Implementação de um autômato finito determinístico para validação de senhas

Gabriel Sponda Freitas Bettarello, Matheus Cardoso da Silva, Vithório da Cunha Marques

Universidade Federal do ABC (UFABC)
Av. dos Estados, 5001 - Bairro Bangu - Santo André - CEP: 09280-560

{sponda.gabriel, vithorio.marques, cardoso.matheus}@aluno.ufabc.edu.br

1. Introdução

O reconhecimento de padrões em cadeias de caracteres é uma tarefa fundamental na ciência da computação, especialmente no que diz respeito à validação de dados e análise de linguagens formais. Um dos métodos para realizar tal tarefa envolve o uso de autômatos finitos, estruturas matemáticas capazes de modelar computações ou processos que aceitam cadeias de caracteres seguindo padrões específicos. Este texto aborda o desafio de projetar um autômato finito que possa reconhecer cadeias de caracteres com restrições específicas: cada cadeia deve ter exatamente 8 caracteres, contendo pelo menos uma letra maiúscula, uma letra minúscula, um número e um caractere especial. A complexidade deste problema reside na necessidade de satisfazer múltiplas condições simultaneamente, o que ilustra bem os desafios encontrados na teoria da computação e as limitações práticas de certas abordagens. Através de uma estratégia de simplificação e unificação de autômatos, exploraremos como tornar esse problema tratável, revelando insights sobre a aplicação de autômatos finitos em problemas de reconhecimento de padrões.

2. Descrição do Problema

O desafio de construir um autômato finito (determinístico ou não) para reconhecer cadeias de caracteres específicas ilustra bem a complexidade inerente à teoria da computação e aos limites práticos de certas abordagens. O problema proposto envolve reconhecer cadeias de exatamente 8 caracteres que incluam pelo menos uma letra maiúscula, uma minúscula, um número e um caractere especial. A complexidade surge da necessidade de considerar múltiplos critérios simultâneos e a vasta gama de possíveis combinações.

O assunto em questão foi inspirado pela crescente importância da segurança da informação nos dias atuais, abordando o problema de validar senhas durante uma redefinição ou um cadastro. Senhas as quais precisam respeitar políticas de segurança que podem incluir restrições quanto ao seu tamanho, uso de números e caracteres especiais, entre outras.

2.1 Complexidade de Um Autômato para o Problema Original

Para um autômato finito que diretamente atenda ao problema sem simplificações, a complexidade é exponencial devido à necessidade de manter o controle de múltiplos aspectos de uma cadeia de caracteres em qualquer momento. Precisamos de estados para acompanhar a quantidade de caracteres já lidos e, simultaneamente, verificar se entre esses caracteres encontram-se

representantes das categorias exigidas (maiúsculas, minúsculas, números, caracteres especiais). A combinação desses requisitos resulta em um número exorbitante de estados, tornando a construção e a análise do autômato impraticáveis.

2.2 Simplificando o Problema

Autômato para Contagem de Caracteres: A primeira simplificação considera um autômato para apenas contar a quantidade de caracteres, o qual é relativamente simples. Com 9 estados, cada um representa a quantidade de caracteres lidos, de 0 a 8. Esse autômato pode facilmente verificar se uma cadeia tem exatamente 8 caracteres.

Autômato para Tipos de Caracteres com Função de Conversão: A segunda simplificação aborda a diversidade de caracteres (maiúsculas, minúsculas, números, caracteres especiais) através de uma função de conversão. Essa função mapeia cada caractere à sua categoria, simplificando o alfabeto para apenas quatro símbolos: uma letra maiúscula (A), uma minúscula (a), um número (1) e um caractere especial (#). Um autômato construído sobre este alfabeto simplificado precisa de 16 estados para cobrir todas as combinações possíveis desses quatro tipos de caracteres, assegurando que todos sejam representados na cadeia.

