Relatório sobre Autômatos finitos e Transdutores

Braz G. S. Souza[1], Davi B. Franco[1], Gabriel S. Amaral[1]

[1]Faculdade de Computação – Universidade Federal do Pará (UFPA)

Caixa postal 479. PABX +55 91 3201-7000. Belém - Pará - Brasil

{braz.souza,davi.franco,gabriel.amaral}@icen.ufpa.br

Resumo. *Este artigo é um conjunto de programas para a resolução de problemas através do uso de autômatos e transdutores finitos, se utilizando da linguagem de programação Python. Começando com testes de arranjos dadas certas expressões regulares, autômatos que reconhecem uma palavra em um dado texto, e por fim um transdutor que retorna um valor dado uma lista de entrada. Se utilizando de testes experimentais para comprovar a funcionalidade dos programas desenvolvidos.*

Abstract. *This article is a set of programs for solving problems using finite automata and transducers, using the Python programming language. Starting with tests of arrangements given certain regular expressions, automata that recognize a word in a given text, and finally a transducer that returns a value given a list of inputs. With the use of experimental tests to prove the functionality of the programs developed.*

1. Introdução

Primeiramente, para poder entender os programas produzidos a seguir, é necessário ter conhecimento de o que são autômatos e transdutores. Os autômatos são dispositivos de aceitação de sentenças, podendo ser gerados tomando como base gramáticas regulares ou conjuntos regulares. O autômato, também conhecido como reconhecedor, pode ser observado na figura 1, tendo 4 partes fundamentais, a fita que contém a cadeia a ser analisada, o cursor que faz a leitura dos símbolos gravados na fita, a máquina de estados que é o controlador central do reconhecedor, contendo uma coleção finita de estados necessárias para determinar escolhas futuras do autômato. Em linguagens mais complexas são utilizadas uma memória auxiliar, possuindo seu próprio alfabeto de memória e sendo uma estrutura de dados em forma de pilha, também é importante destacar que no autômato finito não ocorre a utilização de memória auxiliar.

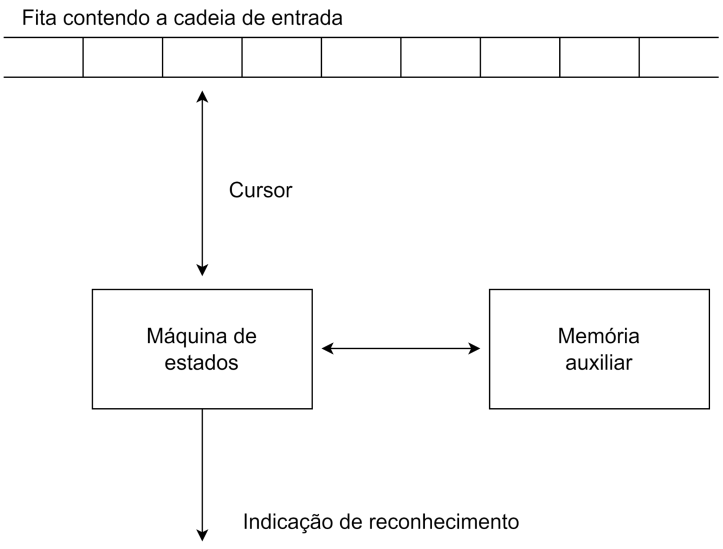

Figura 1. Forma geral de um reconhecedor

Existem autômatos determinísticos e não determinísticos, sendo diferenciado pela existência de duas ou mais transições de um estado em que ocorra o surgimento de um mesmo símbolo, ocorrendo uma indeterminação de qual estado o autômato deve seguir, sendo necessário analisar ambos os casos, o autômato que ocorre o caso citado é chamado de não determinístico.

Um exemplo do caso de duas ou mais transições surgindo de um mesmo estado pode ser observado na figura 2.

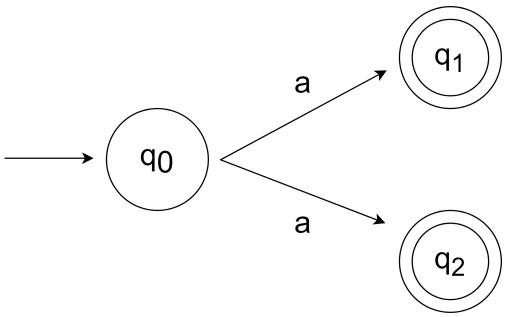

Figura 2. Um Autômato Não Determinístico com duas transições "a" de q_0

Entre os autômatos não determinísticos, existem os autômatos com transição em vazio, em que admitem transições de um estado para o outro com ε, sendo executadas sem consultar o símbolo corrente na fita e sem deslocar o cursor de leitura, podendo coexistir mais de uma transição em vazio de um mesmo estado. Por transições em vazio serem exclusivas de autômatos não determinísticos, não existem autômatos determinísticos com transição em vazio.

Um autômato finito determinístico M é definido algebricamente pela equação 1

$$M = (Q, \Sigma, \delta, q0, F)$$

Equação 1

Em que o "Q" é o conjunto de estados do autômato, "Σ" e o alfabeto de entrada, "δ" é uma função de transição, "q0" é o estado inicial, "F" o conjunto de estados finais. Tendo em vista que "Q" e "Σ" são ambos finitos, e "Σ" é não-vazio, e "q0" e todos os elementos contidos em "F" devem estar obrigatoriamente contidos em "Q".

Além disso, para a compreensão da solução da questão 3, é fundamental o conhecimento de transdutores, que são extensões da aplicação dos autômatos, sendo um dispositivo que retorna uma cadeia de saída dada uma sentença de entrada, podendo haver seu próprio alfabeto para escrever a cadeia de saída, sendo associado de duas formas, por estados, sendo chamadas Máquinas de Moore, ou por transições, as Máquinas de Mealy.

