Turing-Pooh Machine (TPM)

Manual Preliminar do Usuário

Tutorial: Criando e Simulando uma Máquina de Turing

Máquina de Turing de Exemplo

Para ilustrar a correta maneira de codificar e simular uma máquina na TPM, iremos estudar a realização desse processo para uma máquina de Turing relativamente simples. Trata-se da máquina apresentada no slide 8 / aula 8 do curso de CTC-34/2010. Ela é uma máquina que reconhece a linguagem $L = \{0^n 1^n | n \geq 1\}$. Esta máquina está explicada em detalhes no material do curso. Aqui, apresentamos apenas a definição formal da máquina:

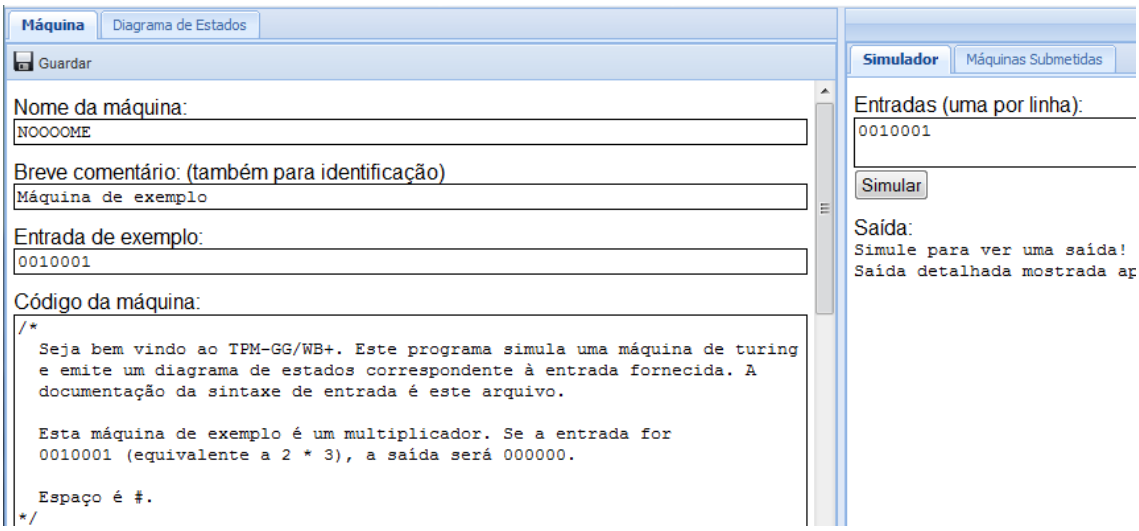
$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$
$$\Sigma = \{0,1\}$$
$$\mathrm{T} = \{0,1,X,Y,B\}$$
$$F = \{q_4\}$$

Transições:

Estado	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	(q_1, 0, R)	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	(q_2, 0, R)	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Passo 1: Codificando a Máquina no Formato Pooh

Até o presente momento a TPM não tem uma funcionalidade especifica para a criação de uma nova máquina. Assim, para iniciar a codificação da máquina acima, use como ponto de partida a máquina de exemplo que é apresentada quando você acessa a TPM ou qualquer outra máquina previamente submetida lá disponível.



Em primeiro lugar, altere o nome da sua máquina, dando a ela um nome que não coincida com uma máquina já existente, para evitar confusões. As máquinas já existentes podem ser acessadas na aba "Máquinas Submetidas" (painel da direita).

O campo breve comentário serve para que você identifique uma versão particular da sua máquina. Nesse primeiro momento, não se preocupe com ele, nem com o campo abaixo (entrada de exemplo).

O segundo passo é transcrever a definição formal da máquina para o formato Pooh. Observamos que o conjunto Q de estados tem 5 elementos, e o conjunto F de estado finais tem 1 elemento. Assim, o cabeçalho do código da máquina vai começar com esses dois números:

```
5 1 /* 5 estados, 1 deles final */
```

O próximo passo é especificar quais são os estados e quais são os estados finais, através de duas listas. Os nomes dos estados podem conter os caracteres de a até z, números e underscore. O código fica então

```
5 1 /* 5 estados, 1 deles final */

q0 q1 q2 q3 q4 /* Estados */
q4 /* Estados Finais */
```

Depois disso, é hora de escrever as transições de estados.

