

TRƯỜNG cntt & tt

TIN HỌC LÝ THUYẾT

vũ thái hà - B2016960

Trần Gia Huy - B2016968

phạm quốc sơn – b1908411

nguyễn khánh trân – b1908427

sinh viên thực hiện:

Đề tài

CHUYỂN ĐỔI NFAℇ SANG DFA

Giảng viên: Trần nguyễn dương chi

# 

MỤC LỤC

[I) MÔ TẢ BÀI TOÁN 1](#_Toc17159)

[A. SƠ LƯỢC VỀ NFAℇ 1](#_Toc26289)

[B. SƠ LƯỢC VỀ DFA 1](#_Toc21490)

[C. XÂY DỰNG DFA TỪ NFA ℇ 2](#_Toc19559)

[II) GIẢI THUẬT GIẢI QUYẾT BÀI TOÁN 3](#_Toc1936)

[III) TRÌNH BÀY CẤU TRÚC VÀ CÀI ĐẶT THUẬT TOÁN 3](#_Toc30953)

[A. LỚP NFAℇ VÀ CÁC PHƯƠNG THỨC 3](#_Toc19218)

[B. CHUYỂN ĐỔI TỪ NFAℇ SANG DFA 6](#_Toc30187)

[IV) KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 11](#_Toc5526)

[A. KẾT LUẬN 11](#_Toc2402)

[B. HƯỚNG PHÁT TRIỂN 11](#_Toc23364)

[V) TÀI LIỆU THAM KHẢO 11](#_Toc11449)

# MÔ TẢ BÀI TOÁN

## SƠ LƯỢC VỀ NFAℇ

* **Định nghĩa**: NFAℇ M(Q, Σ, δ, q0, F)
* δ : hàm chuyển ánh xạ từ Q x (Σ  *{*ℇ*}*)→ 2*Q*
* Khái niệm δ(q, a) là tập hợp tất cả các trạng thái psao cho có phép chuyển nhãn atừ q tới p, với a (Σ  *{*ℇ*}*)
* **Hàm chuyển trạng thái mở rộng**: mở rộng δ thành δ\*
  + δ\*: Q xΣ\*→ 2Q
  + δ\*(q, w) = { p | có đường đi từ q tới p theo nhãn w, trên đường đi có thể chứa cạnh nhãn ℇ}
  + Ta có:
  + δ\*(q, ℇ) = ℇ -CLOSURE(q)
  + δ\*(q,a) = ℇ -CLOSURE( δ( δ\*(q, ℇ), a) )
  + δ\*(q, wa) = ℇ -CLOSURE( δ( δ\*(q, w), a) )

Cách khác: δ\*(q, wa) = ℇ -CLOSURE(P) với P = { p | r δ\*(q, w) và p δ(r, a) }

* + δ\*(R, w) = δ\*(q, w)
* **Ngôn ngữ được chấp nhận bởi NFA**:
  + L(M) được chấp nhận bởi NFAℇ M(Q, Σ, δ, q0, F)

L(M) = { w | δ\*(q0, w) chứa ít nhất một trạng thái trong F }

## SƠ LƯỢC VỀ DFA

* **Định nghĩa**: M=(Q, Σ, δ, q0, F)
  + Q: tập hữu hạn các trạng thái (p, q…)
  + Σ: bộ chữ cái nhập (a, b … ; w, x, y …)
  + δ: hàm chuyển, ánh xạ: Q x Σ → Q, δ (q, a) = p
  + q0 Q : trạng thái bắt đầu.
  + F Q : tập các trạng thái kết thúc.
* **Hàm chuyển trạng thái mở rộng**: Mở rộng hàm chuyển δ là một ánh xạ từ Q x Σ\* →Q áp dụng đối với một trạng thái trên chuỗi, δ (q, w) là trạng thái DFA chuyển đến từ trạng thái q trên chuỗi w:
  + δ(q, ℇ) = q
  + δ(q, wa) = δ( δ(q,w), a) với w, a
* **Ngôn ngữ được chấp nhận**:

