MUST: Mais Um Simulador da Máquina de Turing

Claudio Cesar de Sá[1] **, Rajshekhar Sunderraman**[2]

[1]Departamento de Ciência da Computação (DCC)
Universidade do Estado de Santa Catarina (UDESC)
Campus Universitário Prof. Avelino Marcante, S/N
Bloco F - 3o. Piso - Sala F. Gauss
89223-100 - Joinville, SC

[2]Department of Computer Science
Georgia State University
P.O. Box 4110
Atlanta, GA 30302-4110

claudio@joinville.udesc.br, raj@cs.gsu.edu

Abstract. *This paper discusses and presents another Turing Machine Simulator, written in Prolog, open source, aimed to compute functions and languages accepted by TM. Its original version was developed by R. Sunderraman, and 90% of this coded was rewritten for particular proposals in an undergraduate course of Computer Science Bachaleror.*

Resumo. *Este artigo apresenta um simulador para Máquina de Turing (MT) clássica, sob a visão de contínuos aperfeiçoamentos por se tratar de um código aberto, escrita em Prolog, destinada a computar funções e reconhecer linguagens aceitas por uma MT. A sua versão original foi escrita por R. Sunderraman, e aproximadamente 90% (noventa) do código foi alterado. A importância da MT deve-se ao fato de derivar os termos clássicos da área da Teoria da Computação.*

1. Introdução

1.1. Motivação

A Máquina de Turing (MT) [Brookshear, 1989, Sipser, 1996, Divério, 1999] é um dos conceitos fundamentais da Ciência da Computação pois evidência e demonstra vários conceitos da área, como por exemplo: *algoritmo*, *computação*, *função computável*, etc. Adicionalmente, enfatiza questões clássicas como o *problema da parada* de programas computacionais.

Algumas dezenas de simuladores de MT encontram-se disponíveis para download, e destinados ao uso no ensino de graduação na área de Teoria da Computação. Contudo, a maioria destes encontra-se apenas o código executável e mono-plataforma.

A proposta deste artigo é apresentar o MUST: **M**ais **U**m **S**imulador da Máquina de **T**uring, utilizando um exemplo, e relatar a experiência, de um simples e robusto simulador escrito em Prolog (código aberto) e multi-plataforma. Assim, este artigo pode ser utilizado como um "*manual*" ao **MUST**.

1.2. Reflexões sobre a Máquina de Turing

Enfatiza-se o conceito de MT para disciplina de Teoria da Computação sob os seguintes pontos:

- ✓ Um modelo genérico e formal de computação (e de computador);
- ✓ Uma definição precisa sobre o conceito de algoritmo;
- ✓ Suporta a Tese de Church-Turing, onde todo procedimento é computável por uma MT;
- ✓ A tese se reforça ao longo dos anos em função dos problemas resolvidos via MT, tais como:
  - $a^n b^n$ para $n \geq 0$ (glc)
  - $a^n b^n c^n$ para $n \geq 0$ (gsc)
  - $f(x, y) = x + y$
  - $g(x, y) = x * y$
  - $fat(n) = n!$
  - ...
- ✓ Equivalência com outros modelos (funções parciais, λ-cálculo, Post, Markov, Bird), onde a computabilidade se equivale as MT;
- ✓ Evidencia o clássico resultado da CC que é o "*Problema da Parada*";

1.3. A Experiência

Este simulador tem sido aplicado na disciplina de Teoria da Computação do curso de Bachalerado em Ciência da Computação da UDESC. Neste curso, o aluno usa este simulador para construir exemplos clássicos da área como: $a^n b^n c^n$ para $n >= 0$, a soma (código unário), a multiplicação (código unário), e finalmente, cálculo do fatorial. Alguns destes exemplos bem como o código estão disponíveis em http://www.itajuba.brturbo.com/claudio/cursos/teoria_da_computacao.

As vantagens deste simulador são:

- ✓ A forte relação entre os conceitos teóricos de Máquina de Turing e uma prática com este simulador;
- ✓ Escrito em SWI-Prolog (http://www.swi-prolog.org/), possibilitando ser um código multi-plataforma (Windows, Linux, etc);
- ✓ A possibilidade de se alterar seu código fonte original para extensões e finalidades diversas.