Importante: não é necessário especificar o conjunto de caracteres escrevíveis na fita (Σ e T) nem o símbolo em branco (B). Esses conjuntos, na TPM, são fixados e iguais a:

$$\mathrm{B} = \#$$
$$\Sigma = todos\ os\ caracteres\ ASCII - \{\#,/\}$$
$$\mathrm{T} = \Sigma + \{\#\}$$

A sintaxe para a escrita das transições esta exemplificada abaixo:

```
q0
{
    0:X,q1,R; /* Estando em q0, lendo 0, escrevo X, vou para o estado q1 e movo a
                 cabeça de leitura para a direita */
    Y:Y,q3,R;
}
```

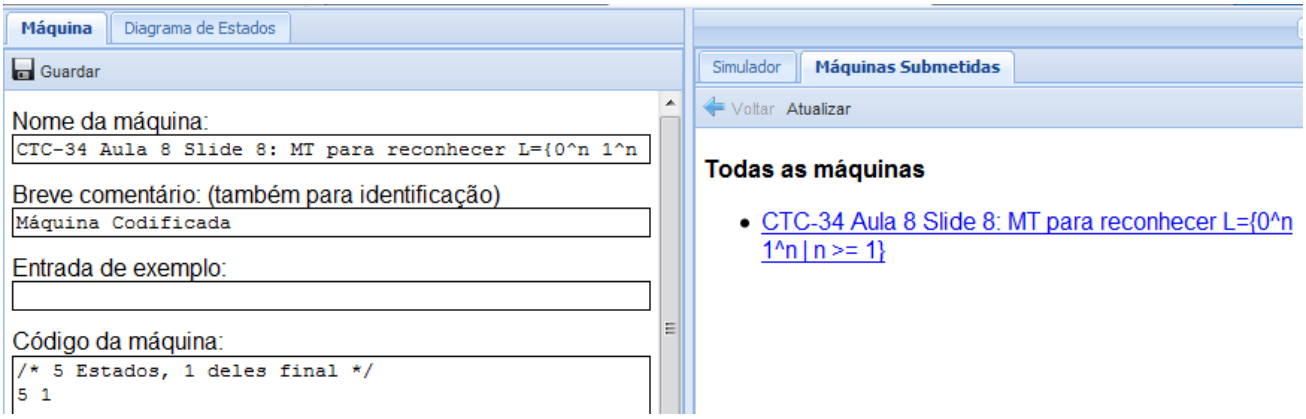
Desta forma, a transcrição completa da máquina apresentada acima em código Pooh segue abaixo:

```
/* 5 Estados, 1 deles final */
5 1

/* Estados */
q0 q1 q2 q3 q4

/* Estados Finais */
q4

q0
{
    0:X,q1,R;
    Y:Y,q3,R;
}

q1
{
    0:0,q1,R;
    1:Y,q2,L;
```

```
    Y:Y,q1,R;
}

q2
{
    0:0,q2,L;
    X:X,q0,R;
    Y:Y,q2,L;
}

q3
{
    Y:Y,q3,R;
    #:#,q4,R;
}

q4
{
}
```

Importante: como mostrado logo acima, caso um estado não tenha nenhuma transição definida, é necessário, mesmo assim, listar que esse estado apresenta um conjunto vazio de transições (caso do estado q_4 mostrado logo acima).

Feito isso, em "Breve Comentário" digite algo como "Primeira Versão" ou "Máquina Codificada" salve a máquina usando o botão "Guardar". Navegando para a aba "Máquinas Submetidas", sua máquina deverá aparecer na lista de máquinas submetidas.



Clicando no nome de sua máquina, todas as versões já submetidas estarão lá disponíveis, em ordem cronológica inversa. O "Breve comentário" é fundamental para se saber o que mudou de uma versão para a outra...

Passo 2: Simulando a Máquina para Verificar seu Funcionamento

Vamos agora verificar se a máquina funciona. No lado direito da tela, vá para a aba “Simulador” se esta já não estiver ativa. Escolha uma cadeia (preferencialmente pequena) para testar o funcionamento da máquina, digite essa cadeia em “Entradas (uma por linha)” e pressione Simular. Aqui, duas coisas podem acontecer:

1) Caso o código digitado apresente algum erro de sintaxe, a saída da TPM será uma mensagem informado qual erro ocorreu
2) A máquina será executada para o input fornecido

Para a nossa máquina, escolhendo a cadeia 0011 (que deve ser aceita), obtemos a seguinte saída:

```
Input accepted!

Final Configuration:
13) XXYY#(#)
          q4

Step Count: 14
Run time: 0.000 seconds

Computation steps:
0) (0)011
   q0

1) X(0)11
     q1

2) X0(1)1
      q1

3) X(0)Y1
     q2

4) (X)0Y1
    q2

5) X(0)Y1
     q0

6) XX(Y)1
      q1

7) XXY(1)
       q1

8) XX(Y)Y
      q2

9) X(X)YY
     q2

10) XX(Y)Y
       q0

11) XXY(Y)
        q3

12) XXYY(#)
         q3

13) XXYY#(#)
          q4
```

