Equivalência entre Máquina de Wang-Post e Máquina de Turing

João Lucas Pereira de Santana - 20071230

Teoria Da Computação - 2009/1

1 Apresentação da Máquina

Considere uma máquina fictícia B com as seguintes características:

- Armazenamento interno ilimitado.
- Fita de armazenamento serial dividida em células (*square*) que se extende em ambas direções.
- Elemento controlador.
- Cabeça de leitura e escrita que em um dado momento scanea um, e somente um *square* da fita

- A cabeça de leitura pode mover para esquerda, direita, ou marcar o *square* scaneado. Pode ainda ir para outro *square* se o *square* atualmente scaneado está marcado ou não.

A cada momento, a próxima instruçao da máquina é determinada pelo passo atual do programa juntamente com o conteúdo da célula scaneada.

São possíveis 4 tipos de operações na fita:

- $\rightarrow$ Cabeça de leitura move um *square* para a direita.
- $\leftarrow$ Cabeça de leitura move um *square* para a esquerda.
- $*$: Marca a posição atual.
- Cn : Elemento controlador pula para outro passo do programa.

Exemplo de programa para parar no *square* branco mais próximo à direita do *square* incialmente scaneado:

$$1.*,\ 2.\rightarrow,\ 3.C2,\ 4.\rightarrow,\ 5.\leftarrow$$

Um programa em uma máquina B, doravante *B-Machine*, poder ser considerado como um conjunto de pares ordenados tal que existe um inteiro positivo $k(k > 2)$ para o qual:

(a) para cada n, n ocorre no primeiro membro se, e somente se, $1 \leq n \leq k$;

(b) o segundo membro de cada par é $*$, ou $\rightarrow$, ou $\leftarrow$, ou Cn, com $1 \leq n \leq k-1$;

(c) existem os pares $\{\langle k-1, \rightarrow\rangle, \langle k, \leftarrow\rangle\}$;

De acordo com a definição anterior, o exemplo pode ser representado por:

$$\{\langle 1, *\rangle, \langle 2, \rightarrow\rangle\ \langle 3, 2\rangle, \langle 4, \rightarrow\rangle\langle 5, \leftarrow\rangle\}$$

2 Todas as funções computáveis por uma *B-Machine* são recursivas

Para simplificar as considerações, é assumido o seguinte:

i) A entrada inicial da fita contém um finito número de *square* marcados;

ii) No inicio de cada programa, a cabeça de leitura scanea o sexto *square* branco à direta do *square* mais à direta marcado.

Por outro lado, subrotinas começam em qualquer posição dentro da porção contendo marcações mais os seis brancos, e terminam similarmente.
De i) segue que a cada momento, existem finitos *square* marcados na fita.

Para um programa Π, sua configuração a cada momento é dada por:

a) Conteúdo da fita;

b) A posição e o conteúdo do *square* scaneado;

c) Instrução do programa que está sendo executada.

Esses 3 fatos somados ao programa determinam o *Complete Stantaneous State* de uma *B-Machine*, daqui para frente, Configuração da *B-Machine.* Esta configuração pode ser representada por números da seguinte forma:
$\rightarrow$ por 1;
$\leftarrow$ por 2;
$*$ por 3
n por $n + 3$.

Com esta convenção, o exemplo $\{\langle 1, *\rangle, \langle 2, \rightarrow\rangle\ \langle 3, 2\rangle, \langle 4, \rightarrow\rangle\langle 5, \leftarrow\rangle\}$ pode ser representado por:

$$2^3 \cdot 3^1 \cdot 5^5 \cdot 7^1 \cdot 11^2$$

$lh(\pi)$ é o comprimento de Π.
$(\pi)_i$ é o expoente do i-ésimo primo na fatoração de Π
Onde $lh(\pi)$ e $(\pi)_i$ são funções recursivas (Kleene [2])
O conteúdo da fita e o *square* scaneado podem ser representados por um número na forma

$$3^a \cdot 3^b \cdot 7^c$$

onde $b = 0$ ou 1 se o *square* scaneado está branco ou marcado, respectivamente. a e c representam, respectivamente, o conteúdo da fita à esqueda e à direita do *square* scaneado, de forma que o i-ésimo digito (da direita para esquerda) de a (resp. c) é 0 ou 1 dependendo se o $i$ésimo *square* à esquerda (resp. direita) é branco ou marcado. Se, por exemplo a porção marcada da fita é