A Máquina de Moore é definida na equação 2, e a Máquina de Mealy na equação 3, sobre o autômato da equação 4. Em que Δ é o alfabeto de saída do transdutor, enquanto λ é a função de transdução do transdutor.

$$T_{Moore} = (Q, \Sigma, \Delta_{Moore}, \delta, \lambda_{Moore}, q0, F)$$

Equação 2

$$T_{Mealy} = (Q, \Sigma, \Delta_{Mealy}, \delta, \lambda_{Mealy}, q0, F)$$

Equação 3

$$M = (Q, \Sigma, \delta, q0, F)$$

Equação 4

A principal diferença entre as duas formas de transdutores finitos é a maneira em que ocorre a geração da cadeia de saída, em que a Máquina de Moore se baseia pelos estados percorridos do autômato para obter uma cadeia, dado os símbolos do alfabeto do transdutor, finalizando a cadeia de saída quando a de entrada finaliza, um exemplo de uma Máquina de Moore pode ser observada em sua forma algébrica na equação 5, e em figura na figura 3.

$$T = (Q, \Sigma, \Delta, \delta, \lambda, q0, F)$$

$$Q = \{q0, q1\}$$

$$\Sigma = \{a, b, c\}$$

$$\Delta = \{1\}$$

$$\delta = \{(q0, a) \rightarrow q1, (q1, b) \rightarrow q1, (q1, c) \rightarrow q0\}$$

$$\lambda = \{q0 \rightarrow 1, q1 \rightarrow \varepsilon\}$$

$$F = \{q1\}$$

Equação 5

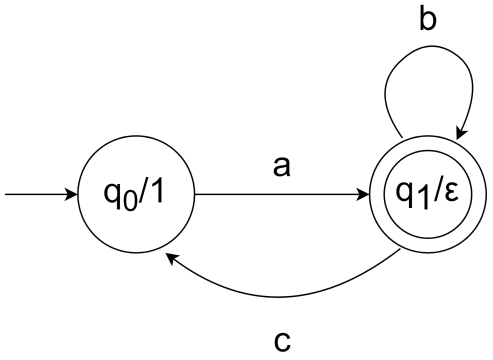

Figura 3. Máquina de Moore da Equação 5

Enquanto a Máquina de Moore utiliza os estados para gerar a cadeia de saída, a Máquina de Mealy se utiliza da associação aos símbolos do alfabeto de saída às transições do autômato para a formação da cadeia de saída do transdutor. Com isso, um exemplo para facilitar a compreensão do transdutor é apresentado em sua forma algébrica na equação 6, e em sua forma desenhada como figura na figura 4.

$$T = (Q, \Sigma, \Delta, \delta, \lambda, q0, F)$$

$$Q = \{q0, q1\}$$

$$\Sigma = \{a, b, c\}$$

$$\Delta = \{a, b, c\}$$

$$\delta = \{(q0, a) \rightarrow q1,(q1, b) \rightarrow q1,(q1, c) \rightarrow q0\}$$

$$\lambda = \{(q0, a) \rightarrow ab,(q1, b) \rightarrow \varepsilon,(q1, c) \rightarrow c\}$$

$$F = \{q1\}$$

Equação 6

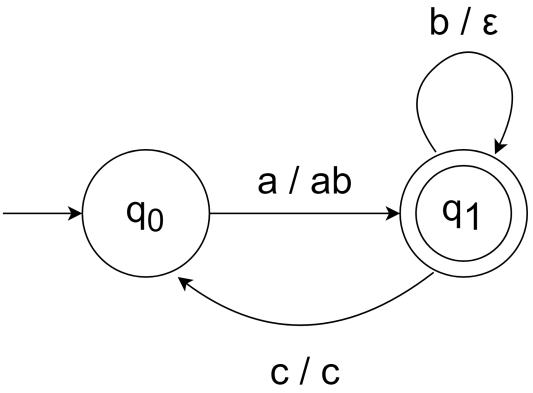

Figura 4. Máquina de Mealy da Equação 6

Dado que existem duas formas de gerar um transdutor finito, é importante destacar que toda Máquina de Mealy pode ser simulada por uma Máquina de Moore, e vice-versa. Dessa maneira, é possível escolher qualquer uma das duas maneiras apresentadas para desenvolver um transdutor para um autômato.

Na questão 1 foi pedido a formação de autômatos finitos que reconheçam cadeias pertencentes a linguagem, sem conter não-determinismos, transições em vazio, estados inacessíveis e nem estados inúteis.

Na questão 2 é solicitado a implementação de um autômato em que ocorre o reconhecimento de todas as ocorrências da palavra "computador" no texto T, e por fim apontar todas as posições em que ocorreram o casamento da palavra, desconsiderando palavras compostas por computador como "microcomputador" ou "computadores".

Na questão 3 é requisitada a produção de um transdutor finito, que funciona como uma máquina de refrigerante, que dada uma sequência de moedas fornece uma lata de refrigerante quando a sequência totaliza 1 real.

2. Materiais e métodos

Para desenvolver as questões utilizamos a linguagem de programação Python, em sua versão 3.11.2, que é muito usada em aplicações da web, desenvolvimento de software, machine learning e em ciência de dados. Entre os motivos da escolha da linguagem neste trabalho, se destacam a sua facilidade em aprendizado e por conseguinte sua praticidade em resolver problemas reais, assim como o conhecimento da linguagem por todos os integrantes do grupo.

Ademais, durante a criação do código, foi utilizado o Visual Studio Code, um ambiente de desenvolvimento integrado para facilitar a produção e testagem do código durante a sua criação. Além disso, também foram utilizados também o subsistema do Windows para Linux (WSL), versão Ubuntu 22.04.2 LTS, para a execução dos programas, e o GitHub, plataforma para hospedagem de código-fonte, sendo utilizado como repositório para os códigos fontes desenvolvidos, assim como o relatório e as imagens presentes neste. E por fim, para o planejamento e desenvolvimento do texto foi utilizado o google documentos devido a sua facilidade para manter o arquivo em conjunto, e da plataforma Overleaf para diagramar o texto e organizar as imagens apresentadas.

Para separar o código de uma maneira mais compreensível, serão utilizados 4 arquivos para produzir o código, sendo um para cada questão, e por fim uma questão de testes experimentais que importará as funções desenvolvidas nas questões anteriores, fazendo testes para garantir que o código produzido esteja de acordo com o desejado.

3. Autômato finito que reconheça cadeias pertencentes a linguagem.

A primeira questão pedia a formulação de autômatos finitos para dadas expressões regulares, para a resolução será feito um autômato finito, determinístico sem transições em vazio, estados inacessíveis e nem estados inúteis. Para então, utilizar o autômato para desenvolver uma função que age como um reconhecedor, retornando um valor booleano, sendo "True" para o caso da cadeia ter sido reconhecida, e "False" caso a cadeia não tenha sido reconhecida pelo autômato desenvolvido na função, se utilizando de ifs para verificar o estado atual e fazer ações dependendo do caso atual, e recursão para repetir a ação até que a cadeia analisada seja vazia ou seja reconhecida como inválida.

a) (ab*c*)*

Com a expressão regular dada pela parte "a)" da primeira questão, é possível converter para um autômato determinístico, que pode ser observado na figura 5.

