|  |  |  |
| --- | --- | --- |
| **AIN SHAMS UNIVERSITY**  **FACULTY OF ENGINEERING**  **International Credit Hours Engineering Programs (i.CHEP)** |  | |
|  | | |
| Automata & Compatibility Project Documentation  **Prepared by Team 2**  **Submitted to**  **Prof. Dr. Gamal A. Ebrahim**  **T.A Eng. Sally E. Shaker** | | |
|  | |  |
|  | |

# Team Namelist:

|  |  |
| --- | --- |
| Name | ID |
| Abdulrahman Ayman Mahmoud | *19P6458* |
| *Ahmed Sameh Mohamed Mourad* | *19P5861* |
| Osama Ayman Mokhtar Amin | *19P1609* |
| *Amr Haithem Abd Almalek Shams* | *19P2352* |

**Table of Contents**

[Team Namelist: ii](#_Toc134546394)

[1. NFA TO DFA: 5](#_Toc134546395)

[1.1 Conversion Steps 5](#_Toc134546396)

[1.2 Output Screenshots 6](#_Toc134546397)

[2. CFG TO PDA: 7](#_Toc134546398)

[2.1 Conversion Steps 7](#_Toc134546399)

[2.2 Output Screenshots 9](#_Toc134546400)

**Table of Figures:**

[Figure 1 shows an example of an NFA. 7](#_Toc134549821)

[Figure 2 shows the converted DFA. 8](#_Toc134549822)

[Figure 3 shows an example of a CFG. 9](#_Toc134549822)

[Figure 4 shows the generated PDA. 10](#_Toc134549822)

1. NFA TO DFA:
   1. Conversion Steps
2. **Inputting The Graph:** The program allows the user to input the graph in the following format: Node, Node, Label. Alternatively, the user can enter the nodes separately and then connect them by the first format. This provides the necessary input to generate the DFA.
3. **Selecting Start and Final States**: The user must select one start state and at least one final state. If the user does identify them, a pop-up error message will show up.
4. **Construct Transition Table**: To convert a Non-Deterministic Finite Automaton (NFA) to a Deterministic Finite Automaton (DFA), a transition table can be constructed using the Subset Construction Algorithm. The algorithm works by constructing a DFA state for each subset of the NFA's states, where each DFA state corresponds to a set of NFA states that can be reached by following a particular input symbol. The algorithm also calculates the transitions between DFA states by following the transitions between the corresponding sets of NFA states. The resulting transition table will have rows corresponding to the DFA states and columns corresponding to the input symbols, with each entry indicating the next DFA state to transition to when a particular input symbol is encountered. The final DFA state is determined by the set of NFA states that includes the initial state and any other states that can be reached from it by following epsilon transitions.
5. **Display DFA**: To obtain the Non-Deterministic Finite Automaton (NFA) from the transition table, the transition table can be used to determine the set of states and transitions for the NFA. Each row in the table corresponds to a state in the NFA, and the entries in that row specify the set of states that can be reached from that state on each input symbol. To create the NFA, a state is created for each row in the table, and the transitions between the states are determined by the entries in the corresponding row. If multiple entries in a row specify transitions on the same input symbol, then the resulting NFA will have non-deterministic transitions. Additionally, if the table includes epsilon transitions, then the NFA will have epsilon transitions between states. Once the states and transitions have been determined, the initial and final states of the NFA can be identified based on the initial and final states of the corresponding DFA.
   1. Output Screenshots