*		*	*			*	*

e a cabeça de leitura scanea o quarto (da esquerda para direita) *square*, o número é $3^{101} \cdot 5 \cdot 7^{1100}$. Usando esta notação, a configuraçã da fita pode ser representada a cada momento pelo número

$$2^\pi \cdot 3^a \cdot 5^b \cdot 7^c \cdot 11^i$$

quando o programa tem o número π, o conteúdo da fita com o *square* scaneado o número $3^a \cdot 3^b \cdot 7^c$, e o passo da execução é i.
Se a *B-Machine* tem a configuração $2^\pi \cdot 3^a \cdot 5^b \cdot 7^c \cdot 11^i$, $i < lh(\pi)$, o próximo passo da execução é dado por:

i) se $(\pi)_i = 1$, $2^\pi \cdot 3^{10a+b} \cdot 5^{c'} \cdot 7^{c/10} \cdot 11^{i+1}$, $c' = c \; mod \; 10$;

ii) se $(\pi)_i = 2$, $2^\pi \cdot 3^{a/10} \cdot 5^{a'} \cdot 7^{10c+b} \cdot 11^{i+1}$, $a' = a \; mod \; 10$;

iii) se $(\pi)_i = 3$, $2^\pi \cdot 3^a \cdot 5 \cdot 7^c \cdot 11^{i+1}$;

iv) $(\pi)_i = 3 + j$, $2^\pi \cdot 3^a \cdot 5^b \cdot 7^c \cdot 11^{i+1}$ se $b = 0$, e $2^\pi \cdot 3^a \cdot 5^b \cdot 7^c \cdot 11^j$ se $b = 1$.

Com esta configuração, quando a *B-Machine* começar a execução, todos os outros estados ser¸ao determinados.

Desses dados segue que:

$\theta_\pi(\iota, t)$ retorna o número de configuração da *B-Machine* no momento t, para uma fita com entrada ι.

$\tau_\pi(\iota) = \mu_t[(\theta_\pi(\iota, t))_5 = lh(\pi)]$ retorna o menor número t, tal que, para uma dada entrada i, a *B-Machine* estará executando a última linha do programa Π no tempo t.

A função recursiva $\theta_\pi(\iota, \tau_\pi(i))/(2^\pi \cdot 11^{lh(\pi)})$ é a função determinada pelo programa Π, uma vez que ela retorna para cada entrada i, a correspondente saída que resulta da execução do programa Π

As definições explícitas dessas funções foram omitidas porque funções similares são encontradas em Kleene [2], pp. 374-376.

Disso segue que a função determinada por qualquer programa da *B-Machine* é uma função recursiva. Pode-se declarar como resultado dizendo que todas as funções computáveis por uma *B-Machine* são recursivas. Isto é uma ligeira generalização do conhecido teorema que diz que todas as funções Turing computáveis são recursivas (Kleene [2], p.374). Esta generalização consiste na renúncia da restrição de que os estados inicial e final da fita devem estar de uma certa forma predefinida, a qual é convencionada de acordo com a necessidade, como representação positiva de inteiros ou n-tuplas de inteiros.

3 Toda função recursiva é B-Computável

Para provar que todas as funções são recursivas são B-computáveis, será usada indução por consequência: as funções iniciais são B-computáveis, dada uma classe de B-computáveis funções, então funções que derivam delas por um esquema de recursão são também B-computáveis.

Uma vez que não estamos usando apagar *squares* da fita, chamaremos uma sequência alternada de P-*squares* (*principal squares*) e outra sequência de A-*squares* (*auxiliary squares*). Entradas e saídas na fita são determinadas pelo conteúdo dos P-*squares*.