L(M) = { w | δ( q0, w ) F }

## XÂY DỰNG DFA TỪ NFA ℇ

* DFA thực chất là một **trường hợp đặc biệt** của NFA nhưng
  + Nó **không có sự truyền rỗng** (truyền trên nhãn ℇ)
  + Với mỗi trạng thái q và ký hiệu nhập a, chỉ có **duy nhất một đường truyền** đến một trạng thái khác
* Ta xây dựng DFA như sau:
  + **Mỗi trạng thái** DFA tượng trưng cho một tập trạng thái của NFA mà NFA có thể chuyển đến sau khi đọc một chuỗi ký hiệu nhập gồm: tất cả ℇ có thể xảy ra trước và sau ký hiệu được đọc
  + **Trạng thái bắt đầu** của DFA là ℇ-CLOSURE(q0) với q0 là trạng thái bắt đầu của NFA ℇ
  + Một trạng thái của DFA là **trạng thái kết thúc** nếu nó là tập các trạng thái của NFA chứa ít nhất một ký hiệu kết thúc của NFA
  + Các trạng thái và hàm chuyển của DFA sẽ được trình bày chi tiết bằng giải thuật bên dưới

A picture containing text, watch

Description automatically generated

*MỘT VÍ DỤ VỀ NFA ℇ*

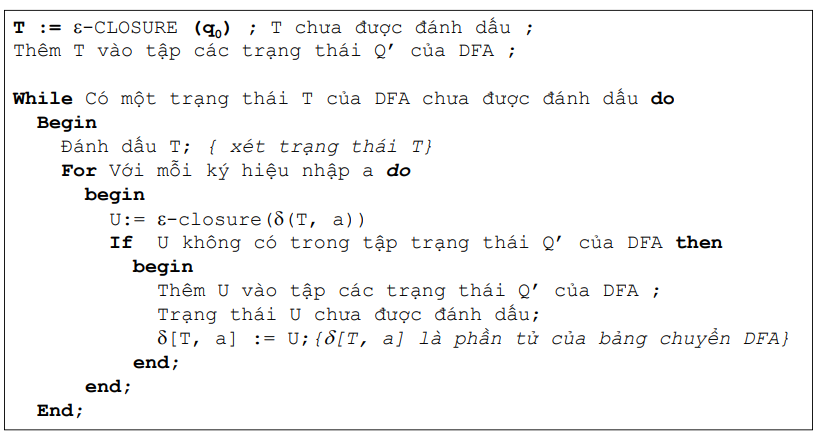
A picture containing text, clock

Description automatically generated

*DFA TƯƠNG ĐƯƠNG CỦA NFA ℇ TRÊN*

# GIẢI THUẬT GIẢI QUYẾT BÀI TOÁN

* Input: Một automata hữu hạn không đơn định NFA epsilon.
* Output: Một automata hữu hạn đơn định DFA nhận dạng cùng ngôn ngữ như NFA epsilon
* Phương pháp: Xây dựng bảng hàm chuyển cho DFA mô phỏng đồng thời tất cả các chuyển dịch của NFA epsilon trên chuỗi nhập cho trước.



# TRÌNH BÀY CẤU TRÚC VÀ CÀI ĐẶT THUẬT TOÁN

## LỚP NFAℇ VÀ CÁC PHƯƠNG THỨC

Trước khi đi vào phân tích các thuật toán được áp dụng, ta sẽ tìm hiểu về các cấu trúc tổng quát cho bài toán. Đầu tiên ta cần đối tượng NFA để lưu trữ.