O formato da saída é (espera-se) bastante simples de entender. A primeira linha diz qual foi o resultado da simulação. Esse resultado pode ser:

- Input accepted! → máquina parou em estado final
- Input rejected! → máquina parou em estado não final
- Machine entered in loop (steps X and Y have identical configurations) → uma mesma configuração instantânea se repetiu duas vezes durante a simulação, indicando a presença de um ciclo
- Computation aborted. Step limit exceeded → a máquina nem parou nem entrou em loop após 50000 passos

Após isso, qualquer que tenha sido o caso, é mostrada a última configuração instantânea obtida, seguida de estatísticas sobre a simulação (número de passos e tempo de execução). Depois disso, são mostradas todas as configurações instantâneas obtidas ao longo da computação.

```
9)  X(X)YY
      q2
```

A configuração instantânea começa indicando o número do passo (no caso acima, o 9º passo da simulação). Depois, é mostrada a fita. O caracter entre parêntesis é aquele sobre o qual a cabeça de leitura se encontra. Embaixo desse caracter, é mostrado em qual estado a máquina se encontra. Na representação $\alpha_1 q \alpha_2$, a configuração acima equivale a Xq_2XYY

Caso as computações não procedam como o esperado, analise como a máquina operou passo-a-passo a fim de encontrar o problema. Caso tenham sido feitas quaisquer correções, digite em “Breve Comentário” o que foi alterado e pressione “Guardar” novamente.

> **Recomendação:** antes de salvar novamente a máquina, digte em “entrada de exemplo” ma das cadeias que sua máquina aceita. Desta forma, caso outras pessoas queiram olhar como sua máquina funciona, elas já tem um ponto de partida para saber em que formato a entrada da máquina deve ser fornecida.

Passo 3: Teste Exaustivo

Para verificar o funcionamento da máquina, provavelmente você realizou várias simulações usando cadeias de entrada diferentes. Assim que constatar que, em geral, a máquina parece estar funcionando bem, não faz mais sentido ficar testando cadeia a cadeia e obter todos os passos da simulação. Para isso, a TPM aceita simular mais de uma entrada de cada vez. Quando isso ocorre, a saída mostrada não é mais uma saída detalhada, como acima, mas apenas um sumário, que mostra a última configuração instantânea obtida, número de passos, e resultado da simulação (aceitação/rejeição/loop/número máximo de passos excedido) numa forma de tabela. Para a nossa máquina de exemplo, podemos então gerar um conjunto de entradas que testa exaustivamente a máquina, como o abaixo:

```
#
0
1
01
001
011
10
00000000001111111111
```

A idéia aqui é escrever todos os diferentes "tipos de entrada". Acima, temos:

```
# → Entrada vazia → deve ser rejeitada
0 → Só um zero → deve ser rejeitada
1→ Só um um → deve ser rejeitada
01 → cadeia aceita
001 → mais zeros do que uns → deve ser rejeitada
011 → mais uns do que zeros → deve ser rejeitada
10 → mesmo número de zeros e uns, mas na ordem errada → deve ser rejeitada
00000000001111111111 → cadeia grande → deve ser aceita
```

Simulando todos aqueles valores, obtemos a seguinte saída:

```
Input                Result     Steps  State: Tape

<empty>              rejected       1 q0: (#)
0                    rejected       2 q1: X(#)
1                    rejected       1 q0: (1)
01                   accepted       6 q4: XY#(#)
001                  rejected       8 q1: XXY(#)
011                  rejected       5 q3: XY(1)
10                   rejected       1 q0: (1)0
00000000001111111111 accepted     222 q4: XXXXXXXXXXYYYYYYYYYY#(#)
```

Essa saída resumida fornece um "raio X" do funcionamento da máquina. Ele indica, rapidamente, para quais cadeias a máquina está funcionando ou não.

> **Recomendação:** Guarde essa lista de cadeias de entrada num arquivo texto. Assim, toda vez que sua máquina for modificada por você ou outra pessoa, executando tais entradas pode-se verificar rapidamente se a máquina continua funcionando corretamente após as modificações feitas.

Passo 4: Análise de Complexidade (Avançado)

Máquinas de Turing são importantes do ponto de vista teórico, mas, na prática, apresentam performance muito ruim (executam lentamente os algoritmos). Por isso, não faz muito sentido (a princípio) tentar analizar a complexidade com qual a máquina executa um algoritmo (complexidade aqui significa, para entradas grandes, como o tempo de execução varia com o tamanho de entrada. Se um algoritmo tem complexidade n^2, por exemplo, isso quer dizer que a máquina leva 10 segundos para processar uma entrada de tamanho 100, para uma entrada de tamanho 1000 (10 vezes maior) ela levará 1000 segundos (100 vezes mais) para terminar a computação).