Para representar um inteiro n, será usada uma *string* de n pares de *squares* (*representação numérica*), a qual começa (esquerda para direita) com um P-*square* e termina com um A-*square* tal que todos os P-*squares* na *string* são marcados e que cada P-*square* que antecede e sucede a *string* são ambos brancos. Um expressão nemérica é chamada *clean* se todos os A-*squares* dela são brancos

Os argumentos $x_1, ..., x_m$ de uma função $f(x_1, ..., x_m)$ serão representados em qualquer lugar da fita da esquerda para a direita, primeiro pelo número x_1, seguido por branco P-*square* e um branco A-*square*, seguido pelo número x_2, etc.

Se $f(x_1, ..., x_m)$ não está definida, a máquina pode nunca parar (circular) ou parar em um ponto quando não há uma expressão numérica na fita tal que todos os *squares* à sua direta são brancos.

Condições para execução do programa:

(a) nã há mais do que dois P-*squares* brancos entre duas representações numéricas,

(b) cada P-*square* marcado é sempre parte de uma representação numérica;

(c) não há A-*squares* marcados fora da porção marcada da fita até o momento;
De (a) e (c) segue que a cada estágio da execução da máquina, nunca aparecem mais do que 5 *squares* brancos entre dois *squares* marcados.
big gap : 5 *squares* brancos entre 2 *squares* marcados.
gap : 3 *squares* brancos entre 2 *squares* marcados.

(d) para quaisquer valores $x_1, ..., x_m$, ou a máquina nunca para ou para e o conteúdo final da fita contém uma expresssão numérica na forma correta;

(e) se a máquina para para uma dada entrada $x_1, ..., x_m$, a última representação numérica sera *clean* (i.e., sem A-*squares* marcados);

(f) na parada, todos os *big gaps* serão eliminados;

(g) na parada, a cabeça de leitura terminará scaneando o quinto *square* branco à direita da última expressão numérica, isto é, o sexto *square* branco após o último *square* marcado (cabeça de leitura termina *scaneando a entrada*);

A condição (e) assegura que é permitido utilizar o resultado de uma função no cálculo de outra função. A condição (f) ajuda a localizar a representação numérica anterior na fita.

Subrotina X: Encontra o mais próximo *square* marcado à esquerda do *square* atual, e termina no *square* à direita.

$$1.C14,\ 2.\leftarrow,\ 3.C14,\ 4.\leftarrow,\ 5.C14,\ 6.\leftarrow,\ 7.C14,\ 8.\leftarrow,\ 9.C14,\ 10.\leftarrow,\ 11.C14,\ 12.\leftarrow,\ 13.C14,\ 14.\rightarrow,\ 15.\rightarrow,\ 16.\leftarrow .$$

Subrotina Y: Se a cabeça de leitura está na região que contém todos os *squares* marcados mais os 6 brancos à direita do último *square* marcado, Y encontra o último *square* marcado e termina *scaneando a entrada*

$$1.X,\ 2.\leftarrow,\ 3.\rightarrow,\ 4.C3,\ 5.\rightarrow,\ 6.C3,\ 7.\rightarrow,\ 8.C3,\ 9.\rightarrow,\ 10.C3,\ 11.\rightarrow,\ 12.C3,\ 13.\rightarrow,\ 14.C3,\ 15.\rightarrow,\ 16.\leftarrow .$$

Subrotina A: Adiciona 1 ao último número na fita.

$$1.Y,\ 2.\leftarrow^4,\ 3.*,\ 4.\rightarrow,\ 5.\leftarrow .$$

ou simplesmente

$$Y,\ \leftarrow^4,\ *,$$

($\leftarrow^n$ significa $\leftarrow$ repetido n vezes.)

