Universidade Federal de Viçosa – Campus UFV-Florestal
Ciência da Computação – Fundamentos da Teoria da Computação
Professor: Daniel Mendes Barbosa
Trabalho Prático 3

Neste trabalho **você e seu grupo (definidos em aula da disciplina nesta planilha)** deverão implementar um programa que seja capaz de, a partir da entrada, configurar um autômato finito determinístico (AFD) e logo em seguida utilizá-lo para aceitar ou rejeitar palavras de acordo com a linguagem configurada. Vocês deverão também implementar **no mínimo** uma funcionalidade extra, dentre as possibilidades listadas ao final deste documento. Quanto mais funcionalidades implementadas, melhor poderá ser considerado o seu trabalho. Tudo deve ser devidamente documentado e serão **duas entregas cadastradas no Google Classroom**: um PDF, com a documentação e um ZIP com todo o projeto. Além disso vocês podem implementar mais detalhes que não estejam especificados aqui, desde que devidamente documentados, e até mesmo outras opções de interface. No fim do semestre está prevista uma aula onde cada grupo fará uma breve apresentação do seu trabalho.

Obs.: a linguagem de programação pode ser escolhida pelo grupo e deve ser justificada na documentação.

Especificação

Entrada

Assuma que o alfabeto do autômato é composto apenas de `0` e `1`, e todos os estados estarão presentes, inclusive o estado de erro. Seu programa deve então receber uma entrada no seguinte formato:

- Na primeira linha, constam os caracteres `Q:` , seguidos de um espaço, e depois são especificados os nomes de cada estado do autômato, separados por espaços. Os nomes dos estados são compostos apenas de caracteres de `a` a `z`, `A` a `Z`, e de `0` a `9`, com no máximo 7 caracteres, em qualquer ordem.

- Na segunda linha, constam os caracteres `I:`, seguidos de um espaço, e depois é especificado o estado inicial, cujo nome é um dos estados listados na linha anterior.

- Na terceira linha, constam os caracteres `F:`, seguidos de um espaço, e depois são especificados os estados finais, cujos nomes estão entre os estados listados na primeira linha.

- Nas linhas seguintes, é definida a função de transição, sendo que em cada linha uma ou mais transições são definida. Cada transição é composta pelo nome de dois estados separados por espaços e por uma seta à direita (`->`), depois uma

barra vertical (`|`), e em seguida uma lista de caracteres, separados por espaço, que são os caracteres de entrada sobre os quais aquela transição ocorre.

- O fim da definição da função de transição ocorre numa linha contendo apenas `---`, encerrando assim a definição também do autômato.
- As linhas seguintes definem, cada uma, um caso de teste, compostos apenas pelos símbolos do alfabeto de entrada, `0` e `1`. Lembrando que pode ocorrer também a palavra vazia, representada nos casos de teste por uma linha sem caracteres além do `\n`.
- A entrada acaba com o fim do arquivo.

Exemplo

Aqui é especificado um AFD que reconhece a linguagem denotada pela expressão regular $\lambda + (10^{+}1 + 0)(0 + 1)^{*}$, seguido de alguns casos de teste. Repare que a terceira palavra a ser testada é a palavra vazia. Veja também que a transição `d -> d` na verdade se trata de duas transições, uma em `0` e outra em `1`. O mesmo ocorre com as transições `e -> e`.

```
Q: a b c d e
I: a
F: a d
a -> b | 1
a -> d | 0
b -> c | 0
b -> e | 1
c -> c | 0
c -> d | 1
d -> d | 0 1
e -> e | 0 1
---
001
1000

10000011
111
1
```

Saída

A saída do seu programa deve ser a resposta de cada caso de teste, apenas. Para cada linha de um caso de teste, se a palavra passada for aceita pelo autômato, imprima `OK` na saída, ou, se for rejeitada, imprima `X` na saída. Cada resultado também deve estar em uma linha separada.

Exemplo

Para o exemplo de entrada acima, a saída correspondente seria:

```
OK
X
OK
OK
X
X
```

Extras

Como supracitado, é necessário que seja feita no mínimo uma funcionalidade extra dentre as seguintes opções, sendo livre a escolha. Em todos os exemplos mostrados, parte da entrada está omitida com reticências, por motivos de simplicidade.

(1) Alfabeto de entrada

Permitir com que o autômato use um alfabeto arbitrário, especificado também na entrada junto com seus outros parâmetros. Logo após a definição dos estados, na primeira linha, uma nova linha será especificada, com os caracteres `S:`, seguidos de um espaço, e logo em seguida uma sequência de caracteres, sendo cada um deles um símbolo do alfabeto. Esses caracteres não serão separados, ou seja, considere todos os caracteres que aparecem nessa linha como símbolos de entrada, exceto, claro, o `\n`, que denota o fim da linha.

Exemplos

Observe que, no terceiro exemplo, o caractere de espaço é considerado símbolo do alfabeto.

```
Q: a b c d e
S: 0123456789
...
```

```
Q: q0 q1 q2
S: abc
...
```

```
Q: s1 s2
S:  ><+-.[]
...
```

(2) Não determinismo

Introduzir a possibilidade de não determinismo no seus autômatos. Neste caso, ao especificar o estado inicial, na verdade pode ser especificado mais de um estado. Além disso, para possibilitar transições , essa letra grega é representada nas transições como uma contrabarra (`\`). Note que ao fazer isso, caso vocês queiram também implementa o extra 1, o símbolo `\` não pode estar presente no alfabeto de entrada.