Đầu tiên là phương thức xây dựng:

Các biến cần lưu ý:

* State: tập hữu hạn các trạng thái
* Alphabet: bộ chứ cái nhập
* Transitions: hàm chuyển
* Start, Finals: trạng thái bắt đầu và kết thúc

|  |
| --- |
| def \_\_init\_\_(self, no\_state, states, no\_alphabet, alphabets, start,  no\_final, finals, no\_transition, transitions):  self.no\_state = no\_state  self.states = states  self.no\_alphabet = no\_alphabet  self.alphabets = alphabets  # Adding epsilon alphabet to the list  # and incrementing the alphabet count  self.alphabets.append('e')  self.no\_alphabet += 1  self.start = start  self.no\_final = no\_final  self.finals = finals  self.no\_transition = no\_transition  self.transitions = transitions  self.graph = Digraph()  # Dictionaries to get index of states or alphabets  self.states\_dict = dict()  for i in range(self.no\_state):  self.states\_dict[self.states[i]] = i  self.alphabets\_dict = dict()  for i in range(self.no\_alphabet):  self.alphabets\_dict[self.alphabets[i]] = i |

Bảng 1.1: Phương thức xây dựng

|  |
| --- |
| # transition table is of the form  # [From State + Alphabet pair] -> [Set of To States]  self.transition\_table = dict()  for i in range(self.no\_state):  for j in range(self.no\_alphabet):  self.transition\_table[str(i) + str(j)] = []  for i in range(self.no\_transition):  self.transition\_table[str(self.states\_dict[self.transitions[i][0]])  + str(self.alphabets\_dict[  self.transitions[i][1]])].append(  self.states\_dict[self.transitions[i][2]]) |

Bảng 1.2: Phương thức xây dựng

Tiếp theo là những phương thức hỗ trợ việc kiểm tra:

* Phương thức nhập dữ liệu từ bàn phím: cho phép nhập các giá trị từ bàn phím

|  |
| --- |
| def fromUser(cls):  no\_state = int(input("Number of States : "))  states = list(input("States : ").split())  no\_alphabet = int(input("Number of Alphabets : "))  alphabets = list(input("Alphabets : ").split())  start = input("Start State : ")  no\_final = int(input("Number of Final States : "))  finals = list(input("Final States : ").split())  no\_transition = int(input("Number of Transitions : "))  transitions = list()  print("Enter Transitions (from alphabet to) (e for epsilon): ")  for i in range(no\_transition):  transitions.append(input("-> ").split())  return cls(no\_state, states, no\_alphabet, alphabets, start,  no\_final, finals, no\_transition, transitions) |

Bảng 2: Phương thức nhập dữ liệu

* Phương thức chuyển sang dữ liệu chuỗi: sử dụng tron việc in để kiểm tra

|  |
| --- |
| # Method to represent quintuple  def \_\_repr\_\_(self):  return "Q : " + str(self.states) + "\nΣ : "\  + str(self.alphabets) + "\nq0 : " + str(self.start) + "\nF : "\  + str(self.finals) \  + "\nδ : \n" + str(self.transition\_table) |

Bảng 3: Phương thức toString

Cuối cùng là những phương thức hỗ trợ giải thuật:

* Phương thức getStateName và isFinalDFA: lấy tên của State để làm giao diện và kiểm tra State có Final hay chưa (DFA).

|  |
| --- |
| def getStateName(self, state\_list):  # Get name from set of states to display in the final DFA diagram  name = ''  for x in state\_list:  name += self.states[x]  return name  def isFinalDFA(self, state\_list):  # Method to check if the set of state is final state in DFA  # by checking if any of the set is a final state in NFA  for x in state\_list:  for y in self.finals:  if (x == self.states\_dict[y]):  return True  return False |

Bảng 4: Phương thức getStateName và isFinalDFA

* Phương thức getEpsilonClosure: nhập vào 1 state trả về epsilon closure của state đó

|  |
| --- |
| def getEpsilonClosure(self, state):  # Method to get Epsilon Closure of a state of NFA  # Make a dictionary to track if the state has been visited before  # And a array that will act as a stack to get the state to visit next  closure = dict()  closure[self.states\_dict[state]] = 0  closure\_stack = [self.states\_dict[state]]  # While stack is not empty the loop will run  while (len(closure\_stack) > 0):  # Get the top of stack that will be evaluated now  cur = closure\_stack.pop(0)  # For the epsilon transition of that state,  # if not present in closure array then add to dict and push to stack  for x in self.transition\_table[str(cur) + str(self.alphabets\_dict['e'])]:  if x not in closure.keys():  closure[x] = 0  closure\_stack.append(x)  closure[cur] = 1  return closure.keys() |