Subrotina H: Encontra o mais próximo *big gap* à esquerda do *square* scaneado e termina lendo o *square* do meio do *big gap*.

$$1.C4,\ 2.\leftarrow,\ 3.C1,\ 4.\leftarrow,\ 5.C4,\ 6.\leftarrow,\ 7.C4,\ 8.\leftarrow,\ 9.C4,\ 10.\leftarrow,\ 11.C4,\ 12.\leftarrow,\ 13.C4,\ 14.\rightarrow^2,\ (15.\rightarrow,\ 16.\leftarrow .)$$

A prova de que todas as funções recursivas são B-computáveis depende da possibilidade de conseguir todas as funções a partir de seis *schemas*. Eles serão definidos a seguir.

Schema (I) $\varphi(x) = x + 1$.

Schema (II) $\varphi(x_1, ..., x_n) = q_i$, q constante.

Schema (III) $\varphi(x_1, ..., x_n) = x_i$, i constante entre $1, ..., n$.

Prova I Para provar o *schema* I, deve-se encontrar um programa que encontre o último *square* marcado na representação do número x, mover seis *squares* para a direita (deixando um *big gap*), copiar o parâmetro x e depois adicionar 1.

Subrotina I_m: Assumindo que foi obtida esta rotina para cópia. Inicia no *square* scaneado, copia nos sucessivos alternados *squares* o m-ésimo numero (contando da direita para a esquerda) que se situa à esquerda do *big gap* que precede (à esquerda de) o *square* scaneado. Termina *scaneando a entrada.*

Subrotina Z : Elimina o último *big gap* e deixa a máquina *scaneando a entrada.*

$$Z : 1.H,\ 2.*,\ 3.Y, \text{ ou simplesmente, } H,\ *,\ Y$$

Usando Z, A, Y e I_m (a ser introduzida formalmente), pode-se provar que todas as funções definidas pelos *schemas* I, II e III são B-computáveis.

(I) $Y,\ I_1,\ A,\ Z$

(II) $Y,\ *,\ A^{q-1},\ Z$, quando $q = 1$, simplesmente $Y,\ *,\ Z$.

(III) $Y,\ I_{n-i+1},\ Z$.

Para definir a subrotina I_m precisamos antes definir algumas outras subrotinas, são elas.

Subrotina D: Quando scaneando um P-*square*, encontra e scanea o último P-*square* da representação numérica predecessora mais próxima. Poder ser repetida para que D^m retorne o útimo (marcado) P-*square* da i-ésima (da direita para a esquerda) representação numérica que precede o *square* scaneado.

$$D : 1.C4,\ 2. \leftarrow^2,\ 3.C6,\ 4. \leftarrow^2,\ 5.C1,\ 6. \rightarrow,\ 7. \leftarrow$$

Subrotina G: Encontra dois sucessivos A-*squares* brancos mais próximos à esquerda do *square* scaneado e termina scaneando o P-*square* entre eles.

$$G : 1. \rightarrow,\ 2. \leftarrow^2,\ 3.C2,\ 4. \leftarrow^2,\ 5.C2,\ 6. \rightarrow.$$

Subrotina K: Quando scaneando um *square* marcado, encontra o *square* branco mais próximo à esquerda e termina no *square* imediatamente anterior.

$$K : 1.C3\ 2.C5,\ 3. \leftarrow,\ 4.C1,\ 5. \leftarrow.$$

Subrotina $M(a)$: Quando scaneando um P-*square*, se ele é branco, a máquina vai para a instrução a, se ele é marcado, então scanea o P-*square* imediatamente anterior: se ele é branco, então adiciona 1 ao último número na fita e vai para a instrução a; se ele é marcado, encontra e marca o *square* branco mais próximo à sua direita, adiciona 2 ao último número e para.

$$M(a) : 1.C3,\ 2.Ca,\ 3. \leftarrow^2,\ 4.C6,\ 5.A,\ Ca,\ 6. \rightarrow,\ 7.C6,\ 8.*,\ A^2.$$

Agora a subrotina I_m pode ser definida formalmente.

$$I_m : 1.*,\ 2.H,\ \leftarrow,\ D^m,\ G,\ M(4),\ 3.C2,\ 4.Y$$

Com I_m, a prova que todas as funções definidas pelos *schemas* (I)-(III) são computáveis está completa.

