Implementation of Turing Machine

Project report

Submitted by:

Names	**Registration numbers**
SR NAVYA SREE	16BCE0223
SHAIK DILSHATH	16BCE0669
ARYAN SAXENA	16BCE0022
VATSAL AGRAWAL	16BCB0090
SRI MADHU CHOWDARY	16BCE0476
GORREPATI MEDHA	16BCE0684

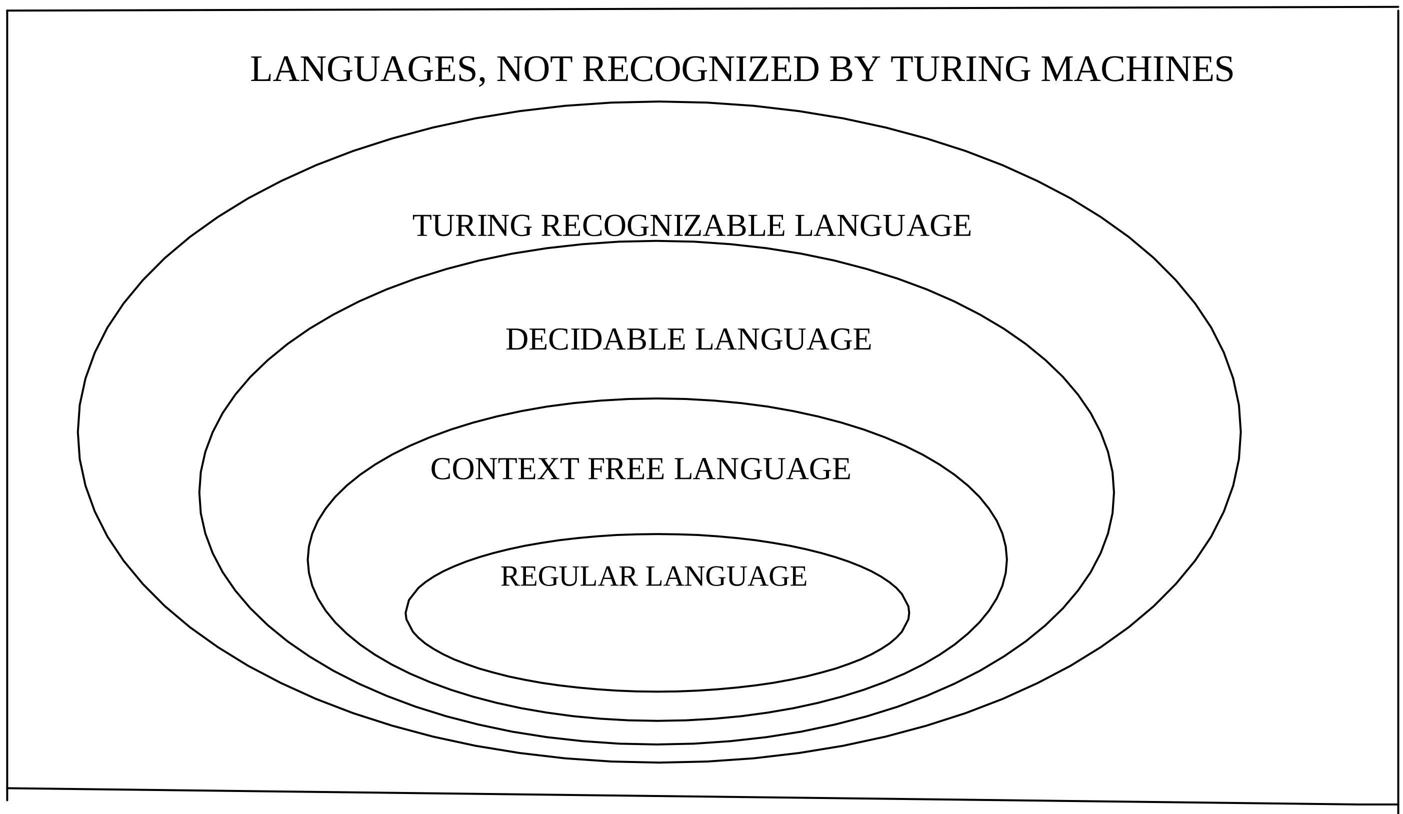
Submitted to:

