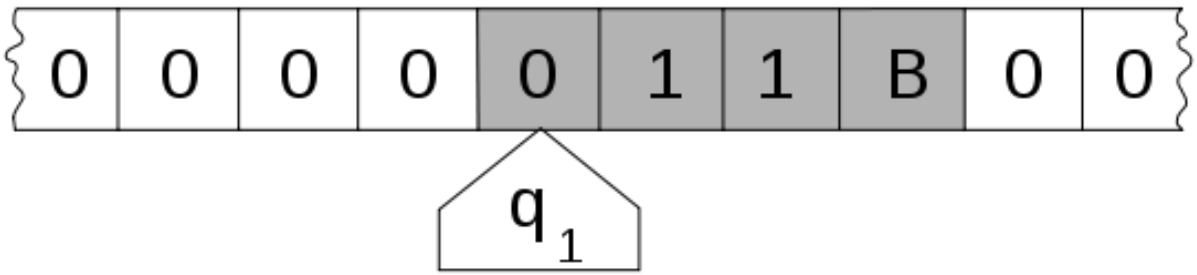

Figura : Maquina de Turing.

Definição

Definição formal

Uma MT e uma quíntupla $(K, \Sigma, \delta, s, H)$, onde
K e um conjunto de estados;
Σ e o alfabeto de entrada, que contem o simbolo de espaço em branco $\sqcup$ e o simbolo de extremidade esquerda $\triangleright$, mas que não contem os símbolos $\leftarrow$ e $\rightarrow$;
$s \in K$ e o estado inicial; $H \subseteq K$ e o conjunto de estados de parada;
δ, a função de transição, e uma função de $(K - H) \times \Sigma$ para $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$, tal que

1. $\forall q \in K - H$, se $\delta(q, \triangleright) = (p, b)$, então $b = \rightarrow$
2. $\forall q \in K - H$ e $a \in \Sigma$, se $\delta(q, a) = (p, b)$, então $b \neq \triangleright$

Exemplo 1

Considere a MT $M = (K, \Sigma, \delta, s, \{h\})$ onde $K = \{q_0, q_1, h\}$, $\Sigma = \{a, \sqcup, \triangleright\}$, $s = q_0$ e δ e dado por

q	σ	$\delta(q, \sigma)$
q_0	a	$(q_1, \sqcup)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	a	(q_0, a)
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$

Exemplo 2

Considere a MT $M = (K, \Sigma, \delta, s, \{h\})$ onde $K = \{q_0, h\}$, $\Sigma = \{a, \sqcup, \triangleright\}$, $s = q_0$ e δ e dado por

q	σ	$\delta(q, \sigma)$
q_0	a	$(q_0, \leftarrow)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$

Configuracao

Uma configuração da MT $M = (K, \Sigma, \delta, s, H)$ e algum membro de $K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{\epsilon\})$

- Toda configuração se inicia com o simbolo $\triangleright$ e nunca terminam com $\sqcup$
- Exemplos: $(q, \triangleright a, aba)$, $(h, \triangleright \sqcup \sqcup\sqcup, \sqcup a)$
- Não são configurações validas: $(q, \triangleright baa, abc\triangleright)$, $(q, \triangleright aa, ba)$
- Uma configuração cujo estado componente seja o estado de parada H sera chamada **configuração de parada**.

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$(q_1, \triangleright \underline{\sqcup} aaaa) \quad \vdash_M \quad (q_0, \triangleright \sqcup \underline{a} aaa) \tag{1}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned} (q_1, \triangleright \underline{\sqcup} aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a} aaa) && (1) \\ &\vdash_M (q_1, \triangleright \sqcup \underline{\sqcup} aaa) && (2) \end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned} (q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1) \\ &\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2) \\ &\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3) \end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup a) && (6)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup a) && (6)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \underline{a}) && (7)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup a) && (6)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \underline{a}) && (7)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup) && (8)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup a) && (6)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \underline{a}) && (7)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup) && (8)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) && (9)
\end{aligned}$$

Computação

Sejam duas configurações de M, $(q_1, w_1 a_1 u_1)$ e $(q_2, w_2 a_2 u_2)$, onde $a_1, a_2 \in \Sigma$.