Para provar os outros 3 *schemas* (IV - VI) são necessárias mais algumas subrotinas. Elas serão definidas a seguir.

Subrotina J_m: Faz uma segunda cópia do m-ésimo número (da direita para a esquerda) para a esquerda do segundo (da direita para a esquerda) *big gap* que precede o *square* scaneado.

$$J_m : 1.*,\ H^2,\ \leftarrow,\ D^m,\ \leftarrow^2,\ 2.M(4),\ H^2,\ \leftarrow,\ D^m,\ K,\ 3.C2, 4.Y$$

Para fazer cópia de uma secessão de representações numéricas:

$$\overline{I}_m : I_m, \leftarrow^2, I_{m-1}, \leftarrow^2, ..., \leftarrow^2, I_1$$

$$\overline{J}_m : J_m, \leftarrow^2, J_{m-1}, \leftarrow^2, ..., \leftarrow^2, J_1$$

Para copiar $\overline{J}_m$ omitindo $\leftarrow^2, 1$, a seguinte notação é usada:

$$\overline{J}_m - J_1$$

Subrotina L_m: copia $y-1$ em vez de y, o m-ésimo numero à esquerda do *big gap* mais próximo que precede o *square* scaneado.

$$L_m : 1.*,\ 2.H,\ D^m,\ \leftarrow^2,\ G,\ M(4),\ 3.C2,\ 4.Y.$$

Difere de I_m na inserção de $\leftarrow^2$ no passo 2. E é aplicvel apenas para $y > 1$

Schema (IV) Se $\chi_1, ..., \chi_m$ e Ψ são recursivas (resp. B-computáveis), então a função φ definida no *schema* abaixo também é recursiva (resp. B-computável).

$$\varphi(x_1, ..., x_n) = \Psi[\chi_1(x_1, ..., x_n), ..., \chi_m(x_1, ..., x_m)]$$

Assumindo que as funções Ψ, χ_1, ..., χ_n têm os respectivos programas $\mathcal{P}(\Psi)$, $\mathcal{P}(\chi_1)$, ..., $\mathcal{P}(\chi_m)$. O programa $\mathcal{P}(\varphi)$, intuitivamente é dado por:

Copiar os argumentos $x_1, ..., x_n$ 2 vezes, manter a primeira cópia *clean* e encontrar o valor de $\chi_m(x_1, ..., x_n)$ com a segunda cópia e o programa $\mathcal{P}(\chi_m)$, então fazer 2 cópias da cópia *clean* de $x_1, ..., x_n$ e encontrar o valor de $\chi_{m-1}(x_1, ..., x_n)$ com uma cópia e o programa $\mathcal{P}(\chi_{m-1})$. Esses passos são feitos até que se chegue $\chi_1(x_1, ..., x_n)$. Após estar com todos os parâmetros $\chi_1(x_1, ..., x_n), ..., \chi_m(x_1, ..., x_m)$ aplica-se o programa $\mathcal{P}(\Psi)$ e teremos o valor de $\varphi(x_1, ..., x_n)$.

Para o caso de $n = 2$ o programa $\mathcal{P}(\Psi)$ é dado por:

$$Y, \overline{I}_n, \overline{J}_n, H^2, *, Y, \mathcal{P}(\chi_2), \leftarrow^3, *, \rightarrow^3, \overline{I}_n, \overline{J}_n, H^2, *, Y, \mathcal{P}(\chi_1), I_1, Z, \leftarrow^2, I_{n+1}, H, *, Y, \mathcal{P}(\Psi)$$

Schema (V) $\begin{cases} \varphi(1,\ x_2, ...,\ x_n) &= \Psi(x_2,\ ...,\ x_n), \\ \varphi(z+1,\ x_2,\ ...,\ x_n) &= \chi[z,\ \varphi(z,\ x_2,\ ...,\ x_n),\ x2,\ ...,\ x_n] \end{cases}$