Bảng 5: Phương thức getEpsilonClosure

## CHUYỂN ĐỔI TỪ NFAℇ SANG DFA

Các biến toàn cục cần lưu ý:

* epsilon\_closure: Mảng lưu trữ epsilon closure của từng state
* dfa\_stack: rỗng khi kết thúc giải thuật
* dfa\_states: lưu tất cả các trạng thái của DFA

|  |
| --- |
| # Finding epsilon closure beforehand so to not recalculate each time  epsilon\_closure = dict()  for x in nfa.states:  epsilon\_closure[x] = list(nfa.getEpsilonClosure(x))  # First state of DFA will be epsilon closure of start state of NFA  # This list will act as stack to maintain till when to evaluate the states  dfa\_stack = list()  dfa\_stack.append(epsilon\_closure[nfa.start])  # List to store the states of DFA  dfa\_states = list()  dfa\_states.append(epsilon\_closure[nfa.start]) |

Bảng 6: Biến toàn cục

Giải thuật xây dựng DFA từ NFA epsilon:

- Bước 1: Kiểm tra dfa\_stack có rỗng hay không nếu không làm tiếp ngược lại kết thúc giải thuật.

- Bước 2: Duyệt từng chữ cái trong NFA.

- Bước 3: Lấy trạng thái đã chuyển lưu vào from\_closure

- Bước 4: Kiểm tra from\_closure có rỗng hay không nếu không làm bước 5 ngược lại bước 8

- Bước 5: Lấy epsilon closure của từng giá trị trong from\_closure lưu vào to\_state

- Bước 6: Kiểm tra to\_state đã tồn tại trong DFA hay chưa và thêm vào dfa\_stack

- Bước 7: Kiểm tra final để làm giao diện và trở về bước 2

- Bước 8: Vẽ giao diện trường hợp from\_closure rỗng

|  |
| --- |
| # Loop will run till this stack is not empty  while (len(dfa\_stack) > 0):  # Getting top of the stack for current evaluation  cur\_state = dfa\_stack.pop(0)  # Traversing through all the alphabets for evaluating transitions in DFA  for al in range((nfa.no\_alphabet) - 1):  # Set to see if the epsilon closure of the set is empty or not  from\_closure = set()  for x in cur\_state:  # Performing Union update and adding all the new states in set  from\_closure.update(  set(nfa.transition\_table[str(x) + str(al)]))  # Check if epsilon closure of the new set is not empty  if (len(from\_closure) > 0):  # Set for the To state set in DFA  to\_state = set()  for x in list(from\_closure):  to\_state.update(set(epsilon\_closure[nfa.states[x]])) |