$$(q_1, w_1 a_1 u_1) \vdash_M (q_2, w_2 a_2 u_2)$$

Seja o exemplo anterior, sua computação e como segue:

$$\begin{aligned}
(q_1, \triangleright \sqcup aaaa) &\vdash_M (q_0, \triangleright \sqcup \underline{a}aaa) && (1)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup aaa) && (2)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a}aa) && (3)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup aa) && (4)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \underline{a}a) && (5)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup a) && (6)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \underline{a}) && (7)\\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup) && (8)\\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) && (9)
\end{aligned}$$

Exemplos

$w \in \Sigma$ e w não contem espaços em branco

Exemplos

$w \in \Sigma$ e w não contem espaços em branco

- Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.

Exemplos

$w \in \Sigma$ e w não contem espaços em branco

- Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.
- Maquina de deslocamento a esquerda $S_{\leftarrow}$, que transforma $\sqcup w \underline{\sqcup}$ em $w \underline{\sqcup}$.

Exemplos

$w \in \Sigma$ e w não contem espaços em branco

- Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.
- Maquina de deslocamento a esquerda $S_{\leftarrow}$, que transforma $\sqcup w \underline{\sqcup}$ em $w \underline{\sqcup}$.
- Maquina M que apaga todos os a's em sua fita.

Exemplos

$w \in \Sigma$ e w não contem espaços em branco

- Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.
- Maquina de deslocamento a esquerda $S_{\leftarrow}$, que transforma $\sqcup w \underline{\sqcup}$ em $w \underline{\sqcup}$.
- Maquina M que apaga todos os a's em sua fita.

Computações usando maquinas de Turing

- MT substituem, como reconhecedores de linguagens, todos os tipos de autômatos vistos ate agora
- Para usar M como reconhecedor de linguagens, temos as seguintes convenções:
  - a cadeia de entrada w não tem espaços em branco e é gravada $\triangleright \sqcup w \sqcup$.
  - a configuração inicial de $M = (K, \Sigma, \delta, s, H)$ com entrada $w \in (\Sigma - \{\sqcup, \triangleright\})^\star$ é $(s, \triangleright \sqcup w)$.

Computações usando maquinas de Turing

Definição

Seja $M = (K, \Sigma, \delta, s, H)$, tal que $H = \{y, n\}$ (y e n denotam sim e não, respectivamente). Qualquer configuração que tenha y como estado componente é dita **configuração de aceitação**, enquanto uma configuração com n é dita **configuração de rejeição**.
Dizemos que M aceita uma entrada $w \in (\Sigma - \{\sqcup, \triangleright\})^\star$, se $(s, \triangleright\sqcup w)$ levar a uma configuração de aceitação. E M rejeita w, se $(s, \triangleright\sqcup w)$ leva a uma configuração de rejeição.
Seja $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ um **alfabeto de entrada** de M. Dizemos que M **decide** a linguagem $L \subseteq \Sigma_0^\star$, se, para qualquer cadeia $w \in \Sigma_0^\star$, se $w \in L$, então M aceita w; e se $w \notin L$, então M rejeita w.
Finalmente, dizemos que L é **recursiva** se houver uma MT que a decide.

Exemplo

Uma MT **decide** L se, quando iniciada com a entrada w, ela sempre para em um estado de parada que corresponde a resposta correta a entrada w, isto é, y se $w \in L$, e n se $w \notin L$.

Exemplo

Uma MT **decide** L se, quando iniciada com a entrada w, ela sempre para em um estado de parada que corresponde a resposta correta a entrada w, isto é, y se $w \in L$, e n se $w \notin L$.

- Considere a linguagem $L = \{a^n b^n c^n : n \leq 0\}$, que, ate aqui, não pode ser capturada por nenhum dos tipos de reconhecedores.

Exemplo

Uma MT **decide** L se, quando iniciada com a entrada w, ela sempre para em um estado de parada que corresponde a resposta correta a entrada w, isto é, y se $w \in L$, e n se $w \notin L$.

- Considere a linguagem $L = \{a^n b^n c^n : n \leq 0\}$, que, ate aqui, não pode ser capturada por nenhum dos tipos de reconhecedores.
- Nesse diagrama, utilizamos duas novas maquinas básicas: y, que leva a MT ao estado de aceitação y e n, que a leva ao estado de rejeição n.