2.3 Unificando os Autômatos via Produto entre dois Autômatos: Para combinar as duas soluções simplificadas em um único autômato capaz de resolver o problema original, utilizamos o método de produto entre autômatos. Esta técnica gera um autômato combinado cujos estados representam pares de estados dos autômatos originais. O estado final desse autômato combinado é determinado pela interseção dos estados finais dos autômatos originais, garantindo que a cadeia atenda simultaneamente aos critérios de tamanho e diversidade de caracteres.

A construção deste autômato combinado, mesmo com a simplificação através da função de conversão, resulta em uma estrutura bastante complexa e extensa. Isso destaca uma limitação prática da abordagem baseada em autômatos para problemas que envolvem múltiplas condições simultâneas em cadeias de caracteres. Em contextos reais, outras abordagens, como expressões regulares ou algoritmos específicos de processamento de strings, podem ser mais eficientes e gerenciáveis.

3. Descrição da Aplicação

3.1 Função de Conversão de Entrada

A aplicação começa com uma função de conversão essencial, categorizar_entrada, projetada para simplificar a análise das cadeias de caracteres. Essa função percorre cada caractere da entrada e o classifica em uma de quatro categorias: letras maiúsculas ('A'), letras minúsculas ('a'), dígitos ('1'), e caracteres especiais ('#'). A escolha dessas categorias está alinhada com os requisitos do problema de incluir ao menos um representante de cada tipo em cadeias de 8 caracteres. O código para esta função é apresentado abaixo:

```
def categorizar_entrada(entrada):
    entrada_categorizada = ""
```

```
    for char in entrada:
        if char.islower():  # Checa se é minúscula
            entrada_categorizada += 'a'
        elif char.isupper():  # Checa se é maiúscula
            entrada_categorizada += 'A'
        elif char.isdigit():  # Checa se é dígito
            entrada_categorizada += '1'
        else:  # Assume que o restante são caracteres
especiais
            entrada_categorizada += '#'
    return entrada_categorizada
```

Esta função serve como uma etapa de pré-processamento, convertendo uma cadeia complexa de caracteres em uma sequência simplificada que reflete a presença das categorias necessárias. Essa abordagem reduz significativamente a complexidade dos autômatos subsequentes, permitindo que se concentrem na lógica de combinação dessas categorias, em vez de lidar com a diversidade completa dos caracteres possíveis.

3.2 Implementação dos Autômatos

Autômato para Contagem de Caracteres: A primeira parte da solução envolve um autômato finito determinístico (DFA) para garantir que a cadeia de entrada contenha exatamente 8 caracteres. Utilizando a biblioteca Automathon, definimos um conjunto de estados Q_m, um alfabeto sigma_m, uma função de transição delta_m, um estado inicial initialState_m, e um conjunto de estados finais F_m. O DFA é construído da seguinte forma:

```
# Autômato M
Q_m = {'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7',
'q8'}
sigma_m = {'a', 'A', '#', '1'}
delta_m = {
    'q0': {'a': 'q1', 'A': 'q1', '#': 'q1', '1': 'q1'},
    'q1': {'a': 'q2', 'A': 'q2', '#': 'q2', '1': 'q2'},
    'q2': {'a': 'q3', 'A': 'q3', '#': 'q3', '1': 'q3'},
    'q3': {'a': 'q4', 'A': 'q4', '#': 'q4', '1': 'q4'},
    'q4': {'a': 'q5', 'A': 'q5', '#': 'q5', '1': 'q5'},
    'q5': {'a': 'q6', 'A': 'q6', '#': 'q6', '1': 'q6'},
    'q6': {'a': 'q7', 'A': 'q7', '#': 'q7', '1': 'q7'},
    'q7': {'a': 'q8', 'A': 'q8', '#': 'q8', '1': 'q8'},
    'q8': {'a': 'q8', 'A': 'q8', '#': 'q8', '1': 'q8'}
}
initialState_m = 'q0'
F_m = {'q8'}

M = DFA(Q_m, sigma_m, delta_m, initialState_m, F_m)
```

Este DFA começa no estado q0 (nenhum caractere lido) e avança um estado a cada caractere lido, independentemente do tipo. Ao atingir o estado q8, a cadeia tem exatamente 8 caracteres. Qualquer caractere adicional não altera o estado, pois o requisito de tamanho já foi satisfeito.

