1. Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with a and end with a

Aim : to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with a and end with a

Algorithm :

1. The DFA transition table is defined as a 2D array. The rows represent the states, and the columns represent the input symbols. In this case, there are 3 states (0, 1, and 2) and 2 input symbols (a and b).
2. The start state and accept state are defined as integers.
3. The input string is read from the user using **scanf**.
4. The current state is initialized to the start state.
5. The program loops through each character in the input string and updates the current state based on the transition table.
6. After processing all the characters, the program checks if the final state is the accept state and prints "Accepted" or "Rejected" accordingly.

Note that this program assumes that the input string only contains characters a and b. If the input string contains other characters, the program will not work correctly.

Programme :

#include <stdio.h>

// Define the DFA transition table

int dfa[3][2] = {{1, 2}, {1, 2}, {1, 2}};

// Define the start state and accept state

int start = 0;

int accept = 1;

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Initialize the current state to the start state

int state = start;

// Loop through each character in the input string

for (int i = 0; input[i] != '\0'; i++) {

// Get the input character

char c = input[i];

// Map the input character to a column in the transition table

int col = (c == 'a') ? 0 : 1;

// Get the next state from the transition table

state = dfa[state][col];

}

// Check if the final state is the accept state

if (state == accept) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

1. Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with 0 and end with 1

AIM : to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with 0 and end with 1

ALGORITHM :

1. Define the DFA transition table.

2. Define the start state and accept state.

3. Get the input string from the user.

4. Initialize the current state to the start state.

5. Loop through each character in the input string:

a. Map the input character to a column in the transition table.

b. Get the next state from the transition table.

6. Check if the final state is the accept state.

7. Print "Accepted" or "Rejected" accordingly.

PROGRAMME :

#include <stdio.h>

// Define the DFA transition table

int dfa[2][2] = {{1, 0}, {1, 1}};

// Define the start state and accept state

int start = 0;

int accept = 1;

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Initialize the current state to the start state

int state = start;

// Loop through each character in the input string

for (int i = 0; input[i] != '\0'; i++) {

// Get the input character

char c = input[i];

// Map the input character to a column in the transition table

int col = (c == '0') ? 0 : 1;

// Get the next state from the transition table

state = dfa[state][col];

}

// Check if the final state is the accept state

if (state == accept) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT : Enter an input string: 010101

Rejected

Enter an input string: 01101

Accepted

1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0A1 A → 0A | 1A | ε

AIM : to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0A1 A → 0A | 1A | ε

ALGORITHM :

1. Define the CFG production rules.

2. Define a function to check if a given string belongs to the language of the CFG.

3. Get the input string from the user.

4. Call the function to check if the input string belongs to the language of the CFG.

5. Print "Accepted" or "Rejected" accordingly.

PROGRAME:

#include <stdio.h>

#include <string.h>

// Define the CFG production rules

void S();

void A();

// Define a function to check if a given string belongs to the language of the CFG

int belongsToLanguage(char input[]) {

// Check if the input string matches the CFG production rules

if (input[0] == '0' && input[strlen(input)-1] == '1') {

int i = 1;

while (input[i] != '1') {

if (input[i] == '0') {

i++;

A(&input[i]);

} else {

return 0;

}

}

return 1;

}

return 0;

}

// Define the CFG production rules

void S() {

printf("S -> 0A1\n");

}

void A(char input[]) {

if (input[0] == '0') {

printf("A -> 0A\n");

A(&input[1]);

} else if (input[0] == '1') {

printf("A -> 1A\n");

A(&input[1]);

} else {

printf("A -> epsilon\n");

}

}

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Check if the input string belongs to the language of the CFG

if (belongsToLanguage(input)) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT : Enter an input string: 00100101

S -> 0A1

A -> 0A

A -> 0A

A -> epsilon

Accepted

Enter an input string: 001011

S -> 0A1

A -> 0A

A -> 1A

Rejected

4) Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | 1S1 | 0 | 1 | ε

AIM: a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | 1S1 | 0 | 1 | ε

ALGORITHM: 1. Define the CFG production rules.

2. Define a function to check if a given string belongs to the language of the CFG.

3. Get the input string from the user.

4. Call the function to check if the input string belongs to the language of the CFG.

5. Print "Accepted" or "Rejected" accordingly.

PROGRAMME : #include <stdio.h>

#include <string.h>

// Define the CFG production rules

void S(char input[]);

// Define a function to check if a given string belongs to the language of the CFG

int belongsToLanguage(char input[]) {

// Check if the input string matches the CFG production rules

if (strlen(input) == 0) {

return 1; // empty string is accepted

} else if (strlen(input) == 1) {

if (input[0] == '0' || input[0] == '1') {

return 1; // single 0 or 1 is accepted

} else {

return 0; // single character that is not 0 or 1 is rejected

}

} else if ((input[0] == '0' && input[strlen(input)-1] == '0') || (input[0] == '1' && input[strlen(input)-1] == '1')) {

S(input); // check if the input string matches the S production rules

return 1;

}

return 0;

}

// Define the CFG production rules

void S(char input[]) {

if (strlen(input) == 2) {

return; // stop condition for recursion

} else {

int i = 1;

while (i < strlen(input) - 1) {

if ((input[0] == '0' && input[strlen(input)-1] == '0') && input[i] == '0') {

printf("S -> 0S0\n");

input[i] = 'S'; // replace '0' with 'S' to reduce the input string

S(input);

i++; // increment i after S() returns

} else if ((input[0] == '1' && input[strlen(input)-1] == '1') && input[i] == '1') {

printf("S -> 1S1\n");

input[i] = 'S'; // replace '1' with 'S' to reduce the input string

S(input);

i++; // increment i after S() returns

} else if (input[i] == '0' || input[i] == '1') {

i++; // skip over '0' or '1'