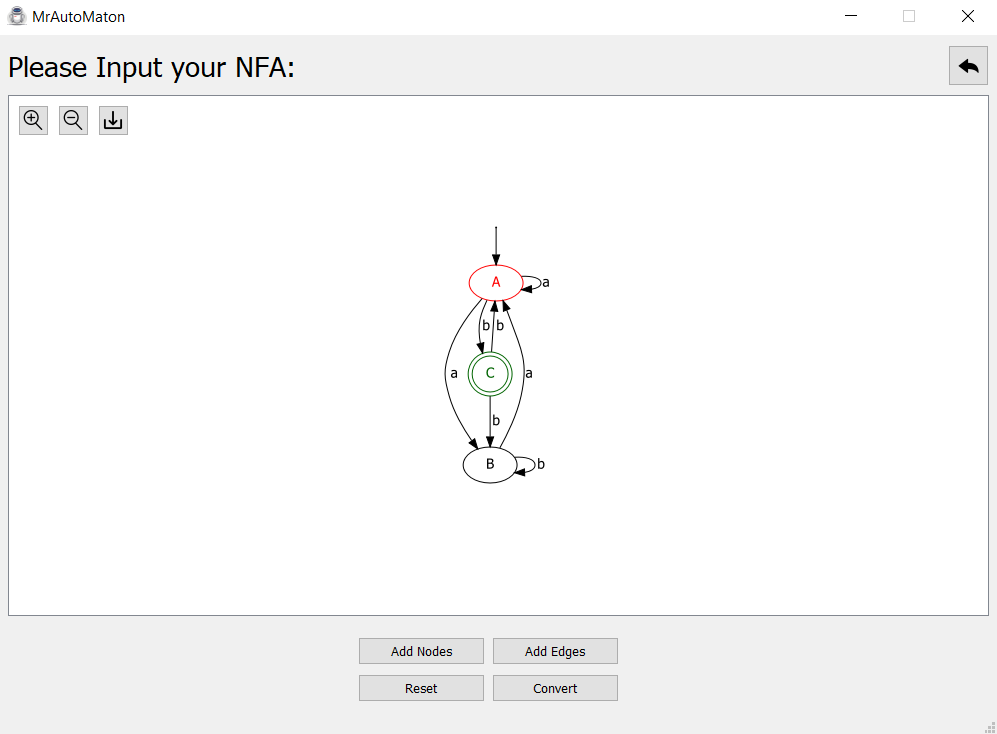


Figure 1 shows an example of an NFA.

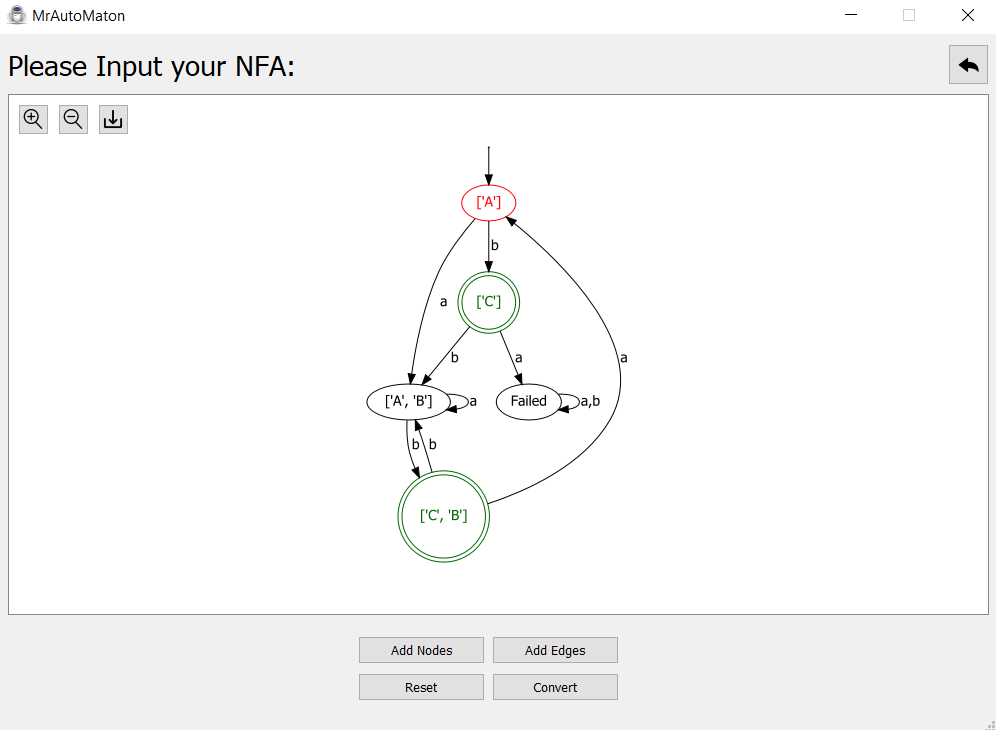
****

Figure 2 shows the converted DFA.

# 2. CFG TO PDA:

## Conversion Steps

1. **Inputting CFG Rules:** The program allows the user to input the context-free grammar rules in the following format: {S -> Aa, S -> c, A -> bc}, etc. This provides the necessary input to generate a pushdown automaton.
2. **Extracting The Start State**: The user has the option to input the starting state or leave it empty. If no starting state is specified, the program automatically selects the first encountered non-terminal in the input CFG rules as the starting state. This ensures that the pushdown automaton is generated correctly based on the provided rules.
3. **Extracting the Terminals & Non-terminals**: The program iterates through the input CFG rules and extracts the terminals, storing them in a list. It then performs the same operation to extract the non-terminals in the CFG rules.
4. **Adding the first two States**: The program adds two initial states to the pushdown automaton. The first state, qstart, is added as (ε, ε -> $), while the second state, q1, is added as (ε, ε -> S). These initial states serve as starting points for the pushdown automaton to begin processing the input strings.
5. **Adding Variable Productions to qloop:** The program adds all variable productions to the qloop state. It does this by popping the variable from the top of the stack and sequentially adding its corresponding productions. This process continues until all variable productions have been added to the qloop state.
6. **Adding Terminals to qloop:** The program adds all terminals to the qloop state. It does this by reading the terminal input from the user and popping it from the stack.
7. **Adding Accepting State (qaccept):** The program adds an accepting state, qaccept, to the pushdown automaton. It transitions to this newly added state by popping the $ symbol from the top of the stack and leaving the qloop state. This finalizes the process of generating the pushdown automaton from the input CFG rules.