Autômato para Tipos de Caracteres: O segundo autômato concentra-se em verificar a presença de ao menos um caractere de cada categoria requerida. Também implementado com a Automathon, este DFA (N) é definido por Q_n, sigma_n, delta_n, initial_state_n, e F_n, com lógica construída para abranger todas as combinações possíveis de categorias de caracteres ao longo da leitura da cadeia:

```
# Autômato N
Q_n = {f'q{i}' for i in range(16)}
sigma_n = {'a', 'A', '#', '1'}
delta_n = {
    'q0': {'a': 'q1', 'A': 'q2', '#': 'q3', '1': 'q4'},
    'q1': {'a': 'q1', 'A': 'q5', '#': 'q6', '1': 'q7'},
    'q2': {'a': 'q5', 'A': 'q2', '#': 'q8', '1': 'q9'},
    'q3': {'a': 'q6', 'A': 'q8', '#': 'q3', '1': 'q10'},
    'q4': {'a': 'q7', 'A': 'q9', '#': 'q10', '1': 'q4'},
    'q5': {'a': 'q5', 'A': 'q5', '#': 'q11', '1': 'q12'},
    'q6': {'a': 'q6', 'A': 'q11', '#': 'q6', '1': 'q13'},
    'q7': {'a': 'q7', 'A': 'q12', '#': 'q13', '1': 'q7'},
    'q8': {'a': 'q11', 'A': 'q8', '#': 'q8', '1': 'q14'},
    'q9': {'a': 'q12', 'A': 'q9', '#': 'q14', '1': 'q9'},
    'q10': {'a': 'q13', 'A': 'q14', '#': 'q10', '1':
'q10'},
    'q11': {'a': 'q11', 'A': 'q11', '#': 'q11', '1':
'q15'},
    'q12': {'a': 'q12', 'A': 'q12', '#': 'q15', '1':
'q12'},
    'q13': {'a': 'q13', 'A': 'q15', '#': 'q13', '1':
'q13'},
    'q14': {'a': 'q15', 'A': 'q14', '#': 'q14', '1':
'q14'},
    'q15': {'a': 'q15', 'A': 'q15', '#': 'q15', '1': 'q15'}
}
initial_state_n = 'q0'
F_n = {'q15'}

N = DFA(Q_n, sigma_n, delta_n, initial_state_n, F_n)
```

Este DFA começa no estado q0 (nenhuma categoria lida) e transita para estados que refletem as combinações de categorias lidas. O estado final q15 representa a leitura de pelo menos um caractere de cada categoria. A transição entre estados é projetada para acumular categorias sem repetição, garantindo que todas sejam representadas.

Esses autômatos simplificados, focados respectivamente na contagem de caracteres e na diversidade de categorias, são depois combinados via produto entre dois autômatos para formar uma solução unificada capaz de validar cadeias conforme os critérios estabelecidos, oferecendo uma abordagem eficiente e elegante para um problema complexo de reconhecimento de padrões em strings.

3.3 Validando os autômatos:

A biblioteca disponibiliza métodos específicos para avaliar a validade dos autômatos, incluindo métodos denominados is_valid() para dois autômatos distintos: um chamado M, que exige cadeias de 8 caracteres, e outro chamado N, focado no teste de caracteres específicos. A utilização desses métodos é simples e direta, M.is_valid() e N.is_valid(). Ambos os comandos retornam True, indicando que os autômatos foram construídos corretamente e estão prontos para serem testados com cadeias de caracteres. Para validar efetivamente esses autômatos, é essencial testar várias cadeias de caracteres, utilizando-se, para isso, a função de conversão adequada. Esses testes abrangem todas as possíveis situações de erro e acerto para cada autômato, e os resultados comprovam a eficácia dos mesmos. Vejamos alguns exemplos desses testes:

O autômato M deve rejeitar qualquer cadeia com menos de 8 caracteres, independentemente dos símbolos utilizados. Testes com as seguintes cadeias confirmam esse comportamento:

```
M.accept("1111")       # Retorna
False
M.accept("1aA#")       # Retorna
False
M.accept("11AAaa#")    # Retorna
False
```

Por outro lado, cadeias com 8 ou mais caracteres são aceitas, como demonstrado a seguir:

```
M.accept("11111111")   # Retorna True
M.accept("aaaaaaaa")   # Retorna True
M.accept("111AAAaaa###")   # Retorna
True
```

Esses resultados validam o autômato M quanto à sua capacidade de processar corretamente cadeias de diferentes comprimentos.

Para o autômato N, também realizamos uma série de testes cobrindo diversas possibilidades, conforme ilustrado abaixo:

```
N.accept("1111")       # Retorna
False
N.accept("1#AAAAA")    # Retorna
False
N.accept("#111#aaa")   # Retorna
False
```

Estes testes revelam que certas cadeias são rejeitadas conforme esperado. No entanto, cadeias com uma composição específica de caracteres são aceitas:

```
N.accept("1aA#")       # Retorna
True
N.accept("1111a#1A#")  # Retorna
True
```

É importante destacar que, nos testes, limitamos os caracteres às letras 'a', 'A', o número '1' e o símbolo '#', devido à função de conversão empregada. A decisão de não utilizar essa função nos testes foi tomada por questões práticas, mas ainda assim, os resultados obtidos garantem a validade dos autômatos, ou seja, ambos os autômatos M e N funcionam conforme esperado.

Após a validação, podemos visualizar os autômatos utilizando o método view, conforme exemplificado abaixo:

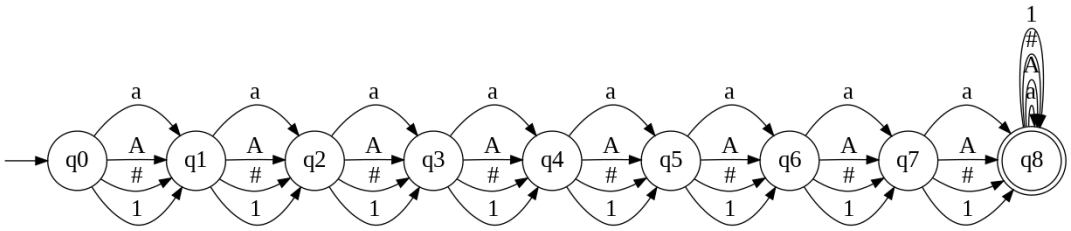

Imagem do Autômato M

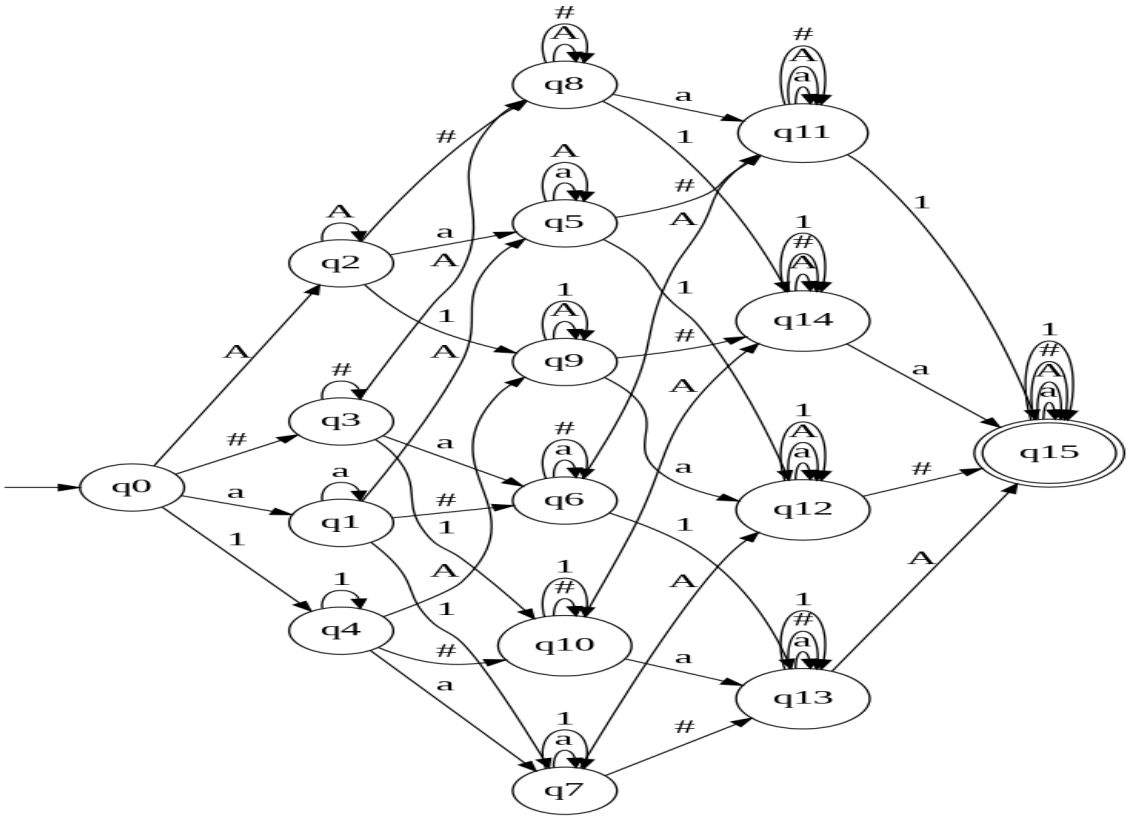

Imagem do Autômato N

Essas visualizações fornecem uma representação gráfica clara da estrutura e do funcionamento dos autômatos, complementando os testes realizados e confirmando a eficácia dos mesmos.

3.4 Implementando o Produto entre Autômatos

O produto entre dois autômatos finitos determinísticos (DFA) M e N, conforme definido na teoria dos autômatos, é uma técnica que permite a criação de um novo DFA que aceita a intersecção das linguagens aceitas por M e N. Esse processo é essencial para combinar critérios distintos de validação, como o comprimento específico de uma cadeia e a presença obrigatória de diferentes categorias de caracteres dentro dessa cadeia.

Segundo Sipser, o produto de dois DFAs é construído definindo-se um conjunto de estados que é o produto cartesiano dos conjuntos de estados dos DFAs originais. Para cada par de estados, um do DFA M e outro do DFA N, cria-se um novo estado no DFA resultante. A função de transição do novo DFA determina suas transições com base nas transições correspondentes dos DFAs originais para cada símbolo do alfabeto.

A função automata_product_dfa(M, N) implementa essa ideia, combinando os autômatos M e N em um novo DFA conforme detalhado a seguir.

Conjunto de Estados (Q_new): É formado pelo produto cartesiano dos conjuntos de estados de M e N. Cada par de estados é representado como uma string formatada, facilitando a identificação dos estados originais no novo DFA.

Alfabeto (sigma_new): Como M e N estão avaliando a mesma cadeia de entrada, o alfabeto do novo DFA é a união dos alfabetos de M e N. No contexto deste problema, M e N compartilham o mesmo alfabeto, então essa etapa simplesmente confirma o alfabeto a ser utilizado.

Função de Transição (delta_new): Define como o novo DFA deve transitar entre seus estados com base nas transições de M e N. Para cada par de estados no novo DFA e cada símbolo do alfabeto, a função de transição determina o próximo estado com base nas funções de transição de M e N.

Estado Inicial (initialState_new): O estado inicial do novo DFA é formado pela combinação dos estados iniciais de M e N, representando o ponto de partida antes de qualquer entrada ser processada.