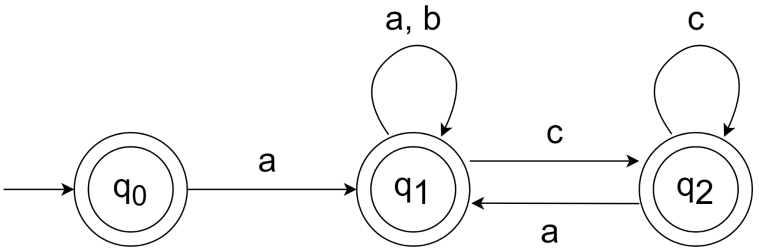

Figura 5. Autômato a)

Dessa forma, é possível gerar uma função em que tem como parâmetros a cadeia a ser analisada, e o estado atual com valor padrão "q0". Iniciando a função, é gerado uma função contendo os estados finais, e após uma verificação se a cadeia é vazia, e no caso de ser vazia, retorna se o estado atual está presente na lista de estados finais, observe trecho de código na figura 6 para observar o código o desenvolvido até o momento.

```
def automato_a(cadeia, estado_atual="q0"):
    estados_finais = ["q0", "q1", "q2"]
    if cadeia == "":
        return estado_atual in estados_finais
```

Figura 6. Trecho inicial do código do autômato a)

E após isso, o programa deve verificar cada um dos estados, de "q0" até "q2", utilizando-se de ifs. Para o caso de "q0", a função deve verificar se o primeiro item da cadeia, ou seja cadeia[0], é igual ao valor de "a", o único valor que percorre de q0, e no caso do valor ser igual a "a", deve ser feito o retorno do valor apresentado pela função sendo desenvolvida atualmente, passando como argumentos a cadeia sem o primeiro elemento e o novo estado atual, que é "q1", e no caso do elemento inicial da cadeia não ser "a" a função deve retornar "False", com isso terminando o caso do estado atual sendo "q0" com o trecho de código desenvolvido na figura 7.

```
if estado_atual == "q0":
    if cadeia[0] == "a":
        return automato_a(cadeia[1:], "q1")
    return False
```

Figura 7. Trecho if para estado atual igual a "q0" para o autômato a)

Para desenvolver os outros estados, será feito de uma forma parecida, verificando o estado desejado utilizando se do “if”, e para cada produção do estado deve ser gerado outro “if” checando o valor de “cadeia[0]” ao valor passado da produção, e caso seja verdadeiro deve retornar a função se utilizando como argumentos o resto da cadeia sem o elemento de index 0, e o segundo argumento sendo o estado que se atinge ao utilizar a letra verificada no “if” anterior, e no fim de todos os “ifs” gerados pelas produções do autômato, deve ser retornado False, demonstrando que o valor não é um valor possível, demonstrando que a cadeia não foi reconhecida. Para uma melhor compreensão é possível ver o código completo na figura 8, que se utiliza da mesma lógica utilizada para desenvolver o caso do estado atual como “q0” para desenvolver o caso do estado atual como “q1” e “q2”.

```
def automato_a(cadeia, estado_atual="q0"):
    estados_finais = ["q0", "q1", "q2"]
    if cadeia == "":
        return estado_atual in estados_finais

    if estado_atual == "q0":
        if cadeia[0] == "a":
            return automato_a(cadeia[1:], "q1")

    if estado_atual == "q1":
        if cadeia[0] == "a":
            return automato_a(cadeia[1:], "q1")
        if cadeia[0] == "b":
            return automato_a(cadeia[1:], "q1")
        if cadeia[0] == "c":
            return automato_a(cadeia[1:], "q2")

    if estado_atual == "q2":
        if cadeia[0] == "a":
            return automato_a(cadeia[1:], "q1")
        if cadeia[0] == "c":
            return automato_a(cadeia[1:], "q2")
    return False
```

Figura 8. Código completo gerado para analisar uma cadeia como o autômato a)

b) aaa(b | c)* | (b | c)* aaa

Para desenvolver essa questão é necessário criar um autômato para o conjunto regular dado. Tal autômato para produzir a função pedida pela questão pode ser observado na figura 9.

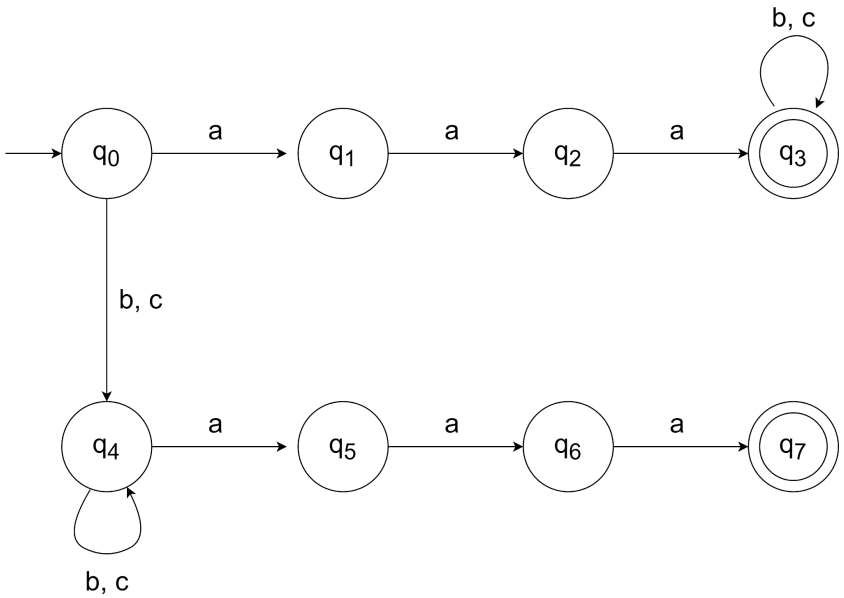

Figura 9. Autômato b)

Tendo o autômato da figura 9, é possível gerar uma função que ocorre como esse autômato. Para desenvolver o código de uma maneira mais simples é possível obter o conhecimento da parte “a)” da primeira questão para solucionar as seguintes, separando a função em 4 partes:

- A definição e parâmetros da função.
- A criação de estados finais e a finalização do autômato no caso da cadeia vazia, retornando True se o estado atual faz parte dos estados finais, e False em caso contrário.
- Produzir ifs para cada um dos estados que têm transições da saída, com um if para cada símbolo da saída, com a expressão dentro do if sendo um return da própria função com o resto da cadeia sem o primeiro símbolo, e segundo parâmetro o estado que recebe a transição. Com um return False no final de todos os ifs, representando a falha de reconhecimento.
- Colocar um “return False” no final de todos os ifs checando o estado atual, demonstrando que o estado atual não pertence a nenhum estado que ocorre um símbolo de transição de saída.