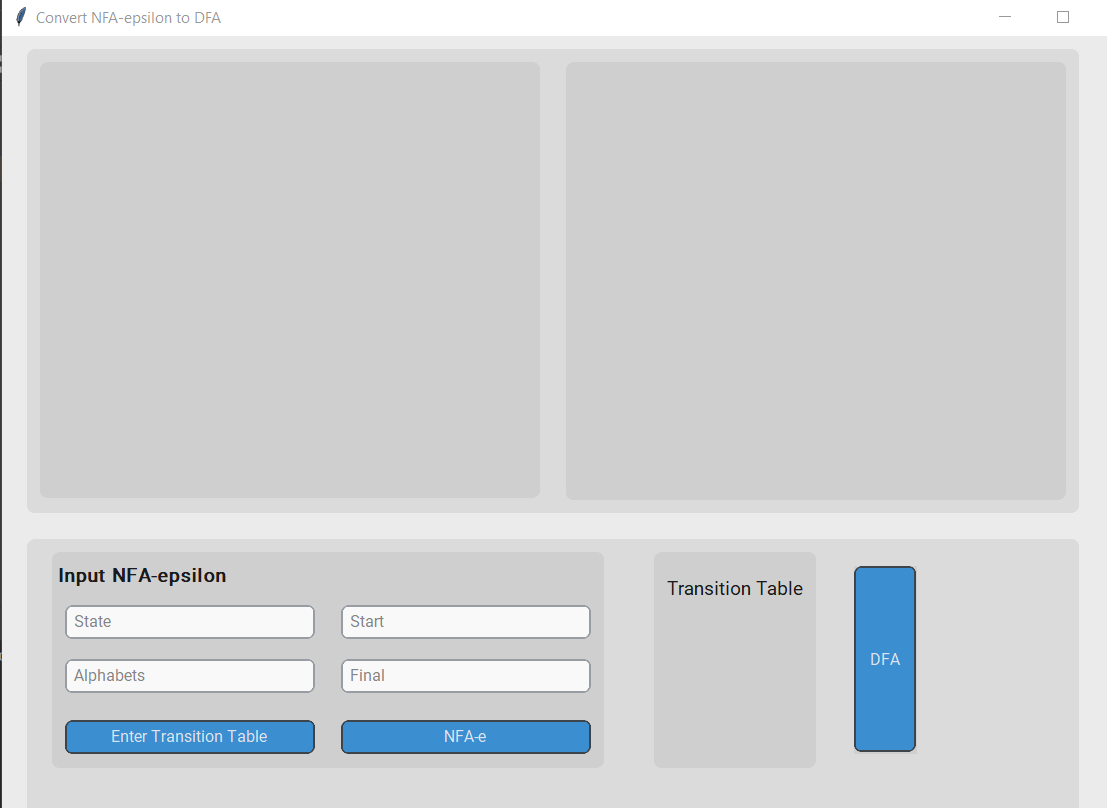
Bảng 7.1: Giải thuật

|  |
| --- |
| # Check if the to state already exists in DFA and if not then add it  if list(to\_state) not in dfa\_states:  dfa\_stack.append(list(to\_state))  dfa\_states.append(list(to\_state))  # Check if this set contains final state of NFA  # to get if this set will be final state in DFA  if (nfa.isFinalDFA(list(to\_state))):  dfa.attr('node', shape='doublecircle')  else:  dfa.attr('node', shape='circle')  dfa.node(nfa.getStateName(list(to\_state)))  # Adding edge between from state and to state  dfa.edge(nfa.getStateName(cur\_state),  nfa.getStateName(list(to\_state)),  label=nfa.alphabets[al])    # Else case for empty epsilon closure  # This is a dead state(ϕ) in DFA  else:  # Check if any dead state was present before this  # if not then make a new dead state ϕ  if (-1) not in dfa\_states:  dfa.attr('node', shape='circle')  dfa.node('ϕ')  # For new dead state, add all transitions to itself,  # so that machine cannot leave the dead state  for alpha in range(nfa.no\_alphabet - 1):  dfa.edge('ϕ', 'ϕ', nfa.alphabets[alpha])  # Adding -1 to list to mark that dead state is present  dfa\_states.append(-1)  # Adding transition to dead state  dfa.edge(nfa.getStateName(cur\_state, ),  'ϕ', label=nfa.alphabets[al]) |