Conjunto de Estados Finais (F_new): É derivado dos conjuntos de estados finais de M e N. Um estado no novo DFA é final se, e somente se, ambos os estados correspondentes em M e N são finais. Isso garante que o novo DFA só aceita uma cadeia se ambas as condições representadas por M e N forem satisfeitas.

```
def automata_product_dfa(M, N):
    # Novo conjunto de estados: produto cartesiano dos conjuntos de estados,
    # convertendo cada par de estados em uma string formatada
```

```
    Q_new = {f"{q1},{q2}" for q1 in M.Q for q2 in N.Q}

    # Novo alfabeto: união dos alfabetos (que, neste caso, são iguais)
    sigma_new = M.sigma.union(N.sigma)

    # Nova função de transição
    delta_new = {}
    for q1 in M.Q:
        for q2 in N.Q:
            q_new = f"{q1},{q2}"
            delta_new[q_new] = {}
            for a in sigma_new:
                delta_new[q_new][a] = f"{M.delta[q1][a]},{N.delta[q2][a]}"

    # Novo estado inicial
    initialState_new = f"{M.initialState},{N.initialState}"

    # Novo conjunto de estados finais
    F_new = {f"{q1},{q2}" for q1 in M.F for q2 in N.F}

    # Cria o novo DFA
    new_dfa = DFA(Q_new, sigma_new, delta_new, initialState_new, F_new)
    return new_dfa
```

O novo DFA, criado pela função automata_product_dfa, representa uma combinação das restrições impostas por M e N, aceitando apenas cadeias que são validadas por ambos os autômatos. Essa abordagem oferece uma solução elegante e eficiente para problemas que exigem a verificação de múltiplos critérios simultaneamente.
Validamos a operação do produto de dois autômatos finitos determinísticos (DFA) M e N utilizando a função automata_product_dfa(M, N). Após construir o DFA combinado O, usamos O.is_valid() para verificar a validade da estrutura resultante, garantindo que ela segue todas as propriedades necessárias de um DFA. Em seguida, testamos se O aceita a string "1#aA1111" através do método O.accept("1#aA1111"), o qual retorna True se a string for aceita pelo autômato ou False caso contrário. Este processo nos permite avaliar a interação entre M e N e entender melhor como suas combinações afetam a aceitação de strings específicas. O resultado final foi um autômato com 144 estados que pode ser visto na imagem da página seguinte.