Após definir cada a função “automato_b”, tendo como parâmetros a cadeia a ser analisada e o estado atual que tem como valor padrão “q0”, é necessário observar os estados finais do autômato, e gerar uma lista com eles, e por fim o if que verifica se a cadeia está vazia e retorna um booleano respectivo ao pertencimento do estado atual a lista de estados finais.

```
def automato_b(cadeia, estado_atual="q0"):
    estados_finais = ["q3", "q7"]
    if cadeia == "":
        return estado_atual in estados_finais
```

Figura 10. Trecho inicial do código do autômato b)

Dessa forma, para gerar o restante do código é necessário observar as transições, para assim gerar todos os ifs para cada um dos estados possíveis, obtendo a função “automato_b”, que não será apresentada em figura devido a grande quantidade de estados. Todavia, o código desenvolvido em questão pode ser observado no código fonte presente no Apêndice A.

c) a*b | ab*

Para gerar uma função que funciona como autômato para a linguagem dada pela questão, é necessário transformar a expressão "a*b | ab*" em um autômato finito determinístico, que resulta na figura 11

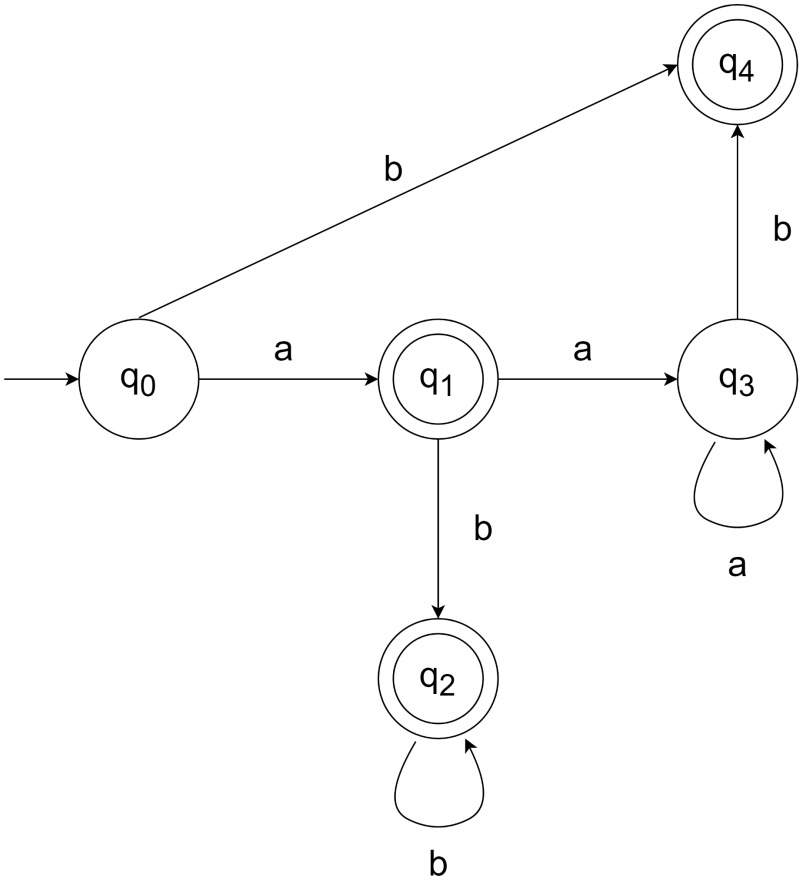

Figura 11. Desenho do autômato c)

Com o autômato, é possível gerar uma função que utiliza-se de recursão e ifs para verificar o estado atual e como agir dependendo dele, repetindo até o fim da análise da cadeia ou a função retornar False por encontrar alguma atitude inválida. O código desenvolvido pode ser observado na figura 12.

```python
def automato_c(cadeia, estado_atual="q0"):
    estados_finais = ["q1", "q3", "q4"]
    if cadeia == "":
        return estado_atual in estados_finais

    if estado_atual == "q0":
        if cadeia[0] == "a":
            return automato_c(cadeia[1:], "q1")
        if cadeia[0] == "b":
            return automato_c(cadeia[1:], "q3")

    if estado_atual == "q1":
        if cadeia[0] == "a":
            return automato_c(cadeia[1:], "q2")
        if cadeia[0] == "b":
            return automato_c(cadeia[1:], "q4")

    if estado_atual == "q2":
        if cadeia[0] == "a":
            return automato_c(cadeia[1:], "q2")
        if cadeia[0] == "b":
            return automato_c(cadeia[1:], "q3")

    if estado_atual == "q4":
        if cadeia[0] == "b":
            return automato_c(cadeia[1:], "q4")
    return False
```

Figura 12. Código completo gerado para o analisar uma cadeia como o autômato c)

d) a*b* (a | ac*)

Por fim, o último autômato pedido na primeira questão é o que tem como linguagem "a*b* (a | ac*)", para obter a função que age como autômato finito determinístico é necessário transformar essa linguagem em autômato finito determinístico, obtendo a figura 13.