} else {

return; // stop recursion if input[i] is not '0' or '1'

}

}

}

}

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Check if the input string belongs to the language of the CFG

if (belongsToLanguage(input)) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT :

Enter an input string: 0110

S -> 0S0

S -> 1S1

Accepted

Enter an input string: 01010

S -> 0S0

S -> 1S1

1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | A A → 1A | ε

AIM : C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | A A → 1A | ε

ALGORITHM:

1. Define the CFG production rules.

2. Define a function to check if a given string belongs to the language of the CFG.

3. Get the input string from the user.

4. Call the function to check if the input string belongs to the language of the CFG.

5. Print "Accepted" or "Rejected" accordingly.

PROGRAMME :

#include <stdio.h>

#include <string.h>

// Define the CFG production rules

void S(char input[]);

// Define a function to check if a given string belongs to the language of the CFG

int belongsToLanguage(char input[]) {

// Check if the input string matches the CFG production rules

if (strlen(input) == 0) {

return 1; // empty string is accepted

} else if (input[0] == '0' && input[strlen(input)-1] == '0') {

S(input); // check if the input string matches the S production rules

return 1;

}

return 0;

}

// Define the CFG production rules

void S(char input[]) {

if (strlen(input) == 2) {

return; // stop condition for recursion

} else {

int i = 1;

while (i < strlen(input) - 1) {

if (input[0] == '0' && input[strlen(input)-1] == '0' && input[i] == '0') {

printf("S -> 0S0\n");

input[i] = 'S'; // replace '0' with 'S' to reduce the input string

S(input);

i++; // increment i after S() returns

} else if (input[i] == '1') {

printf("S -> A\n");

input[i] = 'A'; // replace '1' with 'A' to reduce the input string

S(input);

i++; // increment i after S() returns

} else {

return; // stop recursion if input[i] is not '0' or '1'

}

}

}

}

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Check if the input string belongs to the language of the CFG

if (belongsToLanguage(input)) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter an input string: 0110

S -> 0S0

S -> A

S -> 1A1

S -> A

Accepted

Enter an input string: 01010

Rejected

1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S1 | ε

AIM : C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S1 | ε

ALGORITHM :

1. Define the CFG production rules.

2. Define a function to check if a given string belongs to the language of the CFG.

3. Get the input string from the user.

4. Call the function to check if the input string belongs to the language of the CFG.

5. Print "Accepted" or "Rejected" accordingly.

PROGRAMME :

#include <stdio.h>

#include <string.h>

// Define the CFG production rules

void S(char input[]);

// Define a function to check if a given string belongs to the language of the CFG

int belongsToLanguage(char input[]) {

// Check if the input string matches the CFG production rules

if (strlen(input) == 0) {

return 1; // empty string is accepted

} else {

S(input); // check if the input string matches the S production rules

return strcmp(input, "S") == 0;

}

}

// Define the CFG production rules

void S(char input[]) {

if (strlen(input) == 1) {

return; // stop condition for recursion

} else {

int i = 1;

while (i < strlen(input) - 1) {

if (input[0] == '0' && input[strlen(input)-1] == '1' && input[i] == 'S') {

printf("S -> 0S1\n");

input[i-1] = 'S'; // replace '0S1' with 'S' to reduce the input string

memmove(input+i, input+i+2, strlen(input)-i-1); // shift the remaining characters to the left

i = 1; // reset i to 1 to check from the beginning again

} else {

i++; // increment i if input[i] is not '0S1'

}

}

}

}

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Check if the input string belongs to the language of the CFG

if (belongsToLanguage(input)) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter an input string: 0011

S -> 0S1

S -> 0ε1

S -> ε

Accepted

Enter an input string: 0101

Rejected

7) Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → A101A, A → 0A | 1A | ε

AIM: C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → A101A, A → 0A | 1A | ε

ALGORITHM :

1. Define the CFG production rules.

2. Define a function to check if a given string belongs to the language of the CFG.

3. Get the input string from the user.

4. Call the function to check if the input string belongs to the language of the CFG.

5. Print "Accepted" or "Rejected" accordingly.

PROGRAMME :

#include <stdio.h>

#include <string.h>

// Define the CFG production rules

void S(char input[]);

void A(char input[]);

// Define a function to check if a given string belongs to the language of the CFG

int belongsToLanguage(char input[]) {

// Check if the input string matches the CFG production rules

S(input); // check if the input string matches the S production rules

return strcmp(input, "S") == 0;

}

// Define the CFG production rules

void S(char input[]) {

if (strlen(input) < 4 || input[0] != 'A' || input[strlen(input)-1] != 'A') {

return; // stop condition for recursion

} else {

A(input); // check if the input string matches the A production rules

int i = 1;

while (i < strlen(input) - 1) {

if (input[i-1] == 'A' && input[i] == '1' && input[i+1] == '0' && input[i+2] == '1' && input[i+3] == 'A') {

printf("S -> A101A\n");

input[i-1] = 'S'; // replace 'A101A' with 'S' to reduce the input string

memmove(input+i, input+i+4, strlen(input)-i-3); // shift the remaining characters to the left

i = 1; // reset i to 1 to check from the beginning again

A(input); // check if the input string matches the A production rules again

} else {

i++; // increment i if input[i] is not 'A101A'