Dr. Debi Prasanna Acharjya

Professor, SCOPE

VIT University, Vellore

INTRODUCTION OF LANGUAGES:



TURING MACHINE:

Introduced by Alan Turing in 1936

- ✓ A simple mathematical model of a computer.
- ✓ Models the computing capability of a computer.

DEFINITION:

- ✓ A Turing machine (TM) is a finite-state machine with an infinite tape and a tape head that can read or write one tape cell and move left or right.
- ✓ It normally accepts the input string, or completes its computation, by entering a final or accepting state.
- ✓ Tape is use for input and working storage.

OVERVIEW

The Turing machine actually consists of:

- ✓ An input and Output tape
- ✓ The turing machine itself
- ✓ A rule list
- ✓ The input/output tape is divided into cells
- ✓ The cell contains the input and output symbols and changes frequently as the program is running.
- ✓ The standard turing machine actually consists of only a single tape.

PARAMETERS:

- The behavior of turing machine is completely determined by three parameters
- The state the machine is in
- The number of the cells it is scanning
- A table of instructions.

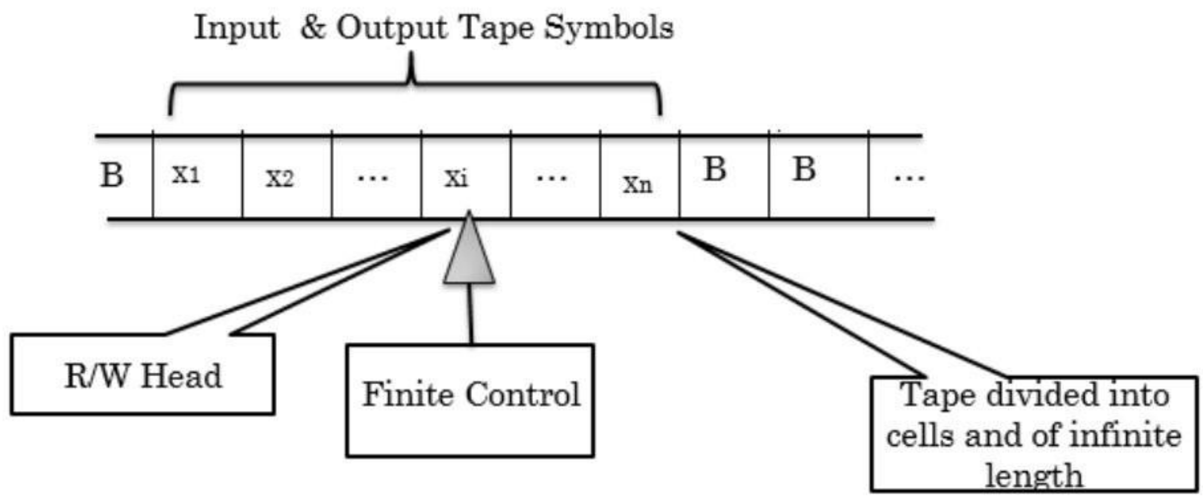
Representation of Turing Machine

T**uring Machine is represented by M=(Q,**S**,** Γ**,δ,q0,B,F) :**

- ✓ **Q** is the finite state of states
- ✓ S a set of **τ** not including B, is the set of input symbols,
- ✓ **τ** is the finite state of allowable tape symbols,
- ✓ **δ** is the next move function, a mapping from Q × Γ to **Q ×** Γ **×{L,R}**
- ✓ **Q0** in Q is the start state,
- ✓ **B** a symbol of Γ is the blank,
- ✓ **F** is the set of final states.