Este simulador encontra-se em fase de testes desde agosto/2003, onde todos alunos da disciplina acima mencionada, desenvolvem seus exercícios com esta ferramenta. Desde então, nenhum problema tem sido detectado.

1.4. Características da Implementação

- ✓ Originalmente foi escrito por Rajshekhar Sunderraman da Georgia State University, http://tinman.cs.gsu.edu/∼raj;

✓ 90% do código foi alterado objetivando torná-lo mais próximo do funcionamento original da definição de MT. Por exemplo, a introdução de marcadores, símbolos de controle de fita, a esquerda (@) e a direita (#) de uma entrada na fita;
✓ Linguagem de Programação: SWI-Prolog (multi-plataforma, etc, http://www.swi-prolog.org/);
✓ Aproximadamente 100 predicados (1 por linha de código);
✓ Todo material disponível em:
http://www.itajuba.brturbo.com/claudio/cursos/teoria_da_computacao

2. Desenvolvendo um Exemplo

O exemplo a ser desenvolvido neste artigo é o da soma unária onde $\sum = \{1\}$. Exemplificando: $4_{10} + 2_{10} \equiv 11_1 + 1111_1 \equiv 111111_1 \equiv 6_{10}$

2.1. Descrição a Nível de Implementação

Seguindo uma proposta de Sipser [Sipser, 1996], numa descrição dos movimentos do cabeçote sobre fita, para o caso da soma unária tem-se o seguinte procedimento:

1. Em q_0, se ler um $+$, escreva @ na fita e mude ao estado de aceitação (h);
2. Caso leia um 1 (ainda em q_0), escreva x na fita, mude para o estado q_1 e vá à direita na fita;
3. Em q_1, enquanto encontrar 1 ou $+$ na fita, vá à direita mantendo o que encontrar na fita;
4. Se ler #, escreva 1 na fita, mude ao estado q_2 e vá à direita na fita;
5. Se ler #, escreva # na fita, mude ao estado q_3 e vá à esquerda na fita;
6. Enquanto encontrar 1 ou $+$ na fita (ainda q_3), vá à esquerda mantendo o que encontrar na fita;
7. Se ler x, mantenha-o, vá à direita, mude ao estado inicial q_0 e então, volte ao primeiro passo.

A soma é realizada neste procedimento por meio do transporte dos 1's que estão do lado esquerdo do $+$ para o lado direito do mesmo. Antes de cada 1 ser transportado, ele deve ser marcado para não ser, no futuro, considerado. No fim, basta sobrescrever o símbolo $+$ pelo marcador de início de string (no caso, @). Este algoritmo funciona para expressões do tipo $1111 + 11$ $(4+2)$, $+111$ $(0+3)$ e $11+$ $(2+0)$. O programa escrito com base neste algoritmo é 'soma-unaria.txt' e funciona no simulador de MT feito em Prolog.

Para essa função soma (unária): $f(x, y) = x + y$, a uma definição formal desses movimentos conforme o procedimento descrito acima é dado por:

1. δ(0,+) = (h,@,r)
2. δ(0,1) = (1,x,r)
3. δ(1,1) = (1,1,r)
4. δ(1,+) = (1,+,r)
5. δ(1,#) = (2,1,r)
6. δ(2,#) = (3,#,l)
7. δ(3,1) = (3,1,l)

8. $\delta(3,+) = (3,+,l)$
9. $\delta(3,x) = (0,x,r)$

A codificação no simulador é feita de maneira direta em um arquivo texto, com a sintaxe do Prolog. Para o caso acima, o arquivo a ser digitado é dado por:

```
start_state(0).  %% estado inicial
pos_inic_cabecote(2).  %% deslocamento inicial do cabeçote
%% delta(est_i, est_novo, simb_lido,simb_escrito, movimentacao do cabecote)

delta(0,h,+,@,r).
delta(0,1,1,x,r).
delta(1,1,1,1,r).
delta(1,1,+,+,r).
delta(1,2,#,1,r).
delta(2,3,#,#,l).
delta(3,3,1,1,l).
delta(3,3,+,+,l).
delta(3,0,x,x,r).
```

2.2. Diagrama de Estado de Transição

Aqui reside uma desvantagem deste simulador. O mesmo não apresenta uma visualização gráfica da MT desenvolvida pelo aluno. Contudo, tal tarefa não seria difícil a partir do arquivo que tem a codificação da MT, ver código acima. Para o exemplo anterior, o mesmo é apresentado na figura 1. Esta tarefa, em geral é atribuída aos alunos.