Funções recursivas

- Ao contrario de outros reconhecedores de linguagens, MTs podem simplesmente não responder nem “sim” nem “não”, simplesmente deixando de parar.

Funções recursivas

- Ao contrario de outros reconhecedores de linguagens, MTs podem simplesmente não responder nem “sim” nem “não”, simplesmente deixando de parar.
- Dada uma MT, ela pode ou não decidir uma linguagem e não há um modo obvio de se verificar isso.

Funções recursivas

- Ao contrario de outros reconhecedores de linguagens, MTs podem simplesmente não responder nem “sim” nem “não”, simplesmente deixando de parar.
- Dada uma MT, ela pode ou não decidir uma linguagem e não há um modo obvio de se verificar isso.
- MTs podem produzir saídas mais elaboradas que “sim” ou “não”.

Funções recursivas

Definição

Seja $M = (K, \Sigma, \delta, s, \{h\})$ uma MT, $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ um alfabeto, e $w \in \Sigma_0^\star$. Suponha que M para ao operar sobre a entrada w e que $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, \triangleright \underline{\sqcup} z)$ para algum $z \in \Sigma_0^\star$. Então z é dito **saída de M para entrada** w e é denotado $M(w)$.
Seja f qualquer função $\Sigma_0^\star \to \Sigma_0^\star$. Dizemos que M computa f, se, para todo $w \in \Sigma_0^\star$, $M(w) = f(w)$. Assim, para todo $w \in \Sigma_0^\star$, M para e sua fita contem a cadeia $\triangleright \sqcup f(w)$.
Uma função f é dita **recursiva**, se houver uma MT que computa f.

Funções recursivas

Exemplo: A função $\kappa : \Sigma^\star \to \Sigma^\star$ definida como $\kappa(w) = ww$ pode ser computada pela maquina $CS_{\leftarrow}$.

Funções recursivas

Cadeias sobre o alfabeto binário $\{0,1\}^\star$ podem ser usadas para representar inteiros não-negativos na *notação binaria* usual. Qualquer cadeia $w = a_1 a_2 \ldots a_n \in \{0,1\}^\star$ representa o numero

Funções recursivas

Cadeias sobre o alfabeto binário $\{0,1\}^\star$ podem ser usadas para representar inteiros não-negativos na *notação binaria* usual. Qualquer cadeia $w = a_1 a_2 \ldots a_n \in \{0,1\}^\star$ representa o numero

$$\text{num}(w) = a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n.$$

Funções recursivas

Cadeias sobre o alfabeto binário $\{0,1\}^\star$ podem ser usadas para representar inteiros não-negativos na *notação binaria* usual. Qualquer cadeia $w = a_1 a_2 \dots a_n \in \{0,1\}^\star$ representa o numero

$$\text{num}(w) = a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n.$$

Qualquer natural pode ser representado de modo único por $0 \cup 1(0 \cup 1)^\star$.

Funções recursivas

Cadeias sobre o alfabeto binário $\{0,1\}^\star$ podem ser usadas para representar inteiros não-negativos na *notação binaria* usual. Qualquer cadeia $w = a_1 a_2 \ldots a_n \in \{0,1\}^\star$ representa o numero

$$\text{num}(w) = a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n.$$

Qualquer natural pode ser representado de modo único por $0 \cup 1(0 \cup 1)^\star$.
MTs que computam funções de $\{0,1\}^\star$ para $\{0,1\}^\star$ podem ser pensadas como funções que computam números naturais a partir de números naturais.

Funções recursivas

Cadeias sobre o alfabeto binário $\{0,1\}^\star$ podem ser usadas para representar inteiros não-negativos na *notação binaria* usual.
Qualquer cadeia $w = a_1 a_2 \ldots a_n \in \{0,1\}^\star$ representa o numero

$$\text{num}(w) = a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n.$$

Qualquer natural pode ser representado de modo único por $0 \cup 1(0 \cup 1)^\star$.
MTs que computam funções de $\{0,1\}^\star$ para $\{0,1\}^\star$ podem ser pensadas como funções que computam números naturais a partir de números naturais.
Funções de múltiplos argumentos podem ser executadas por MTs que computam funções de $\{0,1,;\}^\star$ para $\{0,1\}^\star$, onde ; é o simbolo para separar argumentos binários.