MODEL OF TURING MACHINE:

TRANSITION



TRANSITION FUNCTION

- One move (denoted by |---) in a TM does the following:
- δ***(q , X) = (p ,Y ,R/L)***
- q is the current state
- X is the current tape symbol pointed by tape head
- State changes from q to p
- The Turing machine may
  - Halt and accept the input
  - Halt and reject the input, or
  - Never halt /loop.

Implementation:

The simulator is configured to accept on input strings from the language:

$$L = \{ 0^{2^n} \mid n > 0 \}$$

Or: all strings of 0s whose length is a power of 2.

Code:

File: turing.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include "turing.h"

int state_id = 0;

void die( char *message )
{
    if( message )
    {
        printf( "Error: %s.\n", message );
    }

    // exit unsuccesfully
    exit(1);
}

Transition* Transition_create( char input, char write, Direction move, State
*next )
{
    // allocate memory
    Transition *trans = malloc( sizeof( Transition ));
    if( ! trans ) die( "Memory error" );

    trans->input = input;
    trans->write = write;
```

```
    trans->move = move;
    trans->next = next;

    return trans;
}

void Transition_destroy( Transition* trans )
{
    free( trans );
}

State* State_create( Bool accept, Bool reject )
{
    // allocate mem
    State *state = malloc( sizeof( State ));
    if( ! state ) die( "Memory error" );

    state->id = state_id++;
    state->accept = accept;
    state->reject = reject;
    state->trans_count = 0;

    return state;
}

void State_add_transition( State *state, Transition *trans )
{
    // check if we can still add another transition
    if( state->trans_count == MAX_TRANSITIONS ) {
        char buffer[ 50 ];
        sprintf( buffer, "State %d already has the maximum amount of
transitions.", state->id );

        die( buffer );
    }

    // add the transition
    state->transitions[ state->trans_count ] = trans;
    state->trans_count++;
}

void State_destroy( State *state )
{
    int i = 0;
```

```
    // loop over its transitions
    for( i = 0; i < state->trans_count; i++ ) {
        Transition *trans = state->transitions[ i ];
        if( !trans ) die( "Could not fetch transition." );

        Transition_destroy( trans );
    }

    free( state );
}

Turing* Turing_create()
{
    // allocate mem
    Turing *machine = malloc( sizeof( Turing ));

    machine->state_count = 0;
    machine->current = NULL;
    machine->head = 0;

    return machine;
}

void Turing_destroy( Turing *machine )
{
    int i = 0;

    // loop over it's states
    for( i = 0; i < machine->state_count; i++ ) {
        State *state = machine->states[ i ];
        if( !state ) die( "Could not fetch turing state" );

        State_destroy( state );
    }

    free( machine );
}

void Turing_add_state( Turing *machine, State *state )
{
    if( machine->state_count == MAX_STATES ) {
        die( "The turing machine already has the maximum amount of states" );
    }

    // add the state
```

```
    machine->states[ machine->state_count++ ] = state;
}

State* Turing_step( Turing *machine, char* tape, int tape_len )
{
    int i = 0;
    char input = tape[ machine->head ];
    State* state = machine->current;

    // look for a transition on the given input
    for( i = 0; i < state->trans_count; i++ ) {
        Transition* trans = state->transitions[ i ];
        if( !trans ) die( "Transition retrieval error" );

        // check if this is a transition in the given char input
        if( trans->input == input ) {

            State *next = trans->next;
            if( !next ) die( "Transitions to NULL state" );

            // write if nescesary
            if( trans->write != '\0' ) {
                            tape[ machine->head ] = trans->write;
            }

            // move the head
            if( trans->move == LEFT ) {
                if( machine->head > 0 ) {
                    machine->head--;
                }
            } else {
                if( machine->head + 1 >= tape_len ) {
                    die( "Machine walked of tape on right side" );
                }

                machine->head++;