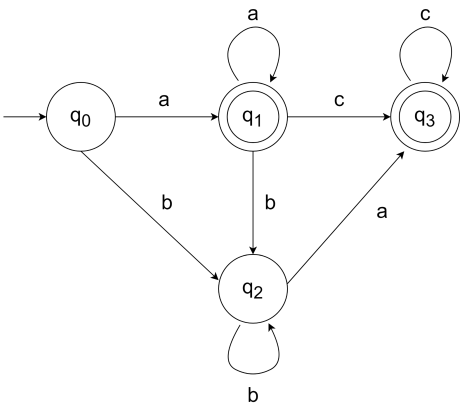

Figura 13. Desenho do autômato d)

Agora para desenvolver a função que age como o autômato da figura 13 basta observar as transições, utilizando se da recursão para analisar cada elemento da cadeia, e ifs para verificar o estado atual e do elemento atual da cadeia, para assim fazer a ação adequada até o fim da análise da cadeia fornecida, ou até a função retornar False por alguma ocorrência imprevista no programa. O código criado utilizando-se dessa estratégia pode ser observado na figura 14.

```python
def automato_d(cadeia, estado_atual="q0"):
    estados_finais = ["q1", "q3"]
    if cadeia == "":
        return estado_atual in estados_finais

    if estado_atual == "q0":
        if cadeia[0] == "a":
            return automato_d(cadeia[1:], "q1")
        if cadeia[0] == "b":
            return automato_d(cadeia[1:], "q2")

    if estado_atual == "q1":
        if cadeia[0] == "a":
            return automato_d(cadeia[1:], "q1")
        if cadeia[0] == "b":
            return automato_d(cadeia[1:], "q2")
        if cadeia[0] == "c":
            return automato_d(cadeia[1:], "q3")

    if estado_atual == "q2":
        if cadeia[0] == "a":
            return automato_d(cadeia[1:], "q3")
        if cadeia[0] == "b":
            return automato_d(cadeia[1:], "q2")

    if estado_atual == "q3":
        if cadeia[0] == "c":
            return automato_d(cadeia[1:], "q3")
    return False
```

Figura 14. Código completo gerado para analisar uma cadeia como o autômato d)

4. Autômato finito para reconhecer todas as ocorrências da palavra computador em um texto.

A segunda questão pede a formulação de um autômato em que reconhece as ocorrências da palavra "computador" dado um texto T de forma exata, sem considerar palavras que contêm a string "computador" como "microcomputador" e "computadores", e apontando as devidas posições em que ocorreram o casamento exato da string. Sendo utilizado a linguagem de programação Python para desenvolver o código para o autômato pedido. Para fazer tal função, é fundamental criar um autômato para facilitar a compreensão do que foi pedido e também para poder criar o código de uma maneira mais compreensível.

Para gerar tal autômato, é necessário considerar o alfabeto como todos os caracteres possíveis, incluindo o espaço " ", Σ = Todos os caracteres possíveis. Tendo isso em vista, deve-se destacar que a palavra computador não deve estar conectada entre nenhuma outra palavra, sendo separada por espaços, como se observa no exemplo em que microcomputador não é reconhecido pelo autômato, podemos considerar o texto pode começar e terminar com a palavra computador para ser reconhecida mas em qualquer outro caso ele necessita ter dois espaços, um no começo e outro no fim, para poder ser considerado como casamento exato. Dessa forma, para gerar o autômato, o q0 será a raiz, ao mesmo tempo em que vai para q1 no

caso de c e q11 no caso de qualquer outra palavra do alfabeto, q11 sendo o caso em que o autômato se repete esperando acontecer a primeira possibilidade da palavra computador, ou seja a string espaço " ". Ao gerar q11, é necessário que ele retorne a si mesmo no caso de qualquer outra string além de " ", e no caso " " ele vá para q0, representando a possibilidade de haver a string " computador ", e q0 até q10 ocorre a transferência de letra por letra da palavra computador, de q0 para q1 ocorrendo c, q1 para q2 ocorrendo o, q2 para q3 ocorrendo, q3 para q4 ocorrendo p, q4 para q5 ocorrendo u, q5 para q6 ocorrendo t, q6 para q7 ocorrendo a, q7 para q8 ocorrendo d, q8 para q9 ocorrendo o, q9 para q10 ocorrendo r, e cada um dos q1 a q9 deve retornar para q11 no caso de qualquer valor diferente do esperado. Todavia, ocorre uma mudança diferente para q10, em que ocorre uma string com espaço indo para q0, e ao ir para q0 a função deve considerar uma palavra reconhecida, e ir para q11 no caso de qualquer outra string sem ser uma string com espaço. Se observa em que esse autômato deve haver fechamento em todas as possibilidades, pois todos os textos a serem analisados podem acabar em qualquer instante, sendo o ponto essencial a ser destacado quando ocorre a mudança de q10 para q0, que apresenta o casamento exato. Dessa forma, é possível observar o autômato gerado com a lógica desenvolvida na figura 15.