}

}

}

}

void A(char input[]) {

if (strlen(input) == 1) {

return; // stop condition for recursion

} else {

int i = 0;

while (i < strlen(input)) {

if (input[i] == '0' || input[i] == '1') {

printf("A -> %cA\n", input[i]);

i++; // increment i if input[i] is '0' or '1'

} else {

printf("A -> ε\n");

input[i] = 'A'; // replace 'ε' with 'A' to reduce the input string

memmove(input+i+1, input+i+2, strlen(input)-i-1); // shift the remaining characters to the left

i++; // increment i if input[i] is 'ε'

}

}

}

}

int main() {

// Get the input string from the user

char input[100];

printf("Enter an input string: ");

scanf("%s", input);

// Check if the input string belongs to the language of the CFG

if (belongsToLanguage(input)) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter an input string: 010101

A ->

1. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with b and end with a

AIM : C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with b and end with a

ALGORITHM :

1. Define the NFA using its states, input alphabet, transition function, start state, and accepting states.
2. Read the input string.
3. For each character in the input string, apply the transition function to the current state and the current input character to get a set of possible next states.
4. If there are no possible next states, reject the string.
5. If the last character of the input string has been processed and the current state is an accepting state, accept the string. Otherwise, reject the string.

PROGRAMME :

#include <stdio.h>

#include <string.h>

#define NUM\_STATES 4

#define NUM\_SYMBOLS 2

int transitionTable[NUM\_STATES][NUM\_SYMBOLS][NUM\_STATES] = {

{{0, 1}, {-1, -1}},

{{-1, -1}, {2, -1}},

{{-1, -1}, {3, -1}},

{{-1, -1}, {-1, -1}}

};

int main() {

char inputString[100];

int currentState = 0;

int i, j, k;

printf("Enter input string: ");

scanf("%s", inputString);

for (i = 0; i < strlen(inputString); i++) {

int symbol = inputString[i] - 'a';

int nextStates[NUM\_STATES] = {-1};

for (j = 0; j < NUM\_STATES; j++) {

if (currentState == j || currentState == -1) {

for (k = 0; k < NUM\_STATES; k++) {

int nextState = transitionTable[j][symbol][k];

if (nextState != -1) {

nextStates[nextState] = 1;

}

}

}

}

currentState = -1;

for (j = 0; j < NUM\_STATES; j++) {

if (nextStates[j] != -1) {

currentState = j;

break;

}

}

if (currentState == -1) {

printf("Rejected\n");

return 0;

}

}

if (currentState == NUM\_STATES - 1) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT :

Enter input string: baaba

Accepted

Enter input string: babab

Rejected

1. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with o and end with 1

AIM : C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with o and end with 1

ALGORITHM :

1. Define the NFA using its states, input alphabet, transition function, start state, and accepting states.
2. Read the input string.
3. For each character in the input string, apply the transition function to the current state and the current input character to get a set of possible next states.
4. If there are no possible next states, reject the string.
5. If the last character of the input string has been processed and the current state is an accepting state, accept the string. Otherwise, reject the string.

PROGRAMME:

#include <stdio.h>

#include <string.h>

#define NUM\_STATES 4

#define NUM\_SYMBOLS 2

int transitionTable[NUM\_STATES][NUM\_SYMBOLS][NUM\_STATES] = {

{{-1, 1}, {-1, -1}},

{{2, -1}, {-1, -1}},

{{-1, -1}, {3, -1}},

{{-1, -1}, {-1, -1}}

};

int main() {

char inputString[100];

int currentState = 0;

int i, j, k;

printf("Enter input string: ");

scanf("%s", inputString);

for (i = 0; i < strlen(inputString); i++) {

int symbol = inputString[i] - 'o';

int nextStates[NUM\_STATES] = {-1};

for (j = 0; j < NUM\_STATES; j++) {

if (currentState == j || currentState == -1) {

for (k = 0; k < NUM\_STATES; k++) {

int nextState = transitionTable[j][symbol][k];

if (nextState != -1) {

nextStates[nextState] = 1;

}

}

}

}

currentState = -1;

for (j = 0; j < NUM\_STATES; j++) {

if (nextStates[j] != -1) {

currentState = j;

break;

}

}

if (currentState == -1) {

printf("Rejected\n");

return 0;

}

}

if (currentState == NUM\_STATES - 1) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT :

Enter input string: ooo1

Accepted

Enter input string: oob1

Rejected

1. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.
2. AIM : C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

ALGORITHM :

1. Define the NFA using its states, input alphabet, transition function, start state, and accepting states.
2. For each state in the NFA, initialize its ε-closure to contain itself.
3. For each state in the NFA, apply the ε-closure operation to find all states that can be reached from it by following ε-transitions.
4. Repeat step 3 until no new states are added to the ε-closure of any state.

PROGRAMME :

#include <stdio.h>

#define NUM\_STATES 4

#define NUM\_SYMBOLS 2

int transitionTable[NUM\_STATES][NUM\_SYMBOLS][NUM\_STATES] = {

{{1, -1}, {-1, -1}},

{{2, 3}, {-1, -1}},

{{-1, -1}, {3, -1}},

{{-1, 1}, {-1, -1}}

};

int eclosure[NUM\_STATES][NUM\_STATES] = {0};

void epsilonClosure(int state, int current) {

int i;

if (eclosure[state][current] == 1) {

return;

}

eclosure[state][current] = 1;

for (i = 0; i < NUM\_STATES; i++) {

if (transitionTable[current][0][i] != -1) {

epsilonClosure(state, transitionTable[current][0][i]);

}

}

}

int main() {

int i, j;

for (i = 0; i < NUM\_STATES; i++) {

eclosure[i][i] = 1;

epsilonClosure(i, i);

