

Instruções para o trabalho

Este documento contém todas as instruções para o desenvolvimento do primeiro trabalho prático. Siga os requisitos apresentados, formatos de entrada e saída, funcionamento dos comandos e etc.

1. Resumo:

O trabalho consiste em desenvolver um simulador para uma variante da Máquina de Turing, doravante denominada MT*. Nossa máquina MT* é equipada com 3 fitas, denominadas X, Y, Z, mais a capacidade para programação de aliases e procedimentos. A Figura 1 ilustra a idéia:

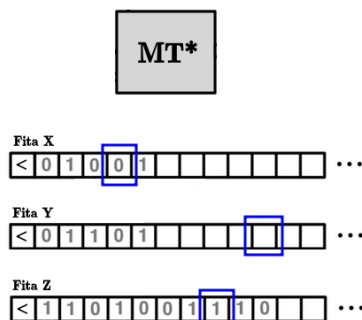


Figura 1: Esquema conceitual para a MT*

A sintaxe dos comandos da MT* foi inicialmente inspirada no formato adotado no simulador disponível em <http://morphett.info/turing/turing.html>, mas devido aos novos recursos adicionados ao projeto da máquina foram acrescentadas algumas alterações simples e poderosas para facilitar a programação da MT* pelos usuários. O simulador da MT* produto deste trabalho vai trabalhar de modo sequencial e determinístico, conforme será explicado nesse documento.

Este documento foi estruturado da seguinte forma:

1. Resumo – Apresenta essa descrição.
2. Linguagem da MT* – Descreve a sintaxe e semântica de todos os comandos aceitos na MT*.
3. Exemplo – Apresenta um breve exemplo de código programado para a MT*.
4. Entrada – Descreve o formato da linha de comando para alimentar a MT*.
5. Saída – Descreve o formato de saída para a MT*.

O trabalho pode ser executado em grupo de até dois alunos. A data de entrega será combinada em sala de aula, bem como as apresentações dos trabalhos.

2. Linguagem da MT*

Basicamente, existem 4 tipos de comandos: os comandos básicos para descrever as computações, os comandos para criação de aliases, os comandos para criação e chamada de blocos, mais alguns poucos comandos especiais.

Características particulares da implementação da MT*

- O simulador não implementa nenhum símbolo especial para denotar o início da fita. Em tese as fitas são infinitas em ambas as direções e se o usuário quiser utilizar alguma marca de início e/ou fim de fita deve acrescentá-las ele no código fonte que será executado na MT*.
- A entrada fornecida para a máquina será sempre alimentada na fita X, permanecendo inalteradas as fitas Y e Z. Depois a execução iniciará com o cabeçote posicionado no primeiro símbolo da cadeia fornecida como entrada para o programa. No caso de a entrada fornecida for nula/vazio, será indiferente onde o cabeçote inicia na fita X porque estará toda ela branca.
- Se em algum momento da simulação o tamanho utilizado em qualquer das fitas ultrapassar o limite possível (alguma restrição de memória definida pelo simulador), o simulador indicará o erro de estouro.
- A simulação termina quando a próxima computação for indefinida ou a MT* executa um dos seguintes comandos: **pare** , **aceite** ou **rejeite** .
- A simulação determina a próxima computação analisando sequencialmente as linhas de código do programa sendo simulado até encontrar a primeira computação viável, por isso a simulação será sempre determinista. Fique atento, porque essa característica torna o funcionamento do nosso simulador diferente do simulador online apresentado na sala de aula.
- Tudo que aparece depois de um ';' no código fonte é tratado como comentário e ignorado pelo simulador.

2.1. Comandos básicos para descrever as computações

- A sintaxe para os comandos desse tipo é:
<estadoA> <fitaA> <simbA> <moveA> - - <estadoB> <fitaB> <simbB> <moveB>
com a semântica baseada numa idéia simples: o lado esquerdo dos marcadores “- -” contém as informações que permitem selecionar qual comando será executado, enquanto o lado direito informa o que será alterado.

Segue o significado dos termos apresentados na sintaxe do comando:

- <estadoA>** – número do estado atual.
- <fitaA>** – identificador da fita de leitura, uma dentre X, Y ou Z.
- <simbA>** – símbolo esperado no cabeçote da <fitaA>.
- <moveA>** – tipo de movimento executado na <fitaA> depois de lido o cabeçote.
- <estadoB>** – número do novo estado caso seja executada a instrução.
- <fitaB>** – identificador da fita de escrita, uma dentre X, Y ou Z.
- <simbB>** – símbolo escrito no cabeçote de <fitaB> caso seja executada a instrução.
- <moveB>** – tipo de movimento executado na <fitaB> caso seja executada a instrução.