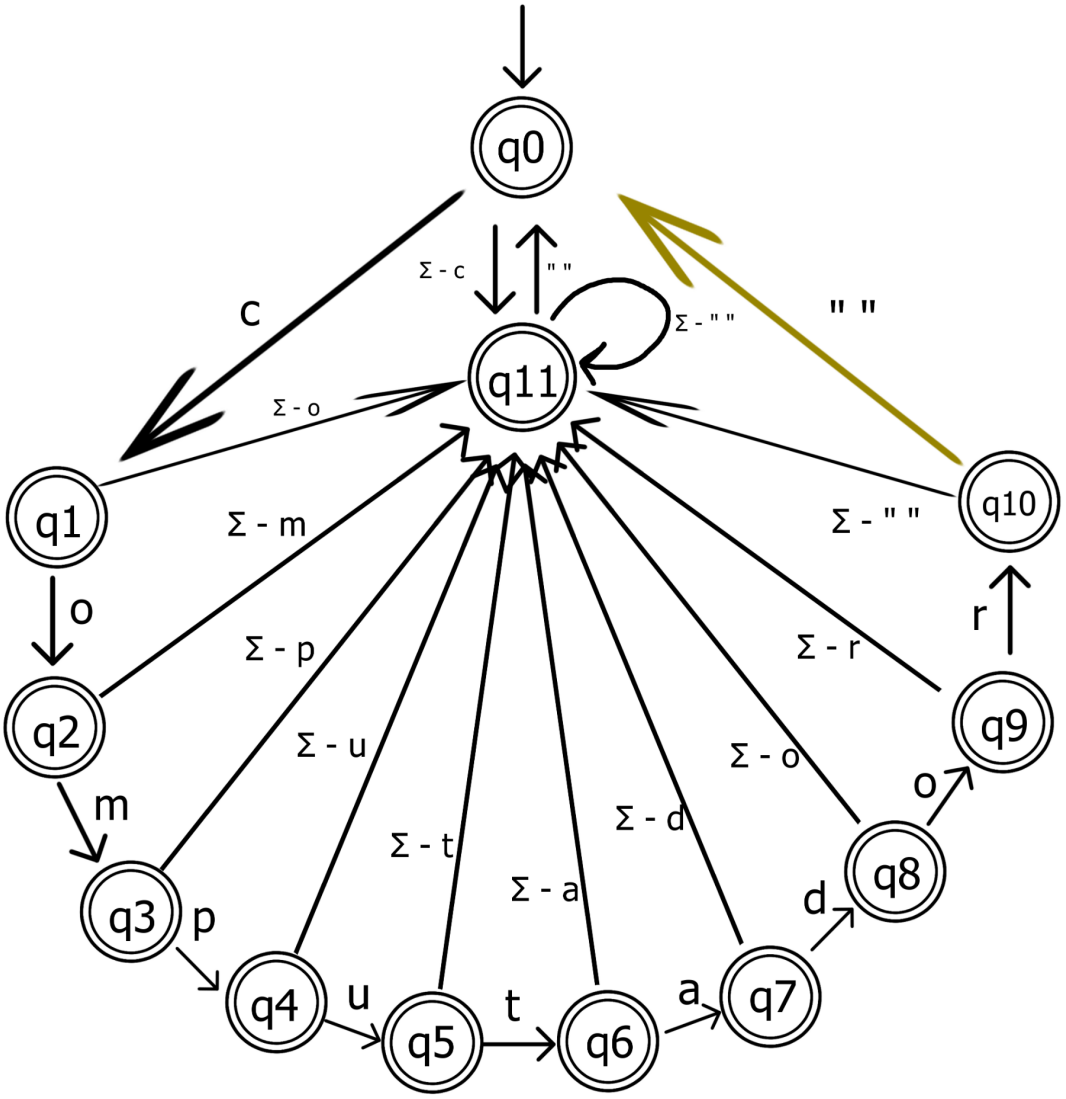

Figura 16. Autômato para a função que irá encontrar a palavra computador

Primeiramente, para poder criar essa função é importante destacar os parâmetros que serão utilizados e o que a função irá retornar. Tendo isso em vista, os parâmetros a serem fornecidos para a função é o texto T em que o autômato irá buscar o reconhecimento da palavra "computador", o estado atual, que tem como valor default igual a “q0”, uma variável auxiliar pos, que tem como valor default igual a 0, o valor inicial, e uma lista para guardar os casamentos exatos da palavra reconhecida, tendo como valor default igual a uma lista vazia, além do index da função, que aumenta na medida que a função repete sobre si mesmo, tendo como valor padrão igual a 0, e necessita retornar uma lista, a lista de posições em que houve o casamento exato da palavra reconhecida. Dessa forma, para poder gerar um bom código que solucione o problema apresentado, é necessário criar um autômato para se utilizar deste para desenvolver uma função que consiga retornar os devidos valores pedidos na questão. Pode-se observar o parâmetro que foi gerado anteriormente na figura 16.

```
def automato_computador(cadeia, estado_atual="q0", ocorrencias=[], pos=0, i = 0):
```

Figura 16. Definição da função e parâmetros para a função

Para desenvolver a função com base no autômato gerado na figura 1, é importante destacar que a função somente irá reconhecer uma palavra na transição do q10 para q0 ou o fim da cadeia ocorrer no q10. Com isso é necessário, no começo da função, verificar se a cadeia está vazia, dentro dessa verificação, deverá ocorrer outra verificação, para saber se o estado atual é igual ao estado q10, no caso do estado ser igual a q10, significa que a variável “pos” na função é um casamento exato da palavra “computador”, onde o valor é o index inicial onde começa a palavra “computador”, sendo necessário por a variável dentro da lista de ocorrências da palavra computador. Por fim, ainda dentro da verificação para saber se a cadeia está vazia, mas fora da verificação do estado atual, a função deve retornar a lista de ocorrências, demonstrando um fim da recursão da função no caso da cadeia vazia, como observado pela figura 17.

```
    if cadeia == "":
        if estado_atual == "q10":
            ocorrencias.append(pos)
        return ocorrencias
```

Figura 17. Definição da parada do autômato e retorno das ocorrências

Após isso, basta fazer uma verificação de estado atual para cada um dos estados, seguindo o padrão de verificar o estado atual, em que se verifica o estado atual igual a um dos estados de q, no caso do exemplo da figura 18 ocorre o uso do estado “q0”, em que retorna a função de si mesmo, utilizando como argumentos a cadeia sem o primeiro elemento, o novo estado que “q0” aponta, a lista de ocorrências, a variável pos, e o valor de index + 1. No caso do primeiro elemento da cadeia ser igual a “c” o novo estado será “q1”, mas caso seja diferente de “c” o novo estado será “q11”.

```
    if estado_atual == "q0":
        if cadeia[0] == "c":
            return automato_computador(cadeia[1:], "q1", ocorrencias, pos, i + 1)
        return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 18. Seção do código, ainda incompleto, em que verifica se o estado atual é q0

Todavia, o código ocorre uma diferença no estado atual igual a “q0” e “q10”, para poder guardar o início da ocorrência da palavra “computador”, e para guardar esse valor na lista de ocorrências, adicionando uma linha de código a mais na verificação de estado atual igual a

“q0”, alterando a variável de pós para o valor da variável de i, sendo possível observar a seção do código completo que verifica se o estado atual é q0 na figura 19.

```
if estado_atual == "q0":
    pos = i
    if cadeia[0] == "c":
        return automato_computador(cadeia[1:], "q1", ocorrencias, pos, i + 1)
    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 19. Seção do código, completo, em que verifica se o estado atual é q0

Para o estado atual sendo “q10”, no caso de ocorrer a transição para “q0”, antes de ocorrer a transição, deve ser adicionada o valor “pos” a lista de ocorrências, podendo ser observada a função gerada na figura 20.

```
if estado_atual == "q10":
    if cadeia[0] == " ":
        ocorrencias.append(pos)
        return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)
    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 20. Seção do código, em que verifica se o estado atual é q10

Por fim, para finalizar o código e deixar ele mais limpo, é possível simplificar a função observando um padrão dentro de cada uma da verificação de estado atual, a linha “return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)”, que pode ser removida de cada um dos ifs e posta para no final da função, fora de todos os estados atuais, observando-se a alteração pela figura 21 e figura 22, mostrando antes e depois da alteração, respectivamente, no trecho final do código.