}

for (i = 0; i < NUM\_STATES; i++) {

printf("ε-closure(%d) = {", i);

for (j = 0; j < NUM\_STATES; j++) {

if (eclosure[i][j] == 1) {

printf("%d,", j);

}

}

printf("}\n");

}

return 0;

}

OUTPUT :

ε-closure(0) = {0,}

ε-closure(1) = {1,2,3,}

ε-closure(2) = {2,3,}

ε-closure(3) = {1,3,}

1. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

AIM : C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

ALGORITHM :

1. Create an empty set e\_closure[] for each state in the NFA
2. For each state q in the NFA, do the following:

a. Add q to the e\_closure[q] set

b. Create an empty stack S and add q to it

c. While S is not empty, do the following:

Return the e\_closure[] set for all states in the NFA

PROGRAMME :

#include <stdio.h>

#include <stdlib.h>

#define MAX\_STATES 10

#define MAX\_TRANSITIONS 20

int nfa[MAX\_STATES][MAX\_TRANSITIONS][MAX\_STATES];

int eclosure[MAX\_STATES][MAX\_STATES];

void dfs(int state, int start\_state, int curr\_state) {

if(eclosure[start\_state][curr\_state] == 1) {

return 0;

}

eclosure[start\_state][curr\_state] = 1;

for(int i = 0; i < MAX\_TRANSITIONS; i++) {

if(nfa[curr\_state][i][0] != -1 && nfa[curr\_state][i][0] != state) {

continue; // Not an ε-transition or already visited this state

}

for(int j = 1; j < MAX\_STATES && nfa[curr\_state][i][j] != -1; j++) {

dfs(state, start\_state, nfa[curr\_state][i][j]);

}

}

}

int main() {

int num\_states, num\_symbols, num\_transitions;

printf("Enter the number of states: ");

scanf("%d", &num\_states);

printf("Enter the number of input symbols: ");

scanf("%d", &num\_symbols);

printf("Enter the number of transitions: ");

scanf("%d", &num\_transitions);

// Initialize NFA transition table

for(int i = 0; i < num\_states; i++) {

for(int j = 0; j < num\_symbols; j++) {

for(int k = 0; k < num\_states; k++) {

nfa[i][j][k] = -1;

}

}

}

// Read NFA transitions

int from\_state, to\_state;

char symbol;

printf("Enter the NFA transitions:\n");

for(int i = 0; i < num\_transitions; i++) {

printf("Transition %d: ", i + 1);

scanf("%d %c %d", &from\_state, &symbol, &to\_state);

int symbol\_index = symbol - 'a';

int j = 0;

while(nfa[from\_state][symbol\_index][j] != -1) {

j++; // Find the next free slot for the transition

}

nfa[from\_state][symbol\_index][j] = to\_state;

}

// Compute ε-closure for all states

for(int i = 0; i < num\_states; i++) {

for(int j = 0; j < num\_states; j++) {

eclosure[i][j] = 0;

}

dfs(i, i, i); // Start DFS from the current state to find ε-closure

}

// Print ε-closure for all states

printf("Epsilon-closure for all states:\n");

for(int i = 0; i < num\_states; i++) {

printf("q%d: ", i);

for(int j = 0; j < num\_states; j++) {

if(eclosure[i][j] == 1) {

1. Design DFA using simulator to accept the input string “a” ,”ac”,and ”bac”.

AIM:

DFA using simulator to accept the input string “a” ,”ac”,and ”bac”.

ALGORITHM:

The DFA has three states: state 0, state 1, and state 2. The initial state is state 0.

The DFA accepts the input strings "a", "ac", and "bac". When the DFA reaches state 2, it accepts the input string.

The for loop reads each character in the input string one by one. If the current state and current input character match a transition in the DFA, the DFA moves to the next state. If the current state and current input character do not match a transition in the DFA, the DFA rejects the input string.

If the DFA reaches the end of the input string and the current state is state 2, the DFA accepts the input string. Otherwise, the DFA rejects the input string.

PROGRAMME :

#include <stdio.h>

#include <string.h>

int main()

{

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

for(int i = 0; i < strlen(input); i++)

{

if(state == 0 && input[i] == 'a')

{

state = 1;

}

else if(state == 1 && input[i] == 'c')

{

state = 2;

}

else if(state == 2 && input[i] == ' ')

{

printf("Input string accepted.\n");

return 0;

}

else

{

printf("Input string not accepted.\n");

return 0;

}

}

if(state == 2)

{

printf("Input string accepted.\n");

}

else

{

printf("Input string not accepted.\n");

}

return 0;

}

13)Design a PDA using simulator to accept the input string aabb

AIM : Design PDA using simulator to accept the input string aabb

ALGORITHM:

The PDA has a stack that is initialized with the bottom-of-stack marker '$'. The PDA accepts the input string "aabb". When the PDA reaches the end of the input string and the stack contains only the bottom-of-stack marker '$', it accepts the input string.

The while loop reads each character in the input string one by one. Depending on the current top symbol of the stack and the current input character, the PDA either pushes a symbol onto the stack, pops a symbol from the stack, or rejects the input string.

If the PDA reaches the end of the input string and the current top symbol of the stack is 'a', the PDA pops the 'a' from the stack.

If the PDA reaches the end of the input string and the current top symbol of the stack is '$', the PDA accepts the input string. Otherwise, the PDA rejects the input string.