Note que é possível examinar uma fita para determinar o que escrever em outra, isso confere maior liberdade para o programador sendo sua responsabilidade especificar computações que façam sentido.

- A cronologia dos eventos que descrevem uma computação é a seguinte:
 - (1) o simulador lê o símbolo sob o cabeçote da <fitaA>, chamemos de <lido> esse caractere;
 - (2) o simulador procura casamento nos pares (<estadoA>, <simbA>) e (<estadoA>, <lido>);
 - (3) seguindo a ordem no código fonte, o primeiro casamento determina qual instrução executar.
- Para denotar <estadoA> ou <estadoB> você pode utilizar um inteiro de até 4 dígitos.
- Para denotar <simbA> ou <simbB> você pode usar qualquer caractere. Use '_' para representar o branco (espaço). O uso de aliases é permitido nesses campos. Aliases serão abordados mais a frente noutro tópico.
- O <moveA> e <moveB> indicam a direção do movimento do cabeçote em <fitaA> e <fitaB> respectivamente: 'e' denota movimento para a esquerda, 'd' denota movimento para a direita, 'i' denota ausência de movimento (imóvel).

Extensões dos comandos básicos:

- '*' pode ser usado como coringa em `<simbA>` para denotar qualquer caractere.
- '*' pode ser usado como coringa em `<simB>` e `<estadoB>` para significar ausência de mudança.
- '!' pode ser usado no final da linha para criar um *breakpoint*. Durante a execução do programa, a máquina vai pausar automaticamente depois de computar uma linha com *breakpoint* e reabrir o *prompt* para aguardar nova opção.

2.2. Criação e uso de aliases

- A sintaxe da instrução para criação de aliases é:
`<alias> = <string>`
com a simples semântica: o `<alias>` casa qualquer caractere contido em `<string>`.
- Nomes válidos para aliases seguem a forma regular '\$[a-z]', ou seja, letra minúscula prefixada com dólar.
- O uso de aliases é permitido na definição de `<simbA>` e `<simB>` na codificação de comandos básicos. O significado do alias no campo `<simbA>` é óbvio, ele casa os caracteres que representa. Diferentemente, no campo `<simbB>` ele denota o mesmo caractere que foi casado no campo `<simA>` nos casos em que esse alias foi utilizado em `<simbA>`. Usos estranhos como usar aliases diferentes em `<simbA>` e `<simbB>`, ou usar alias em `<simbB>` sem ter usado em `<simbA>`, vão produzir erro por mau uso de alias.

Um uso comum de alias, por exemplo, é o reconhecimento de dígitos:

```
$d = '0123456789'
```

2.3. Criação e chamada de blocos

- Para iniciar um novo bloco a sintaxe é:
`inicio <identificador de bloco> <estado inicial>`
- Para finalizar um bloco a sintaxe é:
`fim <identificador de bloco>`
- Para chamar um bloco a sintaxe é:
`<estado atual> <identificador de bloco> <estado de retorno>`
- Os estados dentro de um bloco são independentes e não conflitam com estados de outros blocos, isso define uma regra de escopo para os estados. A execução do bloco vai iniciar no estado inicial fornecido na declaração.
- Para sair e retornar de um bloco utilize a instrução:
`<estado atual> retorne`
que vai devolver a execução para o bloco chamador, no estado de retorno fornecido na chamada.
- A execução do programa sempre inicia no bloco especial de nome "main".

2.4. Comandos especiais

- Para interromper uma execução, use a instrução:
`<estado atual> pare`
- Para interromper uma execução informando aceitação da entrada, use a instrução:
`<estado atual> aceite`
- Para interromper uma execução informando rejeição da entrada, use a instrução:
`<estado atual> rejeite`