## 2.2 Output Screenshots

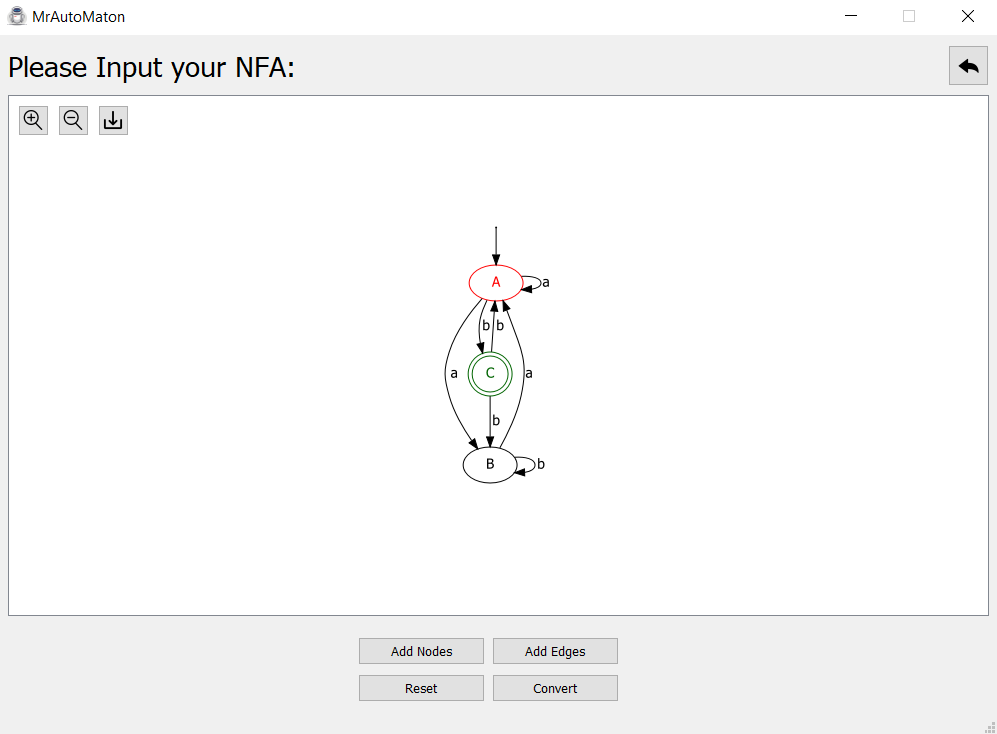


Figure 3 shows an example of a CFG.

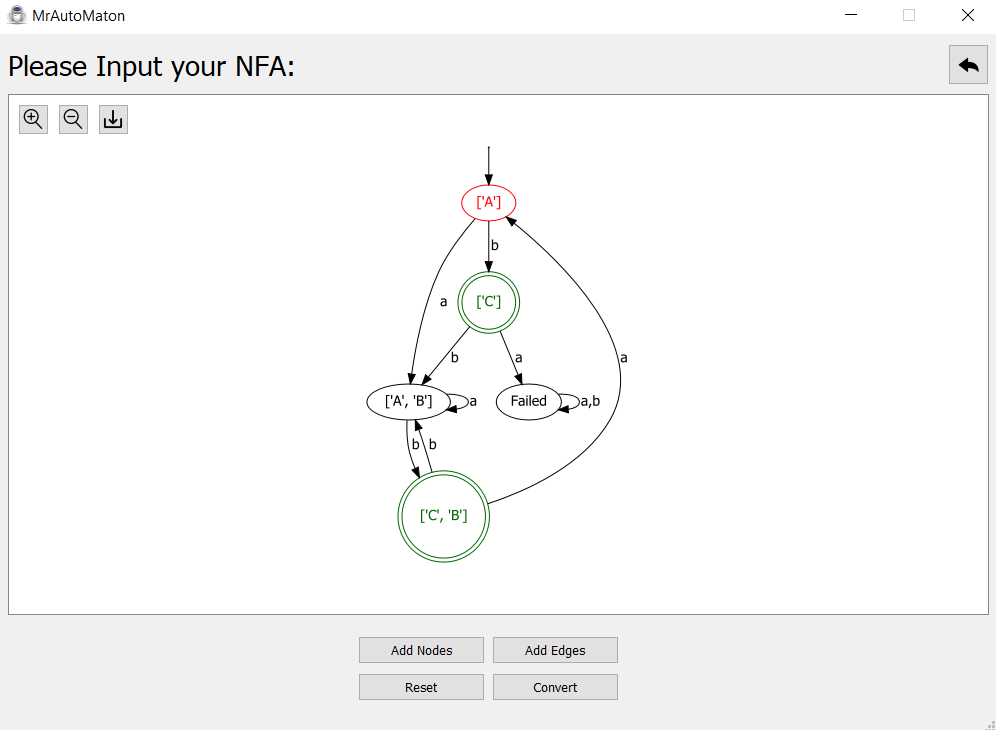
****

Figure 4 shows the Generated PDA.