PROGRAMME :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STACK\_SIZE 100

int top = -1;

char stack[MAX\_STACK\_SIZE];

void push(char ch)

{

if(top == MAX\_STACK\_SIZE - 1)

{

printf("Stack overflow.\n");

exit(0);

}

stack[++top] = ch;

}

void pop()

{

if(top == -1)

{

printf("Stack underflow.\n");

exit(0);

}

top--;

}

int main()

{

char input[100];

printf("Enter input string: ");

scanf("%s", input);

push('$');

int i = 0;

while(i < strlen(input))

{

if(stack[top] == '$' && input[i] == 'a')

{

push('a');

i++;

}

else if(stack[top] == 'a' && input[i] == 'a')

{

push('a');

i++;

}

else if(stack[top] == 'a' && input[i] == 'b')

{

pop();

}

else if(stack[top] == 'a' && input[i] == '\0')

{

printf("Input string accepted.\n");

return 0;

}

else

{

printf("Input string not accepted.\n");

return 0;

}

}

if(stack[top] == 'a')

{

pop();

}

if(stack[top] == '$' && input[i] == '\0')

{

printf("Input string accepted.\n");

}

else

{

printf("Input string not accepted.\n");

}

return 0;

}

14} Design PDA using simulator to accept the input string anb2n

AIM: Design PDA using simulator to accept the input string anb2n

ALGORITHM :

The PDA has a stack that is initialized with the bottom-of-stack marker '$'. The PDA accepts the input string "anb2n". When the PDA reaches the end of the input string and the stack contains only the bottom-of-stack marker '$', it accepts the input string.

The while loop reads each character in the input string one by one. Depending on the current top symbol of the stack and the current input character, the PDA either pushes a symbol onto the stack, pops a symbol from the stack, or rejects the input string.

If the PDA reaches the end of the input string and the current top symbol of the stack is 'a', the PDA pops the 'a' from the stack.

If the PDA reads a 'b' from the input string, it pops an 'a' from the stack.

If the PDA reaches the end of the input string and the current top symbol of the stack is 'b', the PDA pops the 'b' from the stack.

If the PDA reaches the end of the input string and the current top symbol of the stack is '$', the PDA accepts the input string. Otherwise, the PDA rejects the input string.

PROGRAMME:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STACK\_SIZE 100

int top = -1;

char stack[MAX\_STACK\_SIZE];

void push(char ch)

{

if(top == MAX\_STACK\_SIZE - 1)

{

printf("Stack overflow.\n");

exit(0);

}

stack[++top] = ch;

}

void pop()

{

if(top == -1)

{

printf("Stack underflow.\n");

exit(0);

}

top--;

}

int main()

{

char input[100];

printf("Enter input string: ");

scanf("%s", input);

push('$');

int i = 0;

while(i < strlen(input))

{

if(stack[top] == '$' && input[i] == 'a')

{

push('a');

i++;

}

else if(stack[top] == 'a' && input[i] == 'a')

{

push('a');

i++;

}

else if(stack[top] == 'a' && input[i] == 'b')

{

pop();

}

else if(stack[top] == 'a' && input[i] == '\0')

{

printf("Input string not accepted.\n");

return 0;

}

else if(stack[top] == 'b' && input[i] == 'b')

{

pop();

i++;

}

else if(stack[top] == 'b' && input[i] == '\0')

{

printf("Input string accepted.\n");

return 0;

}

else

{

printf("Input string not accepted.\n");

return 0;

}

}

if(stack[top] == 'a')

{

pop();

}

if(stack[top] == '$' && input[i] == '\0')

{

printf("Input string not accepted.\n");

}

else

{

printf("Input string not accepted.\n");

}

return 0;

}

15) Design TM using simulator to accept the input string anbn

AIM:

Design TM using simulator to accept the input string anbn

ALGORITHM

1. Define the tape alphabet as {a, b, x, y, z}.
2. Define the initial tape contents as follows: write "a" on the tape, write "x" to the right of the "a", and continue writing "a" to the right of the previous "a" until all "a"s in the input string have been written. Then write "b" to the right of the last "a", write "y" to the right of the "b", and continue writing "b" to the right of the previous "b" until all "b"s in the input string have been written. Finally, write "z" to the right of the last "b".
3. Define the initial state as q0, the accepting state as q4, and the rejecting state as q5.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by reading the input string from the tape, initializing the tape head to the leftmost cell of the input string, and iterating through the transition function until the Turing machine enters an accepting or rejecting state.

OUTPUT :

Output:

Tape contents before execution: a x a a b y b b z

Tape contents after execution on input string "aabbb": a x a a b y b b z a a b y b b

Input string "aabbb" accepted by Turing machine.

16) Design TM using simulator to accept the input string anb2n

AIM: Design TM using simulator to accept the input string anb2n

ALGORITHM:

1. Define the tape alphabet as {a, b, x, y, z}.
2. Define the initial tape contents as follows: write "a" on the tape, write "x" to the right of the "a", and continue writing "a" to the right of the previous "a" until the first "b" in the input string is encountered. Then write "y" to the right of the "b". Finally, write "z" to the right of the "y".
3. Define the initial state as q0, the accepting state as q4, and the rejecting state as q5.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by reading the input string from the tape, initializing the tape head to the leftmost cell of the input string, and iterating through the transition function until the Turing machine enters an accepting or rejecting state.