Entretanto, seja para justificar a afirmação acima (MT's são úteis apenas do ponto de vista teórico) num relatório de CTC-34, ou por curiosidade, você pode querer analisar como o número de passos varia com o tamanho da entrada. Para isso, pode-se usar uma entrada formada por várias cadeias iguais, a menos do seu tamanho, e observar, na saída resumida, o número de passos necessários na computação:

```
00000000001111111111                     accepted      222 ...
000000000011111111111                    accepted      266 ...
0000000000011111111111                   accepted      314 ...
00000000000011111111111                  accepted      366 ...
000000000000011111111111                 accepted      422 ...
0000000000000011111111111                accepted      482 ...
00000000000000011111111111               accepted      546 ...
000000000000000011111111111              accepted      614 ...
0000000000000000011111111111             accepted      686 ...
00000000000000000011111111111            accepted      762 ...
000000000000000000011111111111           accepted      842 ...
```

Com os dados acima, pode-se observar que o número de passos varia assintóticamente com o quadrado do tamanho da cadeia (a última cadeia tem tamanho n = 20, a primeira, n = 10, e o número de passos para n = 20 é cerca de 4 vezes maior do que para n = 10). Observe que essa é uma análise mais qualitativa do que formal, ela apenas permite a você ter uma idéia da complexidade do algoritmo implementado sem sair fazendo um monte de conta em cima da forma de operar da máquina.

Um algoritmo simples em C consegue aceitar essa linguagem em tempo proporcional ao tamanho da cadeia, isto é, conforme *n* aumenta bastante, o algoritmo em C, com tempo de execução proporcional a *n*, fica muito mais eficiente para reconhecer essa linguagem do a máquina de Turing (que tem tempo proporcional a n^2), comprovando assim que MT's não "servem pra nada" na prática :P.

Gerador de Diagramas de Estaado

Seja para auxiliar no relatório, na depuração de porque a máquina não funciona, ou por diversão, a TPM oferece um módulo gerador de diagramas de estado que lê o código Pooh e produz um grafo com todas as transições da máquina. Como esse diagrama é gerado por rotinas automatizadas, ele nem sempre ficará muito legível. Para isso, algumas configurações podem ser feitas. A idéia é sair experimentando até obter o diagrama mais legível (ou menos ilegível) possível. Para o relatório, pode ser necessário dar uns retoques finais no paint ;P

- Grafo Horizontal: o grafo sempre terá cara de árvore. Aqui, você define se quer uma árvore na vertical ou na horizontal
- Razão Altura / Largura: permite "esticar" o grafo na horizontal ou na vertical, conforme valor escolhido. Pode ser útil para evitar que rótulos fiquem muito próximos, atrapalhando a legibilidade.

Subrotinas

Para verificar como esse recurso funciona, abra o "Escovador 4" já cadastrado na WTPM (qualquer versão lá disponível serve). Ao abrir o Diagrama de Estados, você verá que tem umas molduras envolvendo partes do diagrama: "ajusta_sinal", "copy" e "shift". Essas molduras indicam as subrotinas definidas para aquela máquina

Clicando em "Subrotinas em grafos separados", a TPM trata cada um dos blocos "ajusta_sinal", "copy" e "shift" como uma submáquina, deixando elas em grafos separados, e no grafo principal ficam apenas os estados necessários. Na subrotina "copy", pode-se perceber que apenas os estados "marca_digito" e "desmarca_digitos" apresentam transições vindo ou saindo pra fora dela. Ou seja, no grafo principal, apenas esses estados serão mostrados.

Como definir subrotinas?

Vamos olhar como foi escrito o cabeçalho do código Pooh Escovador 4:

```
25 1 3 ← o "3" aqui indica que além de 25 estados, dos quais 1 final, temos 3 subrotinas

/* Estados */
(...)

/* Estados Finais */
(...)

/* Subrotinas */
ajusta_sinal 6 positivo negativo tira0p tira0n fimpos fimneg
copy 6 marca_digito copia_digito percorre_produto volta_p_2o_num
       volta_para_prox_digito_a_copiar desmarca_digitos
shift 7 shift_init end_carac go_left start_shift l1 l2 l3
```

Teremos então uma lista com 3 subrotinas. Cada item dessa lista deve ser escrito na forma:

<nome da subrotina> <número de estados que fazem parte dela> <lista com o nome de cada um dos estados>

Ou seja, no exemplo acima, a primeira rotina definida chama "ajusta_sinal", sendo composta por 6 estados: positivo, negativo, tira0p, tira0n, fimpos, fimneg.

Importante: Para a simulação, definir ou não subrotinas não faz a menor diferença.