Exemplo

```
...
a -> b | 0
a -> c | 0
a -> d | \
...
```

(3) Autômato de Pilha (AP)

Ao invés de implementar um autômato finito, implementar um autômato de pilha, considerando que as palavras são reconhecidas quando lidas completamente **e** a pilha está vazia.

Após a definição dos estados (a linha que contém `Q:`), ou, caso vocês tenham implementado o extra 1, após a definição do alfabeto de entrada (a linha que contém `S:`), deve ser definido o alfabeto da pilha, representado pela linha que começa com `G:`. As regras são as mesmas para o alfabeto de entrada.

Atenção também ao formato das transições. Sendo `a` o símbolo a ser lido na fita de entrada, `b` o símbolo a ser desempilhado, e `z` uma sequência de símbolos a serem empilhados, cada transição é definida na forma `a,b/z`. Note que mesmo sendo determinístico, o λ pode aparecer, usando então contrabarra, `\`, para representá-lo.

```
Q: igual dif
G: ZU
I: igual
F: dif
igual -> dif | 0,\/Z 1,\/U
dif -> igual | \,\/\
dif -> dif | 0,Z/ZZ 1,Z/\
dif -> dif | 0,U/\ 1,U/UU
...
```

(3) Máquina de Turing (MT)/Autômato Linearmente Limitado (ALL)

Ao invés de implementar um autômato finito, implementar uma máquina de Turing/autômato linearmente limitado, considerando que as palavras são reconhecidas quando a máquina para em um estado final.

Após a definição dos estados (a linha que contém `Q:`), ou, caso vocês tenham implementado o extra 1, após a definição do alfabeto de entrada (a linha que contém `S:`), deve ser definido o alfabeto da fita, representado pela linha que começa com `G:`. Formalmente, o alfabeto de entrada é um subconjunto do alfabeto da fita, mas, para

simplificar, considere a linha do `G:` como apenas os símbolos exclusivos do alfabeto da fita. Por exemplo, é possível considerar apenas `0` e `1` como o alfabeto de entrada, e permitir que a máquina escreva outro símbolo como `a` ou `b`. As regras de como o alfabeto da fita é descrito são as mesmas do alfabeto de entrada, com algumas observações. Não se pode admitir `_`, `<` e `>` como símbolos nos alfabetos de entrada ou da fita, pois esses caracteres devem representar, respectivamente: um espaço em branco na fita, o limite à esquerda da fita, e o limite à direita da fita.

Sobre o formato das transições: seja `a` o símbolo a ser lido da fita, `b` o símbolo a ser escrito na fita, e `d` uma direção (`E` para esquerda ou `D` para direita), cada transição é definida na forma `a/bd`.

Na prática, como todo computador possui memória finita, mesmo que vocês implementem uma MT, essa seria um ALL. Nesse caso, considere um número suficientemente grande de espaços para a fita (ex.: 10^9), e caso a máquina "extrapole" a fita, considere que a palavra não foi reconhecida.

Por fim, a saída do seu programa deve ter um adicional: ao lado da informação de palavra aceita/rejeitada, deve ser impresso o conteúdo da fita. No caso da MT, como a finita é teoricamente infinita, imprima apenas até o último espaço não vazio da fita.

Exemplos

Uma MT que apaga `0` e alternadamente troca `1` por `a` ou `b`, reconhecendo a palavra apenas se a quantidade de `1` for par:

Entrada:

```
Q: A B
G: a b
I: A
F: B
A -> A | 0/_D
A -> B | 1/aD
B -> A | 1/bD
B -> B | 0/_D
---
1000111
0111000
```

Saída:

```
OK <a___bab
X  <_aba
```

Um ALL que apaga `0` da entrada:

Entrada:

```
Q: x fim
G:
I: x
F: fim
x -> x | 0/_D 1/1D
x -> fim | >/>E
---
1000111
0111000
```

Saída:

```
OK <1___111>
OK <_111___>
```

(5) Múltiplos tipos

Permitir que o tipo do autômato seja também um parâmetro a ser escolhido na entrada, de forma que seja possível escolher entre autômato finito (determinístico ou não), autômato de pilha (determinístico ou não) ou máquina de Turing. Note que para realizar este extra, vocês devem já ter feito o extra 2, 3, e/ou 4. Não é necessário que todos os possíveis tipos sejam implementados, mas deve ser especificado quais são permitidos pelo seu programa, sendo no mínimo dois. Ex.: vocês podem possibilitar AFs e APs, ou AFs e ALLs, ou qualquer outra combinação.

Para indicar qual tipo de autômato será descrito na entrada, a primeira linha da entrada será `@TIPO`, sendo `TIPO` alguma das opções: `AF` (autômato finito), `AP` (autômato de pilha), `ALL` (autômato linearmente limitado) ou `MT` (máquina de Turing).

```
@AF
Q: a b c d e
...
```

```
@MT
Q: a b
...
```

Observações gerais

- Caso deseje implementar máquinas que podem nunca parar de executar, vocês podem estabelecer um tempo limite para que seus autômatos executem, de forma que se esse tempo for excedido, a execução é interrompida e a palavra é considerada rejeitada.
- Atenção também às condições de reconhecimento alternativas, que podem ser implementadas desde que isso seja especificado no trabalho.