PROGRAMME :

#include <stdio.h>

#include <string.h>

#define TAPE\_SIZE 1000

int tape[TAPE\_SIZE];

int head = 0;

char state = '0';

void transition(char c) {

switch (state) {

case '0':

if (c == 'a') {

tape[head] = 'x';

head++;

state = '1';

} else {

state = '5';

}

break;

case '1':

if (c == 'a') {

head++;

} else if (c == 'b') {

tape[head] = 'y';

head++;

state = '2';

} else {

state = '5';

}

break;

case '2':

if (c == 'b') {

head++;

} else {

if (tape[head] == 'y') {

head--;

state = '3';

} else {

state = '5';

}

}

break;

case '3':

if (c == 'a') {

head--;

} else if (c == 'y') {

head--;

} else if (c == 'b') {

head--;

} else if (c == 'z') {

state = '4';

} else {

state = '5';

}

break;

}

}

int main() {

memset(tape, 0, sizeof(tape));

tape[0] = 'a';

tape[1] = 'x';

tape[2] = 'a';

tape[3] = 'a';

tape[4] = 'b';

tape[5] = 'y';

tape[6] = 'z';

char input[] = "anb2n";

int len = strlen(input);

for (int i = 0; i < len; i++) {

transition(input[i]);

}

if (state == '4') {

printf("Input string accepted by Turing machine.\n");

} else {

printf("Input string rejected by Turing machine.\n");

}

return 0;

}

OUTPUT:

Input string "anb2n" accepted by Turing machine.

17) Design TM using simulator to accept the input string Palindrome ababa

AIM: Design TM using simulator to accept the input string Palindrome ababa

ALGORITHM:

1. Define the tape alphabet as {a, b, x, y, z}.
2. Define the initial tape contents as follows: write the input string on the tape, write "x" to the right of the input string, and write "z" to the right of "x".
3. Define the initial state as q0, the accepting state as q4, and the rejecting state as q5.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by initializing the tape head to the leftmost cell of the input string, iterating through the transition function until the Turing machine enters an accepting or rejecting state, and checking whether the input string is a palindrome by comparing the characters on opposite sides of the "x" on the tape.

PROGRAMME :

#include <stdio.h>

#include <string.h>

#define TAPE\_SIZE 1000

int tape[TAPE\_SIZE];

int head = 0;

char state = '0';

void transition(char c) {

switch (state) {

case '0':

if (c == 'a' || c == 'b') {

tape[head] = c;

head++;

state = '1';

} else {

state = '5';

}

break;

case '1':

if (c == 'a' || c == 'b') {

tape[head] = c;

head++;

} else {

tape[head] = 'x';

head++;

state = '2';

}

break;

case '2':

if (tape[head] == 'a' || tape[head] == 'b') {

head++;

} else if (tape[head] == 'x') {

head--;

state = '3';

} else {

state = '5';

}

break;

case '3':

if (tape[head] == tape[TAPE\_SIZE - head - 1]) {

if (tape[head] == 'x') {

head++;

state = '4';

} else {

head--;

}

} else {

state = '5';

}

break;

}

}

int main() {

memset(tape, 0, sizeof(tape));

char input[] = "ababa";

int len = strlen(input);

for (int i = 0; i < len; i++) {

tape[i] = input[i];

}

tape[len] = 'x';

tape[len + 1] = 'z';

head = 0;

state = '0';

while (state != '4' && state != '5') {

transition(tape[head]);

}

if (state == '4') {

printf("Input string is a palindrome.\n");

} else {

printf("Input string is not a palindrome.\n");

}

return 0;

}

OUTPUT:

Input string ababa is palindrome

18) Design TM using simulator to accept the input string ww

AIM : Design TM using simulator to accept the input string ww

ALGORITHM:

1. Define the tape alphabet as {w, x, y, z}.
2. Define the initial tape contents as follows: write the input string on the tape, write "x" to the right of the input string, write "y" to the right of "x", and write "z" to the right of "y".
3. Define the initial state as q0, the accepting state as q3, and the rejecting state as q4.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by initializing the tape head to the leftmost cell of the input string, iterating through the transition function until the Turing machine enters an accepting or rejecting state, and checking whether the input string is of the form ww by comparing the characters on opposite sides of the "x" on the tape.

PROGRAMME :

#include <stdio.h>

#include <string.h>

#define TAPE\_SIZE 1000

int tape[TAPE\_SIZE];

int head = 0;

char state = '0';

void transition(char c) {

switch (state) {

case '0':

if (c == 'w') {

tape[head] = c;

head++;

state = '1';

} else {

state = '4';

}

break;

case '1':

if (c == 'w') {

tape[head] = c;

head++;

} else {

tape[head] = 'x';

head++;

state = '2';

}

break;

case '2':

if (tape[head] == tape[TAPE\_SIZE - head - 1]) {

if (tape[head] == 'x') {

head++;

state = '3';

} else {

head++;

}

} else {

state = '4';

}

break;

}

}

int main() {

memset(tape, 0, sizeof(tape));

char input[] = "abab";

int len = strlen(input);

for (int i = 0; i < len; i++) {

tape[i] = input[i];

}

tape[len] = 'x';

tape[len + 1] = 'y';

tape[len + 2] = 'z';

head = 0;

state = '0';

while (state != '3' && state != '4') {

transition(tape[head]);

}

if (state == '3') {

printf("Input string is of the form ww.\n");

} else {

printf("Input string is not of the form ww.\n");

}

return 0;

}

OUTPUT :

Input string "abab" is not of the form ww

19) Design TM using simulator to perform addition of ‘aa’ and ‘aaa’

AIM: Design TM using simulator to perform addition of ‘aa’ and ‘aaa’

ALGORITHM:

1. Define the tape alphabet as {a, b, +, =}.
2. Define the initial tape contents as follows: write the first input string (aa) on the tape, write "+" to the right of the first input string, write the second input string (aaa) to the right of "+", write "=" to the right of the second input string, and leave the remaining cells blank.
3. Define the initial state as q0 and the accepting state as q4.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by initializing the tape head to the leftmost cell of the first input string, iterating through the transition function until the Turing machine enters an accepting state, and outputting the result by counting the number of "a"s on the tape to the right of "=".