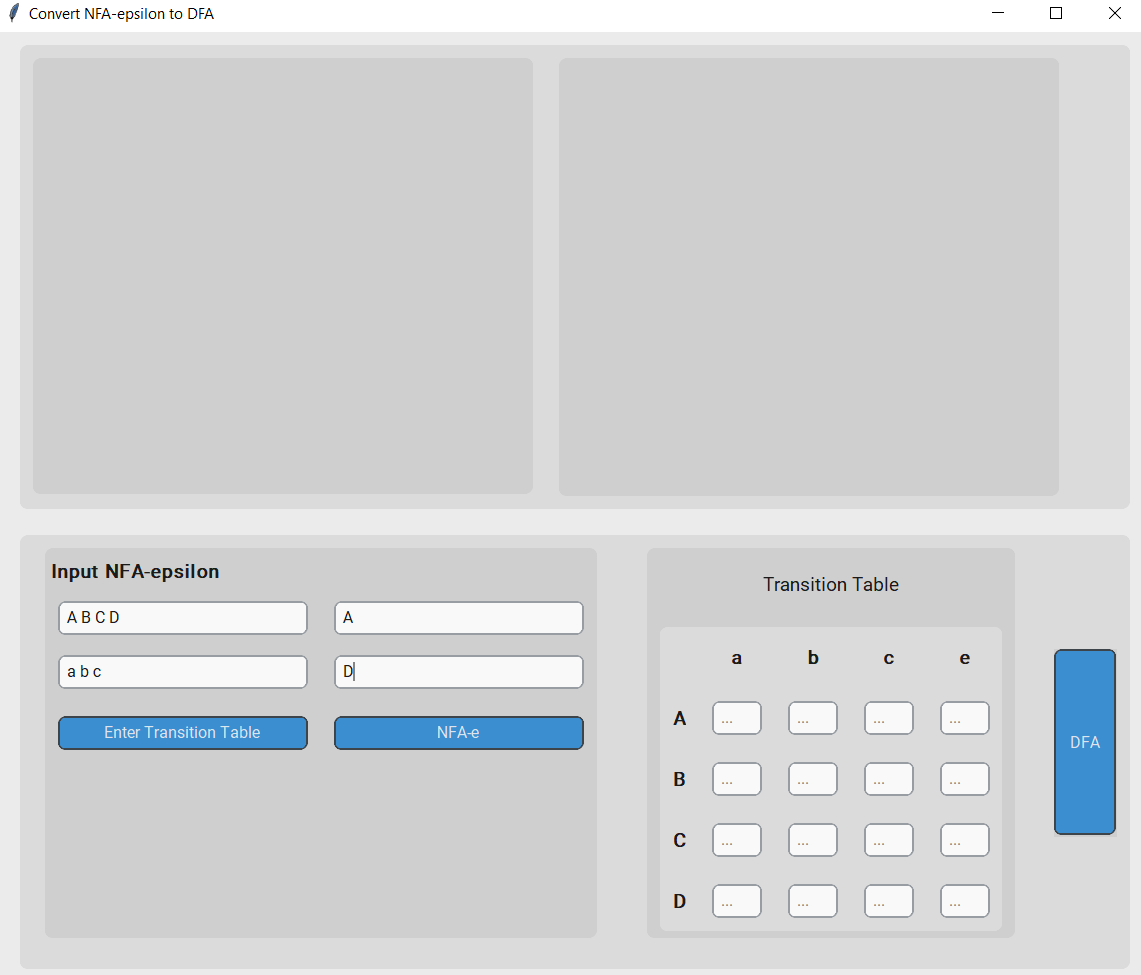
Bảng 7.2: Giải thuật

Một số hình ảnh về kết quả chương trình:

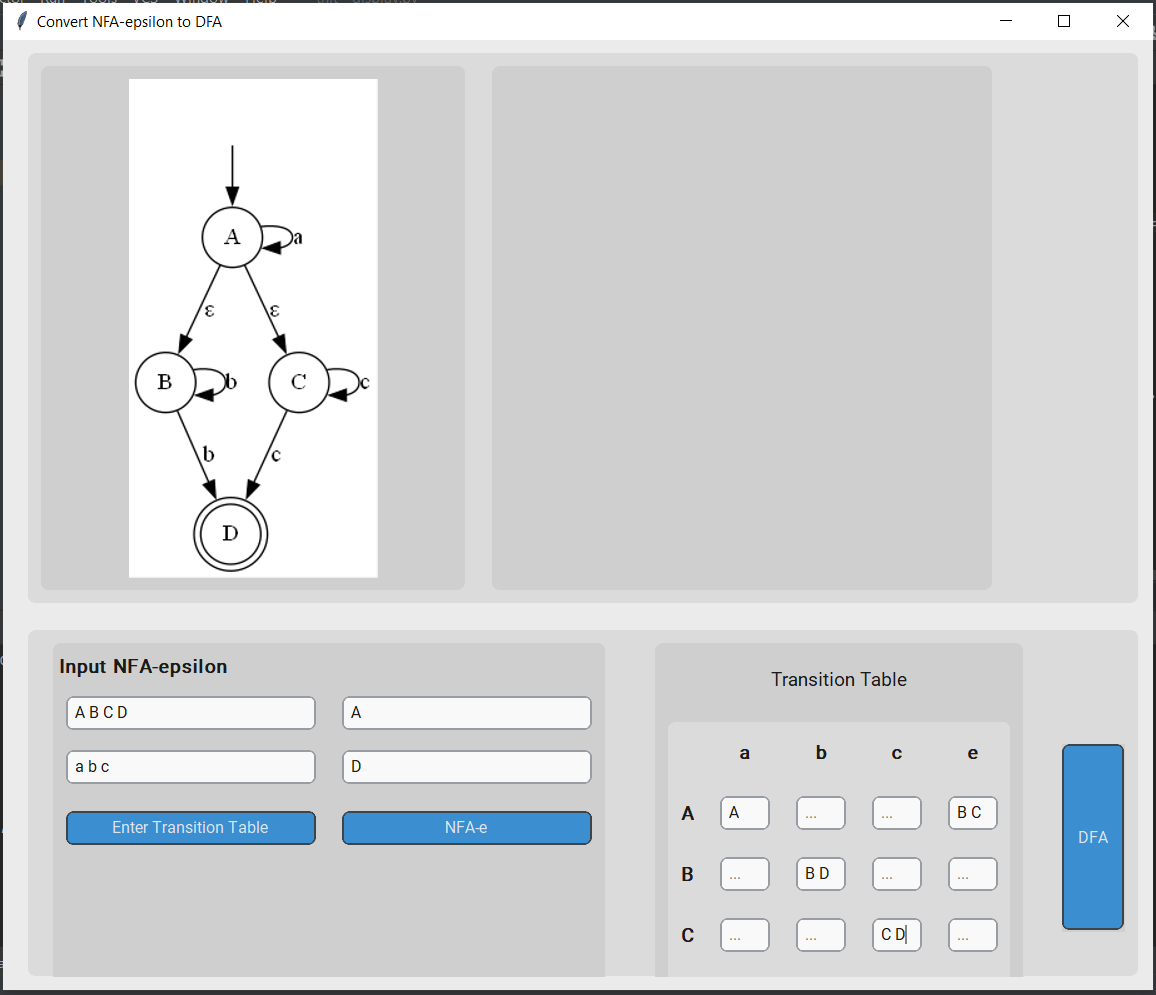
- Giao diên khi bắt đầu.