Funções recursivas

Definição

Seja $M = (K, \Sigma, \delta, s, \{h\})$ uma MT tal que $0, 1, ; \in \Sigma$ e seja f qualquer função de $\mathbb{N}^k$ para $\mathbb{N}$, para algum $k \geq 1$. Dizemos que M **computa** f, se para todo $w_1, \ldots, w_k \in 0 \cup 1(0 \cup 1)^\star$, $\text{num}(M(w_1; \ldots; w_k)) = f(\text{num}(w_1), \ldots, \text{num}(w_k))$.
Uma função $f : \mathbb{N}^k \to \mathbb{N}$ é dita **recursiva** se houver uma MT M que computa f.

Funções recursivas

- Exemplo: A função sucessora $\mathrm{suc}(n) = n + 1$.

Funções recursivas

- Exemplo: A função sucessora $\text{suc}(n) = n + 1$.
- O preço que devemos pagar pela capacidade computacional das MTs é o fato de que não podermos afirmar se uma MT computa de fato uma dada função, isto é, se ela para para qualquer cadeia de entrada.

Funções recursivamente enumeráveis

Pelo fato de uma MT decidir uma linguagem ou computar uma função, pode-se considerar razoável interpreta-la como um **algoritmo**, que executa corretamente uma tarefa computacional.

Funções recursivamente enumeráveis

Pelo fato de uma MT decidir uma linguagem ou computar uma função, pode-se considerar razoável interpreta-la como um **algoritmo**, que executa corretamente uma tarefa computacional.

Definição

Seja $M = (K, \Sigma, \delta, s, H)$ uma MT. Seja $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ um alfabeto e $L \subseteq \Sigma_0^\star$ uma linguagem. Dizemos que M **semidecide** L se, para qualquer cadeia $w \in \Sigma_0^\star$, $w \in L$ se e somente se M para em resposta a entrada w. Uma linguagem L é **recursivamente enumerável** se e somente se existir uma MT que semidecide L.

Funções recursivamente enumeráveis

- Quando M é acionada com $w \in L$, exige-se que pare ao final. Não importa a configuração de parada que M atinge, bastando que M pare.

Funções recursivamente enumeráveis

- Quando M é acionada com $w \in L$, exige-se que pare ao final. Não importa a configuração de parada que M atinge, bastando que M pare.
- Se, entretanto, $w \in \Sigma_0^\star$, então M nunca deve atingir o estado de parada. Ela continuarah sua computação indefinidamente.

Funções recursivamente enumeráveis

- Quando M é acionada com $w \in L$, exige-se que pare ao final. Não importa a configuração de parada que M atinge, bastando que M pare.
- Se, entretanto, $w \in \Sigma_0^\star$, então M nunca deve atingir o estado de parada. Ela continuarah sua computação indefinidamente.
- δ é uma função completamente definida.

Funções recursivamente enumeráveis

- Quando M é acionada com $w \in L$, exige-se que pare ao final. Não importa a configuração de parada que M atinge, bastando que M pare.
- Se, entretanto, $w \in \Sigma_0^\star$, então M nunca deve atingir o estado de parada. Ela continuarah sua computação indefinidamente.
- δ é uma função completamente definida.
- Exemplo: Seja
  $L = \{w \in \{a, b\}^\star : w \text{ contem pelo menos um } a\}$. Então L é semidecidida por uma MT a seguir.

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.
- "Prosseguir indefinidamente percorrendo espaços em branco" é apenas uma das maneiras pelas quais M pode deixar de parar. Exemplo: $\delta(q, a) = (q, a)$.

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.
- "Prosseguir indefinidamente percorrendo espaços em branco" é apenas uma das maneiras pelas quais M pode deixar de parar. Exemplo: $\delta(q, a) = (q, a)$.
- Um AFD sempre para e verificamos seu estado de parada (algoritmo).

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.
- "Prosseguir indefinidamente percorrendo espaços em branco" é apenas uma das maneiras pelas quais M pode deixar de parar. Exemplo: $\delta(q, a) = (q, a)$.
- Um AFD sempre para e verificamos seu estado de parada (algoritmo).
- Em contraste, uma MT que semidecide uma linguagem L pode não ser adequada para verificar se $w \in L$.