            }

            // move the machine to the next state
                machine->current = next;

            return next;
        }
    }
```

```
    char buffer[ 50 ];
    sprintf( buffer, "Turing machine blocked: state %d for input %c", state-
>id, input );

    die( buffer );
}

void Turing_run( Turing *machine, char *tape, int tapelen )
{
    // check if the start state is configured properly
    if( !machine->current ) die( "Turing machine has now start state" );

    while( TRUE ) {
        State* state = Turing_step( machine, tape, tapelen );

        if( state->accept ) {
            printf( "Input accepted in state: %d\n", state->id );
            break;
        } else if( state->reject ) {
            printf( "Input rejected in state: %d\n", state->id );
            break;
        } else {
            printf( "Moved to state: %d\n", state->id );
        }
    }
}

int main( int argc, char* argv[] )
{
    Turing* machine = Turing_create();

    State* q1 = State_create( FALSE, FALSE );
    State* q2 = State_create( FALSE, FALSE );
    State* q3 = State_create( FALSE, FALSE );
    State* q4 = State_create( FALSE, FALSE );
    State* q5 = State_create( FALSE, FALSE );
    State* qaccept = State_create( TRUE, FALSE );
    State* qreject = State_create( FALSE, TRUE );

    Transition* q1_r_space = Transition_create( ' ', '\0', RIGHT, qreject );
    Transition* q1_r_x = Transition_create( 'x', '\0', RIGHT, qreject );
    Transition* q1_q2_zero = Transition_create( '0', ' ', RIGHT, q2 );
    Transition* q2_q2_x = Transition_create( 'x', '\0', RIGHT, q2 );
    Transition* q2_a_space = Transition_create( ' ', '\0', RIGHT, qaccept );
```

```
Transition* q2_q3_zero = Transition_create( '0', 'x', RIGHT, q3 );
Transition* q3_q3_x = Transition_create( 'x', '\0', RIGHT, q3 );
Transition* q3_q4_zero = Transition_create( '0', '\0', RIGHT, q4 );
Transition* q3_q5_space = Transition_create( ' ', '\0', LEFT, q5 );
Transition* q4_q3_zero = Transition_create( '0', 'x', RIGHT, q3 );
Transition* q4_q4_x = Transition_create( 'x', '\0', RIGHT, q4 );
Transition* q4_r_space = Transition_create( ' ', '\0', RIGHT, qreject );
Transition* q5_q5_zero = Transition_create( '0', '\0', LEFT, q5 );
Transition* q5_q5_x = Transition_create( 'x', '\0', LEFT, q5 );
Transition* q5_q2_space = Transition_create( ' ', '\0', RIGHT, q2 );

State_add_transition( q1, q1_r_space );
State_add_transition( q1, q1_r_x );
State_add_transition( q1, q1_q2_zero );
State_add_transition( q2, q2_q2_x );
State_add_transition( q2, q2_a_space );
State_add_transition( q2, q2_q3_zero );
State_add_transition( q3, q3_q3_x );
State_add_transition( q3, q3_q4_zero );
State_add_transition( q3, q3_q5_space );
State_add_transition( q4, q4_q3_zero );
State_add_transition( q4, q4_q4_x );
State_add_transition( q4, q4_r_space );
State_add_transition( q5, q5_q5_zero );
State_add_transition( q5, q5_q5_x );
State_add_transition( q5, q5_q2_space );

Turing_add_state( machine, q1 );
Turing_add_state( machine, q2 );
Turing_add_state( machine, q3 );
Turing_add_state( machine, q4 );
Turing_add_state( machine, q5 );
Turing_add_state( machine, qaccept );
Turing_add_state( machine, qreject );

machine->current = q1;


char* input = "0000000000000000  ";
int len = strlen( input );
char* tape = malloc( len * sizeof( char ));
strcpy( tape, input );
Turing_run( machine, tape, len );