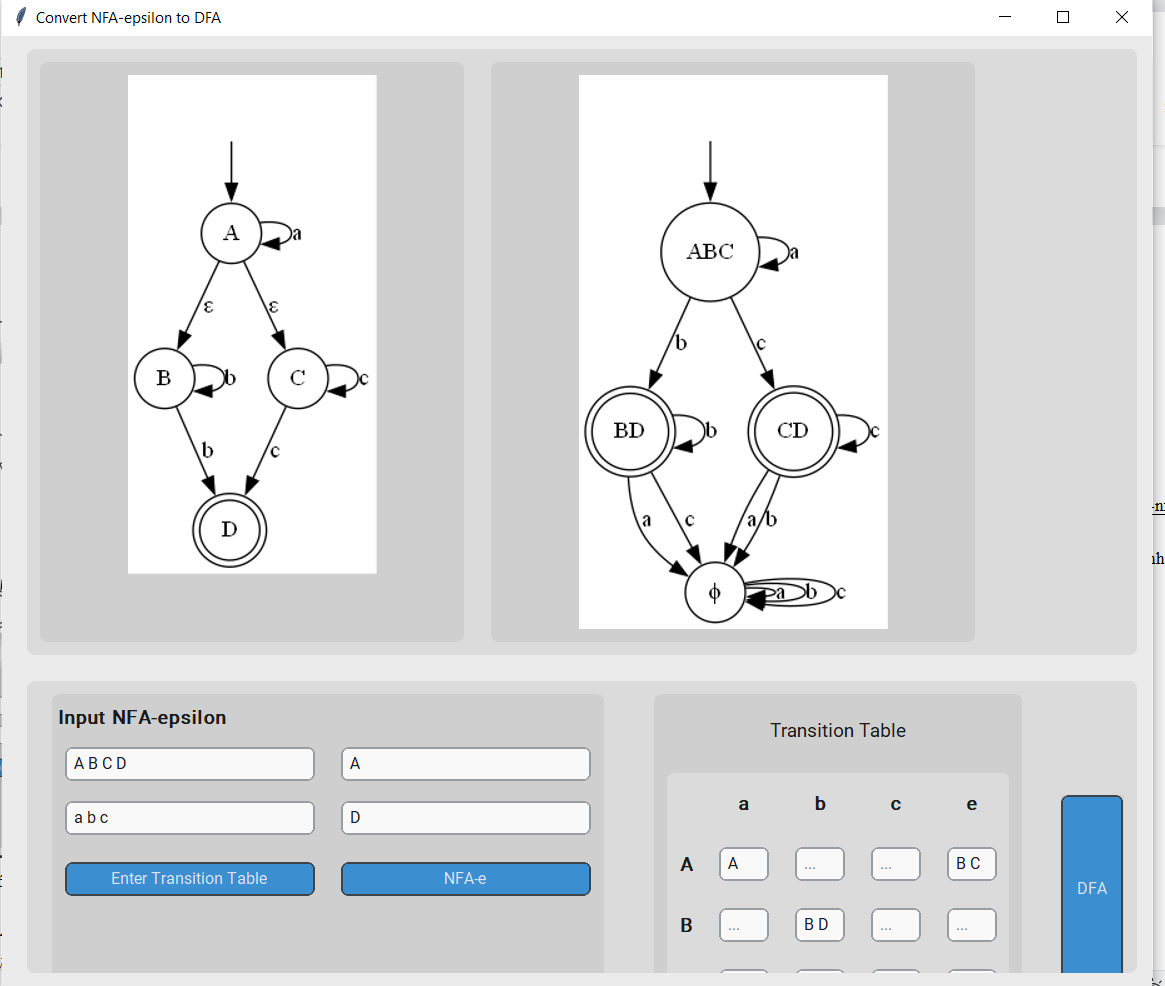
- Nhập vào các Input và Click Enter Transition Table.



- Nhập vào các hàm chuyển và click vào NFA-e: Giao diện sẽ hiển thị 1 hình ảnh sau khi nhập NFA epsilon cho chúng ta kiểm tra.



- Click vào DFA góc dưới cùng chương trình sẽ tự động vẽ DFA



# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## KẾT LUẬN

Qua quá trình nghiên cứu và tìm hiểu về cách giải quyết vấn đề CHUYỂN ĐỔI NFAℇ SANG DFA , nhóm đã đạt được các kết quả sau:

* Nắm vững được các kiến thức lý thuyết về NFAℇ và DFA.
* Hoàn thành được một chương trình giải quyết giải quyết vấn đề trên bằng Python với nhiều cải tiến về giao diện.

Do vấn đề về mặt thời gian, nhóm chưa thể giải quyết được các vấn đề:

* Giao diện chưa được hoàn chỉnh còn nhiều lỗi nhỏ.
* Chưa xử lý được lỗi input
* Còn dựa nhiều vào tài liệu tham khảo.

## HƯỚNG PHÁT TRIỂN

* Cải tiến về mặt giao diện cho hoàn chỉnh và đẹp hơn
* Xây dựng thêm một số chức năng để có thể phát triền.

# TÀI LIỆU THAM KHẢO

1/Website: https://www.geeksforgeeks.org/converting-epsilon-nfa-to-dfa-using-python-and-graphviz/

2/Tài liệu học phần Tin học lý thuyết – Khoa Khoa Học Máy Tính – Trường CNTT & TT – Đại Học Cần Thơ

3/Tài liệu báo cáo học phần Tin học lý thuyết – Năm 2021-2022