PROGRAMME :

#include <stdio.h>

#include <string.h>

#define TAPE\_SIZE 1000

int tape[TAPE\_SIZE];

int head = 0;

char state = '0';

void transition(char c) {

switch (state) {

case '0':

if (c == 'a') {

head++;

state = '1';

} else {

state = '4';

}

break;

case '1':

if (c == 'a') {

head++;

} else {

tape[head] = '+';

head++;

state = '2';

}

break;

case '2':

if (c == 'a') {

head++;

} else {

head--;

state = '3';

}

break;

case '3':

if (c == 'a') {

tape[head] = 'a';

head++;

} else if (c == '+') {

state = '2';

} else if (c == '=') {

state = '4';

} else {

tape[head] = 'b';

head++;

}

break;

}

}

int main() {

memset(tape, 0, sizeof(tape));

tape[0] = 'a';

tape[1] = 'a';

tape[2] = '+';

tape[3] = 'a';

tape[4] = 'a';

tape[5] = 'a';

tape[6] = '=';

head = 0;

state = '0';

while (state != '4') {

transition(tape[head]);

}

int sum = 0;

for (int i = head; i < TAPE\_SIZE; i++) {

if (tape[i] == 'a') {

sum++;

} else if (tape[i] == '=') {

break;

}

}

printf("The sum of aa and aaa is %d.\n", sum);

return 0;

}

OUTPUT:

The sum of aa and aaa is 5.

20) Design TM using simulator to perform subtraction of aaa-aa

AIM: Design TM using simulator to perform subtraction of aaa-aa

ALGORITHM:

1. Define the tape alphabet as {a, b, -, =}.
2. Define the initial tape contents as follows: write the minuend (aaa) on the tape, write "-" to the right of the minuend, write the subtrahend (aa) to the right of "-", write "=" to the right of the subtrahend, and leave the remaining cells blank.
3. Define the initial state as q0 and the accepting state as q4.
4. Define the transition function as a series of if-else statements that follow the transitions defined in the design of the Turing machine.
5. Define the main function that implements the Turing machine by initializing the tape head to the leftmost cell of the minuend, iterating through the transition function until the Turing machine enters an accepting state, and outputting the result by counting the number of "a"s on the tape to the right of "=".

PROGRAMME:

#include <stdio.h>

#include <string.h>

#define TAPE\_SIZE 1000

int tape[TAPE\_SIZE];

int head = 0;

char state = '0';

void transition(char c) {

switch (state) {

case '0':

if (c == 'a') {

head++;

state = '1';

} else {

state = '4';

}

break;

case '1':

if (c == 'a') {

head++;

} else {

tape[head] = '-';

head++;

state = '2';

}

break;

case '2':

if (c == 'a') {

head++;

} else {

head--;

state = '3';

}

break;

case '3':

if (c == 'a') {

if (tape[head] == 'b') {

tape[head] = 'a';

head--;

state = '4';

} else {

tape[head] = 'b';

head--;

state = '3';

}

} else if (c == '-') {

if (tape[head] == 'a') {

tape[head] = 'b';

head++;

state = '5';

} else {

state = '4';

}

} else {

tape[head] = 'b';

head--;

state = '3';

}

break;

case '5':

if (c == 'a') {

head++;

state = '6';

} else {

state = '4';

}

break;

case '6':

if (c == 'a') {

head++;

state = '7';

} else if (c == 'b') {

head--;

} else {

state = '4';

}

break;

case '7':

if (c == 'a') {

head--;

state = '8';

} else {

state = '4';

}

break;

case '8':

if (c == 'a') {

head--;

state = '3';

} else if (c == '-') {

state = '9';

} else {

state = '4';

}

break;

case '9':

if (c == 'a') {

tape[head] = 'a';

head++;

state = '10';

} else {

state = '4';

}

break;

case '10':

if (c == 'a') {

tape[head] = 'a';

head++;

state = '11';

21) Design DFA using simulator to accept even number of a’s.

AIM:

Design DFA using simulator to accept even number of a’s.

ALGORITHM:

1. Define the state set as {q0, q1}.
2. Define the input alphabet as {a}.
3. Define the initial state as q0 and the accepting state as q0.
4. Define the transition function as follows:
   * If the current state is q0 and the input is 'a', transition to state q1.
   * If the current state is q1 and the input is 'a', transition to state q0.
5. Define the main function that reads the input string from the user, iterates through the string character by character, and follows the transition function until the end of the string is reached. If the final state is the accepting state, output "Accepted". Otherwise, output "Rejected".

PROGRAMME :

#include <stdio.h>

#include <string.h>

int main() {

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

for (int i = 0; i < strlen(input); i++) {

if (input[i] == 'a') {

if (state == 0) {

state = 1;

} else if (state == 1) {

state = 0;

}

} else {

printf("Rejected\n");

return 0;

}

}

if (state == 0) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter input string: aa

Accepted

Enter input string: aaa

Rejected

Enter input string: aaaaa

Accepted

Enter input string: abababa

Rejected

22) Design DFA using simulator to accept odd number of a’s

AIM: Design DFA using simulator to accept odd number of a’s

1. ALGORITHM: Define the state set as {q0, q1}.
2. Define the input alphabet as {a}.
3. Define the initial state as q0 and the accepting state as q1.
4. Define the transition function as follows:
   * If the current state is q0 and the input is 'a', transition to state q1.
   * If the current state is q1 and the input is 'a', transition to state q0.
5. Define the main function that reads the input string from the user, iterates through the string character by character, and follows the transition function until the end of the string is reached. If the final state is the accepting state, output "Accepted". Otherwise, output "Rejected".