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.
- "Prosseguir indefinidamente percorrendo espaços em branco" é apenas uma das maneiras pelas quais M pode deixar de parar. Exemplo: $\delta(q, a) = (q, a)$.
- Um AFD sempre para e verificamos seu estado de parada (algoritmo).
- Em contraste, uma MT que semidecide uma linguagem L pode não ser adequada para verificar se $w \in L$.
- Se $w \notin L$, então nunca saberemos quando teremos esperado o suficiente para obter uma resposta.

Funções recursivamente enumeráveis

- Escreve-se $M(w) = \nearrow$ se M falhar em parar em resposta ah entrada w. $M(w) = \nearrow$ se e somente se $w \notin L$.
- "Prosseguir indefinidamente percorrendo espaços em branco" é apenas uma das maneiras pelas quais M pode deixar de parar. Exemplo: $\delta(q, a) = (q, a)$.
- Um AFD sempre para e verificamos seu estado de parada (algoritmo).
- Em contraste, uma MT que semidecide uma linguagem L pode não ser adequada para verificar se $w \in L$.
- Se $w \notin L$, então nunca saberemos quando teremos esperado o suficiente para obter uma resposta.
- MTs que semidecidem linguagens não são algoritmos.

Funções recursivamente enumeráveis

- Qualquer linguagem recursiva também é recursivamente enumerável.

Funções recursivamente enumeráveis

- Qualquer linguagem recursiva também é recursivamente enumerável.
- Podemos converter uma MT que decide L em uma MT que a semidecide através da transformação do estado de rejeição n em uma estado que não seja de parada e se garanta que a MT não irah parar.

Funções recursivamente enumeráveis

- Qualquer linguagem recursiva também é recursivamente enumerável.
- Podemos converter uma MT que decide L em uma MT que a semidecide através da transformação do estado de rejeição n em uma estado que não seja de parada e se garanta que a MT não irah parar.
- Teorema: Se uma linguagem é recursiva, então ela é recursivamente enumerável.

Funções recursivamente enumeráveis

- Qualquer linguagem recursiva também é recursivamente enumerável.
- Podemos converter uma MT que decide L em uma MT que a semidecide através da transformação do estado de rejeição n em uma estado que não seja de parada e se garanta que a MT não irah parar.
- Teorema: Se uma linguagem é recursiva, então ela é recursivamente enumerável.
- Há linguagens recursivamente enumeráveis que não são recursivas.

Extensões da Maquina de Turing

- A fim de melhorar o entendimento sobre o poder computacional das MTs, devemos estudar extensões do modelo em varias direções.

Extensões da Maquina de Turing

- A fim de melhorar o entendimento sobre o poder computacional das MTs, devemos estudar extensões do modelo em varias direções.
- Em cada caso, os recursos adicionados nada acrescentam as classes de funções computáveis e linguagens decidíveis.

Extensões da Maquina de Turing

- A fim de melhorar o entendimento sobre o poder computacional das MTs, devemos estudar extensões do modelo em varias direções.
- Em cada caso, os recursos adicionados nada acrescentam as classes de funções computáveis e linguagens decidíveis.
- Esses melhoramentos podem ser simulados pelo modelo de MT convencional.

Extensões da Maquina de Turing

- A fim de melhorar o entendimento sobre o poder computacional das MTs, devemos estudar extensões do modelo em varias direções.
- Em cada caso, os recursos adicionados nada acrescentam as classes de funções computáveis e linguagens decidíveis.
- Esses melhoramentos podem ser simulados pelo modelo de MT convencional.
- MT é, de fato, o modelo computacional definitivo.

Extensões da Maquina de Turing

- A fim de melhorar o entendimento sobre o poder computacional das MTs, devemos estudar extensões do modelo em varias direções.
- Em cada caso, os recursos adicionados nada acrescentam as classes de funções computáveis e linguagens decidíveis.
- Esses melhoramentos podem ser simulados pelo modelo de MT convencional.
- MT é, de fato, o modelo computacional definitivo.
- Estamos livres para explorar novos recursos para problemas particulares, sabendo que nossa dependência a esses recursos podem ser eliminados.

Fitas múltiplas

- Cada fita é conectada ao controle finito por meio de um cabeçote correspondente.