```
if estado_atual == "q9":
    if cadeia[0] == "r":
        return automato_computador(cadeia[1:], "q10", ocorrencias, pos, i + 1)
    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)

if estado_atual == "q10":
    if cadeia[0] == " ":
        ocorrencias.append(pos)
        return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)
    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)

if estado_atual == "q11":
    if cadeia[0] == " ":
        return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)
    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 21. Seção do código antes da mudança

```
    if estado_atual == "q9":
        if cadeia[0] == "r":
            return automato_computador(cadeia[1:], "q10", ocorrencias, pos, i + 1)

    if estado_atual == "q10":
        if cadeia[0] == " ":
            ocorrencias.append(pos)
            return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)

    if estado_atual == "q11":
        if cadeia[0] == " ":
            return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)

    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 22. Seção do código após a mudança

Outra alteração que foi feita para o código evitar repetições, é gerar uma variável no início chamada "novo_estado", definida inicialmente como "q11", e para facilitar o código, todas as chamadas de "return" dentro de um "if" serão trocadas para a mudança do valor de "novo_estado" para o estado desejado, um exemplo de código em antes e depois da mudança, respectivamente, pode ser observado na figura 23 e figura 24.

```
    if estado_atual == "q0":
        pos = i
        if cadeia[0] == "c":
            return automato_computador(cadeia[1:], "q1", ocorrencias, pos, i + 1)
        return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)

    # RESTANTE DO CÓDIGO OMITIDO POR QUESTÕES DE ESPAÇO

    if estado_atual == "q11":
        if cadeia[0] == " ":
            return automato_computador(cadeia[1:], "q0", ocorrencias, pos, i + 1)

    return automato_computador(cadeia[1:], "q11", ocorrencias, pos, i + 1)
```

Figura 23. Seção do código antes a mudança

```
    novo_estado = "q11"
    if estado_atual == "q0":
        pos = i
        if cadeia[0] == "c":
            novo_estado = "q1"

    # RESTANTE DO CÓDIGO OMITIDO POR QUESTÕES DE ESPAÇO

    if estado_atual == "q11":
        if cadeia[0] == " ":
            novo_estado = "q0"

    return automato_computador(cadeia[1:], novo_estado, ocorrencias, pos, i + 1)
```

Figura 24. Seção do código após a mudança

E por fim, para poder apresentar o código em uma figura da maneira mais simplificada, será alterada a forma de escrita dos códigos para não haver ifs aninhados, e caso o if execute uma única função, o código será reescrito em uma única linha, podendo observar um exemplo na figura 25 e figura 26, sendo uma seção do código antes e após a mudança.

```
if estado_atual == "q0":
    pos = i
    if cadeia[0] == "c":
        novo_estado = "q1"

if estado_atual == "q1":
    if cadeia[0] == "o":
        novo_estado = "q2"

if estado_atual == "q2":
    if cadeia[0] == "m":
        novo_estado = "q3"
```

Figura 25. Seção do código antes a mudança

```
if estado_atual == "q0" and cadeia[0] == "c":
    pos = i
    novo_estado = "q1"

if estado_atual == "q1" and cadeia[0] == "o": novo_estado = "q2"

if estado_atual == "q2" and cadeia[0] == "m": novo_estado = "q3"
```

Figura 26. Seção do código após a mudança

Com as alterações, o código final foi desenvolvido, como é possível observar na figura 27, sendo ainda necessário aplicar testes experimentais para provar a funcionalidade do programa desenvolvido.

```
def automato_computador(cadeia, estado_atual="q0", ocorrencias=[], pos=0, i=0):
    if cadeia == "":
        if estado_atual == "q10":
            ocorrencias.append(pos)
        return ocorrencias

    novo_estado = "q11"
    if estado_atual == "q0" and cadeia[0] == "c":
        pos = i
        novo_estado = "q1"

    if estado_atual == "q1" and cadeia[0] == "o": novo_estado = "q2"

    if estado_atual == "q2" and cadeia[0] == "m": novo_estado = "q3"

    if estado_atual == "q3" and cadeia[0] == "p": novo_estado = "q4"
    if estado_atual == "q4" and cadeia[0] == "u": novo_estado = "q5"
    if estado_atual == "q5" and cadeia[0] == "t": novo_estado = "q6"
    if estado_atual == "q6" and cadeia[0] == "a": novo_estado = "q7"
    if estado_atual == "q7" and cadeia[0] == "d": novo_estado = "q8"
    if estado_atual == "q8" and cadeia[0] == "o": novo_estado = "q9"
    if estado_atual == "q9" and cadeia[0] == "r": novo_estado = "q10"
    if estado_atual == "q10" and cadeia[0] == " ":
        ocorrencias.append(pos)
        novo_estado = "q0"
    if estado_atual == "q11" and cadeia[0] == " ": novo_estado = "q0"

    return automato_computador(cadeia[1:], novo_estado, ocorrencias, pos, i + 1)
```

Figura 27. Código para achar o casamento exato da palavra computador em um texto

5. Transdutor finito que dado uma sequência de moedas fornece ou não uma lata de refrigerante.

[Texto Importante Aqui]

Nessa questão, o objetivo era implementar um transdutor finito (máquina de Moore ou Mealy) que, dada uma determinada sequência de moedas de valor 25 centavos, 50 centavos e de 1 real, que ao longo do texto e do código vai ser representado como 100, forneceria uma lata de refrigerante caso a soma dos valores dessas moedas fosse maior ou igual a 100, se a soma dos valores dessas moedas fosse menor que 100, a máquina não forneceria a lata de refrigerante. A máquina também teria que imprimir a saída que é gerada toda vez que fosse colocado uma moeda, imprimindo a saída 1 se a soma de moedas na máquina após a colocação da moeda for maior ou igual a 100 e imprimindo a saída 0 se a soma das moedas na máquina após a colocação da moeda for menor do que 100. Com isso em vista, o transdutor da figura 28 representa a linguagem pedida, podendo finalizar em qualquer um dos estados existentes.

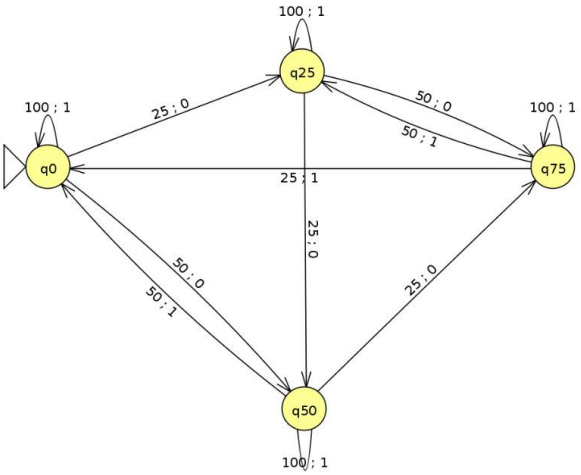

Figura 28. Transdutor para a máquina de refrigerante pedida na questão

Para implementar esse transdutor finito usamos a linguagem de programação python e o transdutor máquina de Mealy para resolver a questão teoricamente, para implementar a máquina criamos uma função chamada "maquina_refri" que recebe como parâmetro uma lista com os valores das moedas. Nessa função declaramos como parâmetros a cadeia a ser analisada pelo transdutor, o estado atual do transdutor, com valor default igual a “q0”, e a lista com os valores de retorno do transdutor, sendo chamada de "saída" e tendo valor padrão igual a uma lista vazia.

Tendo os parâmetros da função definidos, é possível gerar o início do código, fazendo uma verificação para saber se a cadeia está vazia, e se ela estiver a função deve retornar a lista de saída do transdutor, obtendo o código da figura 29, que apresenta a primeira parte da função que utilizará da recursão para servir como transdutor.