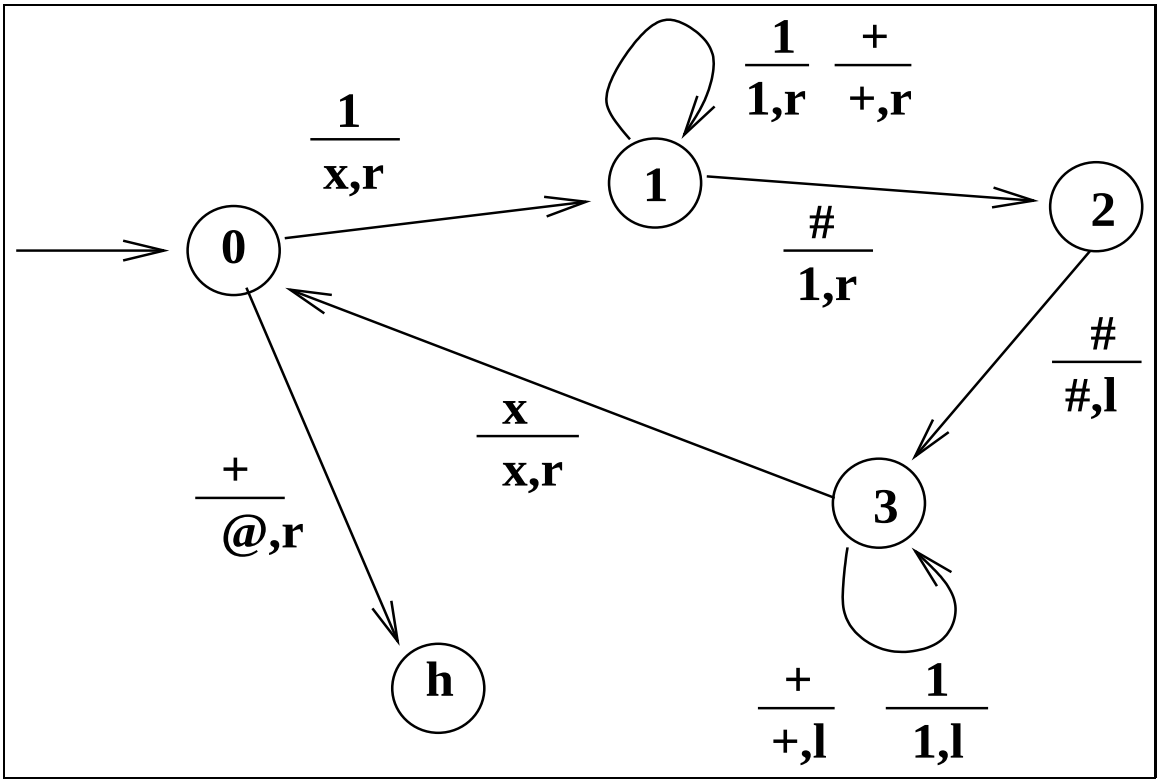

Figura 1: Diagrama de Estado de Transição Equivalente

2.3. Definição Formal

Finalmente, a definição formal desta MT é dada pela séptula seguinte:

$MT = (\{q_0, q_1, q_2, q_3, h, q_{rej}\}, \{1, +x\}, \{@, \#\}, \delta, q_0, h, q_{rej})$, onde δ é assim definida:

δ	@	#	1	+	x
q_0	-	-	(q_1, x, r)	(h, @, r)	-
q_1	-	(q_2,1,r)	(q_1, 1, r)	(q_1, 1, r)	-
q_2	-	(q_3, #, l)	-	-	-
q_3	-	-	(q_3, 1, l)	(q_3, +, l)	(q_0, x, r)
h	-	-	-	-	-

Tabela 1: Tabela Função de Transição entre Estados

Onde:

- h: um estado de parada (aceitação);
- @ e # são marcadores *default* a esquerda e a direita de uma entrada qualquer. Uma convenção para facilitar a identificação dos lados esquerdo e direito da fita;
- q_{rej}: estado de rejeição, que por *default* são todas transições não especificadas.