Fitas múltiplas

- Cada fita é conectada ao controle finito por meio de um cabeçote correspondente.
- A computação ocorre em todas as k fitas.

Fitas múltiplas

- Cada fita é conectada ao controle finito por meio de um cabeçote correspondente.
- A computação ocorre em todas as k fitas.
- Uma configuração deve incluir informações sobre o conteúdo de todas as fitas.

Fitas múltiplas

- Cada fita é conectada ao controle finito por meio de um cabeçote correspondente.
- A computação ocorre em todas as k fitas.
- Uma configuração deve incluir informações sobre o conteúdo de todas as fitas.
- $(q, (w_1 \underline{a_1} u_1, \ldots, w_k \underline{a_k} u_k))$ é uma configuração e $\delta(p, (a_1, \ldots, a_k)) = (b_1, \ldots, b_k)$ é uma transição.

Fitas múltiplas

- Exemplo: Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.

Fitas múltiplas

- Exemplo: Maquina de copiar C, que transforma $\sqcup w \underline{\sqcup}$ em $\sqcup w \sqcup w \underline{\sqcup}$.
- Corolário: Qualquer função computada ou linguagem decidida ou semidecidida por uma MT de k fitas também é respectivamente computada, decidida ou semidecidida por uma MT convencional.

Fita infinita em ambas as direções

- Todas as celulas contem espaços em branco, exceto as que contem a cadeia de entrada.

Fita infinita em ambas as direções

- Todas as celulas contem espaços em branco, exceto as que contem a cadeia de entrada.
- $\triangleright$ é desnecessário.

Fita infinita em ambas as direções

- Todas as celulas contem espaços em branco, exceto as que contem a cadeia de entrada.
- $\triangleright$ é desnecessário.
- Não proporcionam poder adicional, pois pode ser simulada por uma MT convencional.

Múltiplos cabeçotes

- Uma soh fita e vários cabeçotes.

Múltiplos cabeçotes

- Uma soh fita e vários cabeçotes.
- Todos os cabeçotes lêem os símbolos correspondentes e podem mover-se ou escrever independentemente.

Múltiplos cabeçotes

- Uma soh fita e vários cabeçotes.
- Todos os cabeçotes lêem os símbolos correspondentes e podem mover-se ou escrever independentemente.
- Assim como o uso de fitas múltiplas, a aplicação de múltiplos cabeçotes podem simplificar drasticamente a construção de uma MT.

Múltiplos cabeçotes

- Uma soh fita e vários cabeçotes.
- Todos os cabeçotes lêem os símbolos correspondentes e podem mover-se ou escrever independentemente.
- Assim como o uso de fitas múltiplas, a aplicação de múltiplos cabeçotes podem simplificar drasticamente a construção de uma MT.
- Uma versão da maquina de copiar C poderia funcionar de modo mais natural do que a versão com um soh cabeçote.

Múltiplos cabeçotes

- Uma soh fita e vários cabeçotes.
- Todos os cabeçotes lêem os símbolos correspondentes e podem mover-se ou escrever independentemente.
- Assim como o uso de fitas múltiplas, a aplicação de múltiplos cabeçotes podem simplificar drasticamente a construção de uma MT.
- Uma versão da maquina de copiar C poderia funcionar de modo mais natural do que a versão com um soh cabeçote.
- Não proporcionam poder adicional.

Fita bidimensional

- Sua “fita” é uma grade bidimensional infinita.

Fita bidimensional

- Sua "fita" é uma grade bidimensional infinita.
- Pode permitir dimensionalidade mais alta.

Fita bidimensional

- Sua "fita" é uma grade bidimensional infinita.
- Pode permitir dimensionalidade mais alta.
- Extensões da MT podem ser combinadas:
  - MTs com varias fitas, todas ou algumas sendo infinitas nas duas direções, com mais de um cabeçote ou ainda podem ser multidimensionais.

Fita bidimensional

- Sua "fita" é uma grade bidimensional infinita.
- Pode permitir dimensionalidade mais alta.
- Extensões da MT podem ser combinadas:
  - MTs com varias fitas, todas ou algumas sendo infinitas nas duas direções, com mais de um cabeçote ou ainda podem ser multidimensionais.
- Qualquer linguagem decidida ou semidecidida e qualquer função computadas por MTs com varias fitas, vários cabeçotes, fitas infinitas nas duas direções ou fitas multidimensionais, podem ser decididas, semidecididas ou computadas, respectivamente, por uma MT convencional.