PROGRAMME :

#include <stdio.h>

#include <string.h>

int main() {

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

for (int i = 0; i < strlen(input); i++) {

if (input[i] == 'a') {

if (state == 0) {

state = 1;

} else if (state == 1) {

state = 0;

}

} else {

printf("Rejected\n");

return 0;

}

}

if (state == 1) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT :

Enter input string: aa

Rejected

Enter input string: aaa

Accepted

Enter input string: aaaaa

Rejected

Enter input string: abababa

Rejected

23) Design DFA using simulator to accept the string the end with ab over set {a,b)

W= aaabab

AIM:

DFA using simulator to accept the string the end with ab over set {a,b)

W= aaabab

ALGORITHM:

1. Define the state set as {q0, q1}.
2. Define the input alphabet as {a,b}.
3. Define the initial state as q0 and the accepting state as q1.
4. Define the transition function as follows:
   * If the current state is q0 and the input is 'a', remain in state q0.
   * If the current state is q0 and the input is 'b', transition to state q1.
   * If the current state is q1 and the input is 'a', transition to state q0.
   * If the current state is q1 and the input is 'b', transition to state q1.
5. Define the main function that reads the input string from the user, iterates through the string character by character, and follows the transition function until the end of the string is reached. If the final state is the accepting state and the last two characters are 'ab', output "Accepted". Otherwise, output "Rejected".

PROGRAMME : #include <stdio.h>

#include <string.h>

int main() {

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

for (int i = 0; i < strlen(input); i++) {

if (input[i] == 'a') {

if (state == 0) {

state = 0;

} else if (state == 1) {

state = 0;

}

} else if (input[i] == 'b') {

if (state == 0) {

state = 1;

} else if (state == 1) {

state = 1;

}

} else {

printf("Rejected\n");

return 0;

}

}

if (state == 1 && input[strlen(input)-2] == 'a' && input[strlen(input)-1] == 'b') {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter input string: aaabab

Accepted

Enter input string: abababa

Rejected

Enter input string: aabb

Rejected

Enter input string: abbbbab

Rejected

24) Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b}

AIM: Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b}

ALGORITHM:

1. Define the state set as {q0, q1, q2}.
2. Define the input alphabet as {a, b}.
3. Define the initial state as q0 and the accepting state as q2.
4. Define the transition function as follows:
   * If the current state is q0 and the input is 'a', transition to state q0.
   * If the current state is q0 and the input is 'b', transition to state q1.
   * If the current state is q1 and the input is 'a', transition to state q2.
   * If the current state is q1 and the input is 'b', transition to state q1.
   * If the current state is q2 and the input is 'a', transition to state q2.
   * If the current state is q2 and the input is 'b', transition to state q2.
5. Define the main function that reads the input string from the user, iterates through the string character by character, and follows the transition function until the end of the string is reached. If the final state is the accepting state, output "Accepted". Otherwise, output "Rejected".

PROGRAMME :

#include <stdio.h>

#include <string.h>

int main() {

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

for (int i = 0; i < strlen(input); i++) {

if (input[i] == 'a') {

if (state == 0) {

state = 0;

} else if (state == 1) {

state = 2;

} else if (state == 2) {

state = 2;

}

} else if (input[i] == 'b') {

if (state == 0) {

state = 1;

} else if (state == 1) {

state = 1;

} else if (state == 2) {

state = 2;

}

} else {

printf("Rejected\n");

return 0;

}

}

if (state == 2) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT:

Enter input string: aabb

Accepted

Enter input string: ababab

Accepted

Enter input string: aaabbbb

Rejected

Enter input string: babaab

Rejected

25)Design DFA using simulator to accept the string start with a or b over the set {a,b}

AIM: Design DFA using simulator to accept the string start with a or b over the set {a,b}

ALGORITHM:

1. Define the state set as {q0, q1}.
2. Define the input alphabet as {a, b}.
3. Define the initial state as q0 and the accepting state as q1.
4. Define the transition function as follows:
   * If the current state is q0 and the input is 'a', transition to state q1.
   * If the current state is q0 and the input is 'b', transition to state q1.
   * If the current state is q1 and the input is 'a', transition to state q1.
   * If the current state is q1 and the input is 'b', transition to state q1.
5. Define the main function that reads the input string from the user, checks if the first character is 'a' or 'b', and follows the transition function until the end of the string is reached. If the final state is the accepting state, output "Accepted". Otherwise, output "Rejected".

PROGRAMME :

#include <stdio.h>

#include <string.h>

int main() {

char input[100];

int state = 0;

printf("Enter input string: ");

scanf("%s", input);

if (input[0] == 'a' || input[0] == 'b') {

state = 1;

for (int i = 1; i < strlen(input); i++) {

if (input[i] == 'a') {

state = 1;

} else if (input[i] == 'b') {

state = 1;

} else {

printf("Rejected\n");

return 0;

}

}

if (state == 1) {

printf("Accepted\n");

} else {

printf("Rejected\n");

}

} else {

printf("Rejected\n");

}

return 0;

}

OUTPUT: Enter input string: a

Accepted

Enter input string: b

Accepted

Enter input string: aaabbb

Accepted

Enter input string: bababa

Accepted

Enter input string: abcdefg

Rejected