2.4. Uma Execução

Considerando o exemplo da soma-unária, a execução de $4_{10} + 2_{10}$ neste simulador seria feito diretamente ao carregar o simulador e a função a ser computada. Esta codificada no arquivo da funça δ, neste exemplo é dado por 'soma-unaria.txt' (ver código da seção 2.1). Num ambiente Prolog qualquer, o predicado utilizado é o *consult*. Exemplificando esta execução tem-se:

```
% d:/claudio/teoria_comp/simulador/soma-unaria.txt compiled 0.00 sec, 336 bytes
Digite a entrada + <Enter>:: 1111+11

Fita: @1111+11#
       ^                          estado corrente é: 0
Fita: @x111+11#
        ^                         estado corrente é: 1
Fita: @x111+11#
         ^                        estado corrente é: 1
.....................................................
/* vai escrever 1 ao fim da entrada e retornar */
Fita: @x111+11#
             ^                    estado corrente é: 1
Fita: @x111+11#
              ^                   estado corrente é: 1
Fita: @x111+111#
               ^                  estado corrente é: 2
Fita: @x111+111#
              ^                   estado corrente é: 3
```

```
Fita: @x111+111#
              ^                    estado corrente é: 3
Fita: @x111+111#
             ^                     estado corrente é: 3
.....................................................
/* escrevendo um último 1 e retornando para mais 1's */
Fita: @xxxx+11111#
               ^                   estado corrente é: 1
Fita: @xxxx+11111#
                ^                  estado corrente é: 1
Fita: @xxxx+11111#
                 ^                 estado corrente é: 1
Fita: @xxxx+111111#
                  ^                estado corrente é: 2
Fita: @xxxx+111111#
                 ^                 estado corrente é: 3
Fita: @xxxx+111111#
                ^                  estado corrente é: 3
Fita: @xxxx+111111#
               ^                   estado corrente é: 3
Fita: @xxxx+111111#
              ^                    estado corrente é: 3
Fita: @xxxx+111111#
             ^                     estado corrente é: 3
Fita: @xxxx+111111#
            ^                      estado corrente é: 3
Fita: @xxxx+111111#
           ^                       estado corrente é: 3
Fita: @xxxx+111111#
          ^                        estado corrente é: 3
Fita: @xxxx+111111#
         ^                         estado corrente é: 3
Fita: @xxxx+111111#
           ^                       estado corrente é: 0
Fita: @xxxx@111111#
            ^                      estado corrente é: h

Conteúdo Final da Fita:..............
Fita: @xxxx@111111#
            ^                      estado corrente é: h
 Deslocamento do Cabecote: 7
 Parada com h: halt  ... parabéns !

Yes
?-
```

3. Conclusões

Algumas reflexões do exemplo apresentado que tangem o conceito de *computação*, citam-se:

✓ Técnica de marcadores ("*contagem*");

✓ Uso de movimentações (“*atribuição*”);
✓ Procedimento recursivo;
✓ Uso do r/w.

Os detalhes sobre estes conceitos podem ser encontrados em livros de Teoria da Computação, e para o enfoque aqui apresentado, mais especificamente em [Sipser, 1996].

Os alunos tem utilizado o MUST, e complementam o seu conhecimento com o simulador gráfico do prof. Tiaraju [Divério, 1999] http://www.inf.ufrgs.br/∼hgmc/teorica/paginas/tecomp.html. O conhecimento sobre MT fica enriqucedor sob estas visões de simulador: um gráfico o outro carácter. Os alunos ficam confiantes em testarem suas idéias, e validarem seus códigos sob estes dois simuladores.

Referências

Brookshear, G. J. (1989). *Theory of Computation: Formal Languages, Automata, and Complexity*. The Benjamin/Cummings Publishing Company, United States.

Divério, T. A.; Menezes, P. F. B. (1999). *Teoria da Computação: Máquinas Universais e Computabilidade*. Sagra-Luzzatto, . Porto Alegre - RS. Série Livros Didáticos do Instituto de Informática da UFRGS, n.5.

Sipser, M. (1996). *Introduction to the Theory of Computation*. PWS Publishing Company, 2nd. edition edition. Errata url: http://www-math.mit.edu/šipser/itoc-errs1.2.html.