Maquinas de Turing não-determinísticas

- Aplicamos o não-determinismo ahs MTs.

Maquinas de Turing não-determinísticas

- Aplicamos o não-determinismo ahs MTs.
- MTs podem ter, para certas combinações de símbolos e estados, mais de uma escolha de procedimento possível.

Maquinas de Turing não-determinísticas

- Aplicamos o não-determinismo ahs MTs.
- MTs podem ter, para certas combinações de símbolos e estados, mais de uma escolha de procedimento possível.
- A relação $\vdash_M$ não precisa mais ter um único valor: uma configuração pode produzir varias outras em um passo.

Maquinas de Turing não-determinísticas

- Aplicamos o não-determinismo ahs MTs.
- MTs podem ter, para certas combinações de símbolos e estados, mais de uma escolha de procedimento possível.
- A relação $\vdash_M$ não precisa mais ter um único valor: uma configuração pode produzir varias outras em um passo.
- Assim como ocorre com os demais recursos, o não-determinismo pode ser eliminado das MTs, ou seja, há MT determinística equivalente para toda MT não-determinística.

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).
- Semelhante a GLC, exceto que os lados esquerdos das regras podem consistir de mais de um simbolo.

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).
- Semelhante a GLC, exceto que os lados esquerdos das regras podem consistir de mais de um simbolo.
- O lado esquerdo pode ter qualquer cadeia de terminais e não-terminais, que contenha pelo menos um não-terminal.

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).
- Semelhante a GLC, exceto que os lados esquerdos das regras podem consistir de mais de um simbolo.
- O lado esquerdo pode ter qualquer cadeia de terminais e não-terminais, que contenha pelo menos um não-terminal.
- O produto final é também uma cadeia de terminais.

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).
- Semelhante a GLC, exceto que os lados esquerdos das regras podem consistir de mais de um simbolo.
- O lado esquerdo pode ter qualquer cadeia de terminais e não-terminais, que contenha pelo menos um não-terminal.
- O produto final é também uma cadeia de terminais.
- Uma regra pode assumir a forma $uAv \rightarrow uwv$, que pode ser lida "substituir A por w no contexto de u e v".

Gramaticas

- MTs são **reconhecedores de linguagens**. A partir delas, surgiram as linguagens recursivas e recursivamente enumeráveis.
- Novo tipo de **gerador de linguagem**: uma generalização da gramatica livre de contexto, chamada de **gramatica** (ou **gramatica irrestrita**).
- Semelhante a GLC, exceto que os lados esquerdos das regras podem consistir de mais de um simbolo.
- O lado esquerdo pode ter qualquer cadeia de terminais e não-terminais, que contenha pelo menos um não-terminal.
- O produto final é também uma cadeia de terminais.
- Uma regra pode assumir a forma $uAv \rightarrow uwv$, que pode ser lida "substituir A por w no contexto de u e v".
- Substituição depende do contexto.

Exemplo

$G = (V, \Sigma, R, S)$ gera a linguagem $L = \{a^n b^n c^n : n \geq 1\}$, onde $V = \{S, a, b, c, A, B, C, T_a, T_b, T_c\}$, $\Sigma = \{a, b, c\}$ e

$$
\begin{aligned}
R \quad = \quad & \{S \to ABCS && (11)\\
& S \to T_c && (12)\\
& CA \to AC && (13)\\
& BA \to AB && (14)\\
& CB \to BC && (15)\\
& CT_c \to T_c c && (16)\\
& CT_c \to T_b c && (17)\\
& BT_b \to T_b b && (18)\\
& BT_b \to T_a b && (19)\\
& AT_a \to T_a a && (20)\\
& T_a \to \epsilon\}. && (21)
\end{aligned}
$$

Gramaticas

- Teorema: Uma linguagem eh gerada por uma gramatica se e somente se ela for recursivamente enumerável.

Gramaticas

- Teorema: Uma linguagem eh gerada por uma gramatica se e somente se ela for recursivamente enumerável.
- Propriedade da semidecisão em comum com MTs
  - Se uma cadeia puder se gerada por uma gramatica, podemos pacientemente procurar todas as possíveis derivações.