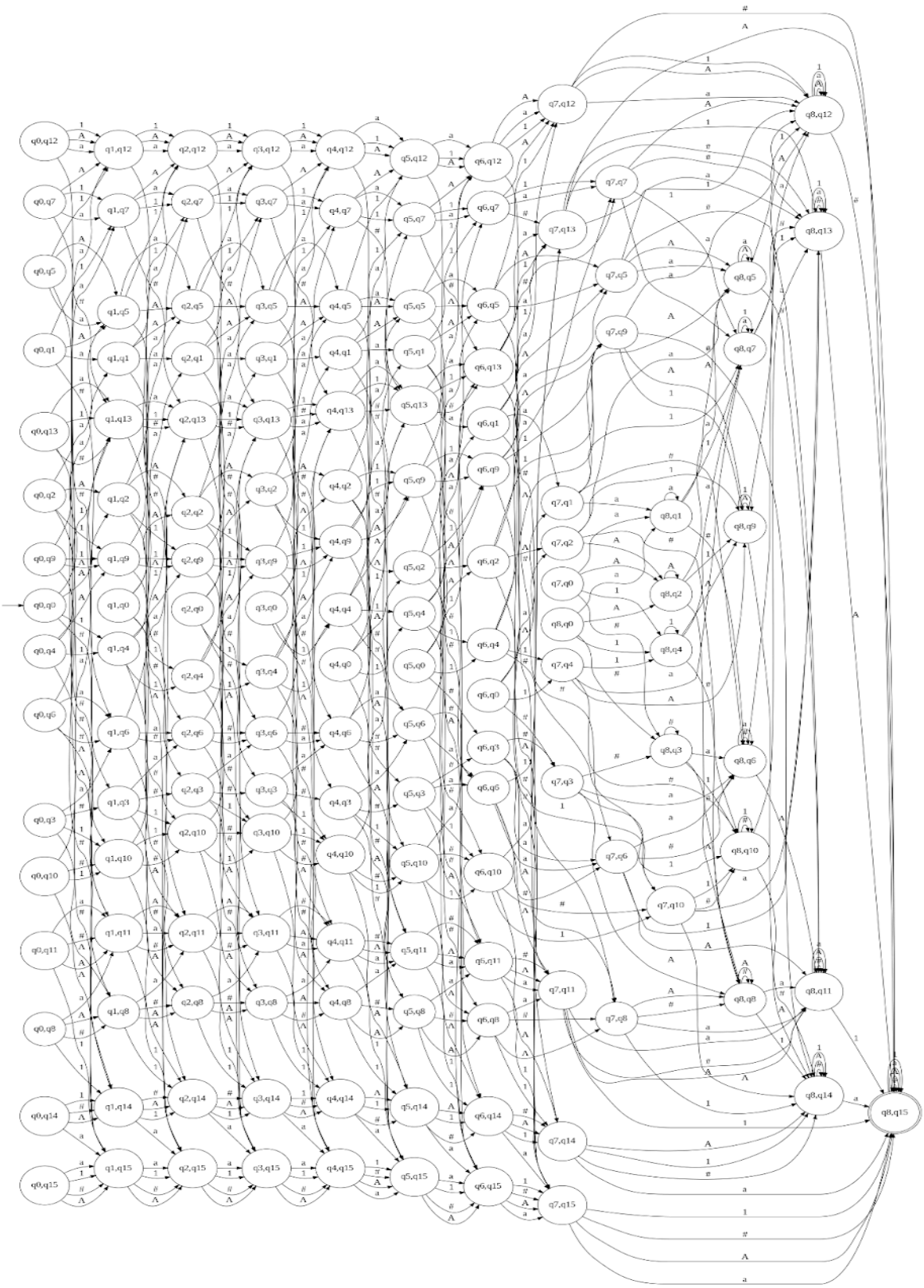

Imagem do autômato O resultando do produto entre os autômatos M e N

3.5. Minimização do Autômato Resultante

A minimização do autômato resultante foi feita utilizando a biblioteca automathon. Esta biblioteca possui uma função de minimização para autômatos finitos não determinísticos, por isso foi necessário utilizar a função “get_nfa()” para converter o autômato determinístico. Em seguida foi aplicada a função “minimize()” para fazer de fato a minimização do autômato e depois a função “get_dfa()” para converter o autômato finito não determinístico já minimizado novamente para um autômato finito determinístico. Após essas operações foi obtido um autômato finito determinístico com 104 estados, constatando que a minimização foi capaz de diminuir o autômato anterior em 40 estados. Devido à impraticidade de se escrever a tabela de transições em consequência do grande número de estados, o autômato foi apresentado apenas com sua representação na imagem. Segue imagem do autômato minimizado abaixo:

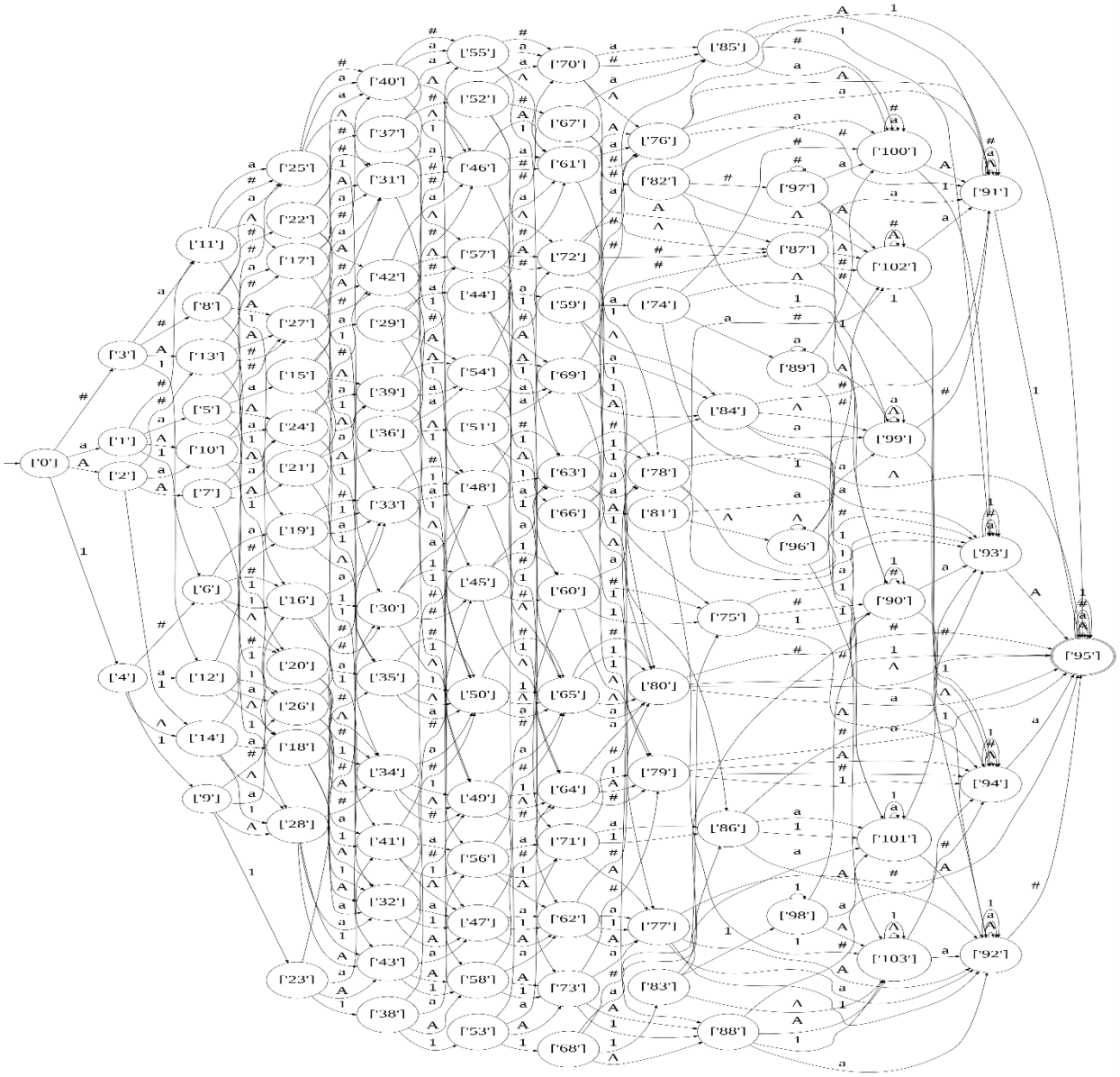

Imagem do autômato O_minimized

4. Considerações Finais

A implementação do produto entre dois autômatos demonstrou ser eficaz, permitindo a criação de um autômato combinado que atende a critérios específicos de validação. Apesar da complexidade do autômato resultante, o objetivo de reconhecer cadeias com características particulares foi alcançado. Este sucesso aponta para a utilidade da técnica de produto e minimização de autômatos, gerando um autômato que consegue modelar e resolver um problema do mundo real (validação de senhas).

5. Link para acesso ao projeto

https://colab.research.google.com/drive/1udxTrxo_ClfQpDa6W6CcrRoJFh94X6Jf?usp=sharing

https://github.com/gabrielsponda/linguagens-formais-e-automatos

6. Referências

[SIPSER 2005] SIPSER, M. - “Introdução à Teoria da Computação”, Editora Cengage, Tradução da 2ª edição norte-americana, 2005.