3. Exemplo de código

O breve e simples exemplo de código abaixo serve para ilustrar como é feita a programação da MT*.

```
1      ; copia inteiro na fita X, fornecido na entrada , para as fitas Y e Z
2
3      $d = "0123456789"
4
5      ; checa validade da entrada
6      inicio main 01
7          01 X $d i — 03 X $d i
8          01 X * i — 05 X * i
9          03 copiaX 04
10         04 aceita
11         05 rejeita
12     fim main
13
14     ; copia o inteiro em X para Y e Z
15     inicio copiaX 1
16         01 X $d i — 03 Y $d e
17         01 X * i — 02 X * i
18         02 retorne
19         03 X $d e — 01 Z $d e
20     fim copiaX
```

4. Entrada

Para facilitar a implementação do simulador, suas entradas e saídas serão sempre em modo texto. O simulador deve iniciar na linha de comando cuja sintaxe será:

simuladorMT <opções> <arquivo> <entrada>

com <arquivo> denotando o nome do arquivo (padrão *.mt) contendo o código do programa a ser simulado e <entrada> denotando uma string fornecida como entrada para a execução do programa. As <opções> podem ser:

- **-step <n>** Executa n computações e para, mostrando o conteúdo das três fitas, então reabre o *prompt* para ler novo valor para <n> antes de continuar a simulação. Caso seja fornecido o valor 0 para n a simulação termina imediatamente, caso seja fornecido um valor negativo o programa considera a opção -resume.
- **-resume** Executa o programa até o fim, mostra o conteúdo das três fitas e o resultado do reconhecimento: ACEITA ou REJEITA. Essa é a opção padrão e será considerada se nenhuma opção -step for fornecida.
- **-debug <arquivo log>** A simulação produz um relatório mostrando por linha as instruções executadas, as entradas e saídas dos blocos.
- **-help** Exibe mensagem explicando o formato da linha de comando.

Para prevenir contra loops infinitos (no caso das opções -resume), o simulador deve interromper a simulação depois da execução de 1000 computações, e reabrir o *prompt* para a leitura de nova opção.

Exemplo:

```
1      prompt> simuladorMT —step 1 clonador.mt "1234"
2
3      Simulador de Máquina de Turing Suave ver 1.0
4      Desenvolvido como trabalho prático para a disciplina de Teoria da Computação
5      Autor Fulano de Tal, IFMG — Formiga, 2020.
6
7      fitaX: [1]234
8      fitaY: [ ]
9      fitaZ: [ ]
10
11     Opção? (n=passos, 0=termina, -1=resume): _
```

4. Saída:

Finalizado o programa, o simulador imprime a palavra ACEITA ou REJEITA, caso a execução tenha terminado ou não em estado final, seguido do conteúdo das três fitas. Exemplo:

```
1  prompt> simuladorMT clonador.mt "1234"
2
3  Simulador de Máquina de Turing Suave ver 1.0
4  Desenvolvido como trabalho prático para a disciplina de Teoria da Computação
5  Autor Fulano de Tal, IFMG – Formiga, 2020.
6
7  ACEITA
8  _____
9  fitaX: 1234[ ]
10 fitaY: 1234[ ]
11 fitaZ: 1234[ ]
12 _____
13 FIM DA SIMULAÇÃO.
```

Para decrever o formato da execução passo a passo, caso seja a opção `-debug` escolhida pelo usuário, o simulador imprime para cada instrução executada uma linha no formato::

<bloco> : <instrução>

Descrição dos campos:

- <bloco> identificador do bloco em execução, máximo de 16 caracteres significativos.
- <instrução> comando executado.

Exemplo do relatório de saída para examinar a execução o código no caso de `-debug`:

```
1  prompt> simuladorMT -debug clonador.mt "1234"
2
3  Simulador de Máquina de Turing Suave ver 1.0
4  Desenvolvido como trabalho prático para a disciplina de Teoria da Computação
5  Autor Fulano de Tal, IFMG – Formiga, 2020.
6
7  .....main : 01 X $d i — 03 X $d i
8  .....main : 03 copiaX 04
9  .....copiaX : 01 X $d i — 03 Y $d e
10 .....copiaX : 03 X $d e — 01 Z $d e
11 .....copiaX : 01 X $d i — 03 Y $d e
12 .....copiaX : 03 X $d e — 01 Z $d e
13 .....copiaX : 01 X $d i — 03 Y $d e
14 .....copiaX : 03 X $d e — 01 Z $d e
15 .....copiaX : 01 X $d i — 03 Y $d e
16 .....copiaX : 03 X $d e — 01 Z $d e
17 .....copiaX : X * i — 02 X * i
18 .....copiaX : 02 retorne
19 .....main : 04 aceita
```