Gramaticas

- Teorema: Uma linguagem eh gerada por uma gramatica se e somente se ela for recursivamente enumerável.
- Propriedade da semidecisão em comum com MTs
  - Se uma cadeia puder se gerada por uma gramatica, podemos pacientemente procurar todas as possíveis derivações.
  - Mas se não existir nenhuma derivação, esse processo continuará indefinidamente, sem prover nenhuma informação útil.

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria zero$(n_1, \ldots, n_k) = 0$

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria zero$(n_1, \ldots, n_k) = 0$
  2. Funcao identidade k-aria id$_j(n_1, \ldots, n_k) = n_j$

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria $\text{zero}(n_1, \ldots, n_k) = 0$
  2. Funcao identidade k-aria $\text{id}_j(n_1, \ldots, n_k) = n_j$
  3. Funcao sucessor $\text{suc}(n) = n + 1$

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria $\text{zero}(n_1, \ldots, n_k) = 0$
  2. Funcao identidade k-aria $\text{id}_j(n_1, \ldots, n_k) = n_j$
  3. Funcao sucessor $\text{suc}(n) = n + 1$
- Combinando funcoes

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria $\text{zero}(n_1, \ldots, n_k) = 0$
  2. Funcao identidade k-aria $\text{id}_j(n_1, \ldots, n_k) = n_j$
  3. Funcao sucessor $\text{suc}(n) = n + 1$
- Combinando funcoes
  1. Composicao:
     $f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l))$

Funções numéricas

- Exemplo: $f(x, y) = x^2 y + 5x^{y+2}$ eh construída com composição de funções: adição, multiplicação, exponenciação.
- Funcoes basicas
  1. Funcao nula k-aria $\text{zero}(n_1, \ldots, n_k) = 0$
  2. Funcao identidade k-aria $\text{id}_j(n_1, \ldots, n_k) = n_j$
  3. Funcao sucessor $\text{suc}(n) = n + 1$
- Combinando funcoes
  1. Composicao:
     $f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l))$
  2. Definicao recursiva:

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k)$$

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

Funções recursivas primitivas

- Sao **funcoes recursivas primitivas** todas as funcoes basicas e tambem todas as funcoes que podem ser obtidas a partir delas por qualquer numero de aplicacoes sucessivas de composicao e definicao recursiva.

Funções recursivas primitivas

- Sao **funcoes recursivas primitivas** todas as funcoes basicas e tambem todas as funcoes que podem ser obtidas a partir delas por qualquer numero de aplicacoes sucessivas de composicao e definicao recursiva.
- Exemplo: A funcao mais2, definida como mais2(n) $= n + 2$ eh recursiva primitiva, pois pode ser obtida a partir de suc com uma composicao com ela mesma.

Funções recursivas primitivas

- Sao **funcoes recursivas primitivas** todas as funcoes basicas e tambem todas as funcoes que podem ser obtidas a partir delas por qualquer numero de aplicacoes sucessivas de composicao e definicao recursiva.
- Exemplo: A funcao mais2, definida como mais2(n) $= n + 2$ eh recursiva primitiva, pois pode ser obtida a partir de suc com uma composicao com ela mesma.
- Dizemos que uma funcao eh μ-**recursiva** se ela puder ser obtida a partir de funcoes basicas pela aplicacao das operacoes de composicao, definicao recursiva e *minimizacao de funcoes minimizaveis*.

Funções recursivas primitivas

- Sao **funcoes recursivas primitivas** todas as funcoes basicas e tambem todas as funcoes que podem ser obtidas a partir delas por qualquer numero de aplicacoes sucessivas de composicao e definicao recursiva.
- Exemplo: A funcao mais2, definida como mais2(n) $= n + 2$ eh recursiva primitiva, pois pode ser obtida a partir de suc com uma composicao com ela mesma.
- Dizemos que uma funcao eh μ-**recursiva** se ela puder ser obtida a partir de funcoes basicas pela aplicacao das operacoes de composicao, definicao recursiva e *minimizacao de funcoes minimizaveis*.
- Teorema: A funcao $f : \mathbb{N}^k \to \mathbb{N}$ eh μ-recursiva se e somente se ela for recursiva.