```
def automato_maquina_refri(cadeia, estado_atual = "q0", saida = []):
    if cadeia == "": return saida
```

Figura 29. Parte inicial do código do transdutor finito da terceira questão

Após gerar o início da figura 29, é possível continuar o código fazendo uma verificação para cada estado, e dentro de cada verificação fazer outra verificação para saber se o primeiro elemento da cadeia é uma das transições desejadas, “25”, “50” ou “100”, e dentro de cada uma dessas verificações fazer a mudança da variável “estado_atual” para o estado apontado no transdutor. Por exemplo, a transição de “q0” com o valor de 25 centavos é para o estado “q25”, portanto o estado atual deve mudar para “q25”. Ao mesmo tempo, deve ocorrer a adição do valor passado pela transição do transdutor, a variável “saida”. Como ocorre, por exemplo, na transição de “q0” para “q25”, adicionando-se a lista o valor de 0, ou no caso de “q0” para “q0”, em que ocorre na verificação da cadeia como 100 centavos, em que se adiciona a lista o valor de 1. Resultando no código observado na figura 30, utilizando dos preceitos falados anteriormente.

```
if estado_atual == "q0":
    if cadeia[0] == 25:
        estado_atual = "q25"
        saida.append(0)
    if cadeia[0] == 50:
        estado_atual = "q50"
        saida.append(0)
    if cadeia[0] == 100:
        estado_atual = "q0"
        saida.append(1)
```

Figura 30. Seção do código que verifica se o estado atual é “q0” e faz as devidas ações

Com o código para o estado atual “q0” feito, é possível gerar o código para os outros estados, de uma forma parecida, somente alterando o valor trocado do estado atual, para o novo estado tendo como base o primeiro item da cadeia analisada, e o valor adicionado a lista de saída, que será 0 ou 1, tendo como base a transição ocorrida do estado atual para o novo.

E no final de todas as verificações, é necessário fazer a recursão, utilizando de um return para a própria função, tendo como parâmetros a cadeia sem o primeiro elemento, devido ele já ter sido analisado pelas verificações de estado atual, o segundo elemento sendo estado atual, que foi mudado dentro das verificações de estado atual, e a lista de saída, que já foi alterada anteriormente, como observado pela figura 30. E assim, é possível obter o fim do código, que faz a recursão até o fim da lista, devido o tamanho do código ele não será apresentado em figuras, mas é possível obtê-lo no Apêndice A, por meio da plataforma GitHub.

6. Testes experimentais

Para verificar se as funções verificadas estão funcionando devidamente foram feitos testes experimentais para garantir a eficácia do código, utilizando-se da biblioteca pytest, tomando como base o exemplo da figura 31.

```
def teste_automato_a_1():
    assert automato_a("abbbccca") == True
```

Figura 31. Primeiro teste para o autômato a) da primeira questão

Com isso, foi repetido o exemplo acima alterando o valor de teste da função e o valor esperado para um bom número de testes, para cada uma das funções produzidas importadas para o arquivo “test_sample.py”. Após gerar todas as funções de teste, foi utilizado a execução do comando “pytest” no terminal, que executou o teste do código criado, resultando em sucesso, como é possível observar na figura 32

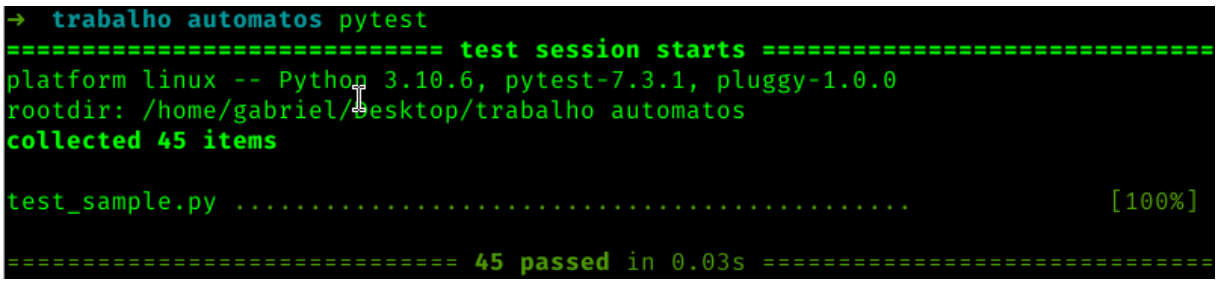

Figura 32. Retorno do terminal com pytest apresentando sucesso

7. Comentários finais

Em suma, é notável o quão poderosa e efetiva é a linguagem python para os problemas aqui representados. Permite uma expressão concisa, clara e indistinta adequada para este assunto de autómatos e transdutores. Também é importante destacar a variedade das questões, que ajudaram a compreender o assunto por diferentes pontos de vista, ao mesmo tempo mostrando a necessidade e eficácia dos autómatos, assim como uma melhor compreensão de como funcionam os reconhecedores e do assunto estudado.

Apêndice A

Código fonte gerado com o desenvolvimento deste relatório:

https://github.com/Braz-Souza/Trabalho-Automatos-e-Transdutores