Intuitivamente, para encontrar o valor de $\varphi(y,\ x_2, ...,\ x_n)$, para um dado y, precisamos calcular o valor de $\mathcal{P}(\chi)$ $y-1$ vezes. Isso é realizado efetuando-se testes sucessivos durante a execução se $y = 1$, $y-1 = 1$ ou $y-2 = 1$ ou etc. Desse modo, $\Psi(x_2,\ ...,\ x_n)$ é avaliado por $\mathcal{P}(\Psi)$ e testado se $y = 1$. Se $y = 1$, $\Psi(x_2,\ ...,\ x_n)$ é dado como resposta por $\mathcal{P}(\varphi)$; se $y \neq 1$, $\chi[1,\ \Psi(x_2,\ ...,\ x_n),\ x2,\ ...,\ x_n]$ é calculado e testado se $y-1 = 1$. Se $y-1 \neq 1$, $\chi\{2,\ \chi[1,\ \Psi(x_2,\ ...,\ x_n),\ x2,\ ...,\ x_n],\ x_2,\ ...,\ x_n\}$ é calculado e testado se $y-2 = 1$. E assim por diante. O programa $\mathcal{P}(\varphi)$ é dado por:

1.$Y,\ \overline{I}_n,\ \leftarrow^2,\ *,\ \rightarrow^6,\ \overline{J}_{n-1},\ H^2,\ *,\ Y,\ \mathcal{P}(\Psi),\ \leftarrow^2,\ I_{n+1},\ \leftarrow^6,$
2. $\leftarrow^2,\ C4,$
3. $\rightarrow^2,\ C5,$
4. $\rightarrow^6,\ \overline{I}_n,\ A,\ Y,\ L_1,\ I_{n+2},\ \overline{J}_n - J_1,\ H^2,\ *,\ Y,\ \mathcal{P}(\chi),\ \leftarrow^2,\ L_{n+1},\ \leftarrow^6,\ C2,$
5. $\rightarrow^4,\ I_2,\ Z.$

Schema (VI) $\varphi(x_1, ..., x_n) = \mu_y[\chi(x_1, ..., x_n, y) = 1]$

A construção de $\mathcal{P}(\varphi)$ a partir de $\mathcal{P}(\chi)$ é similar ao caso do *schema (V)*. Para quaisquer $x_1, ..., x_n$, verifica-se se $\chi(x_1, ..., x_n, 1)$ é igual a 1, se for igual a 1, para a execução; se não for 1, $\chi(x_1, ..., x_n, 2)$ é avaliado, e assim por diante. O programa $\mathcal{P}(\varphi)$ é dado por:

1.$Y,\ \overline{I}_n,\ \leftarrow^2,\ *,\ \rightarrow^6,\ \overline{J}_n,\ \leftarrow^2,\ *,$
2.$H^2,\ *,\ \mathcal{P}(\chi),\ \leftarrow^3,\ *,\ \leftarrow^8,\ C4,$
3. $\rightarrow^2,\ C5,$
4. $\rightarrow^6,\ \overline{I}_{n+1},\ A,\ \rightarrow^6,\ \overline{J}_{n+1},\ A,\ C2,$
5. $\rightarrow^4,\ I_1,\ Z.$

Isto completa a prova do teorema todas as funções recursivas são B-computáveis.

Utilizando a prova anterior e o que foi mostrado em [3] e [1], que toda função recursiva é Turing Computável e que toda função Turing Computável é recursiva. Fica demonstrada a equivalência entre Turing-Computabilidade e B-Computabilidade.

Referências

[1] *Teoria da Computação - Máquinas Universais e Computabilidade.* Paulo Blauth Menezes, Tiarajú Asmuz Diverio.

[2] *Introduction to Metamathematics.* Stephen Cole Kleene, 1952.

[3] *Languages and Machines - An Introduction to the Theory of Computer Science.* Thomas A Sudkamp, 1997.

[4] Hao Wang. A variant to turing's theory of computing machines. *Journal of the ACM (JACM)*, 1957.

Este trabalho foi feito tomando como base o artigo [4], disponível em:
http://portal.acm.org/citation.cfm?id=320856.320867