// clean
```

```
    Turing_destroy( machine );
    free( tape );
}
```

File: turing.h

```
#ifndef __turing_h__

#define __turing_h__


#define MAX_TRANSITIONS 5

#define MAX_STATES 25


// forward declare structs

struct State;

struct Transition;


typedef enum {

    LEFT, RIGHT

} Direction;


typedef enum {

    FALSE, TRUE

} Bool;


struct Transition {

    char input; char

    write; Direction

    move; struct State

    *next;
```

```
};

typedef struct Transition Transition;

struct State {
    int id;
    int trans_count;
    struct Transition* transitions[ MAX_TRANSITIONS ];
    Bool accept;
    Bool reject;
};

typedef struct State State;

struct Turing {
    int state_count;
    State* states[ MAX_STATES ];
    State* current;
    int head;
};

typedef struct Turing Turing;

#endif
```

OUTPUT:

output - Notepad

File Edit Format View Help

```
Moved to state: 1
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 3
Moved to state: 2
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 1
Moved to state: 1
Moved to state: 2
Moved to state: 2
```

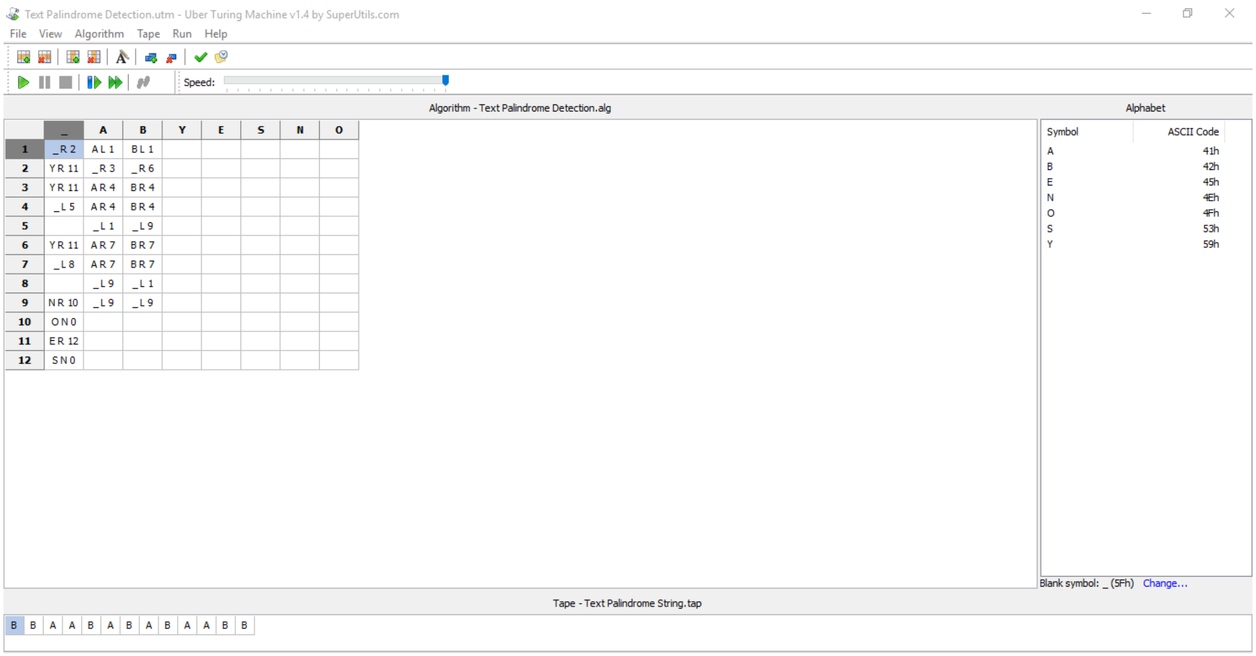
output - Notepad

File Edit Format View Help

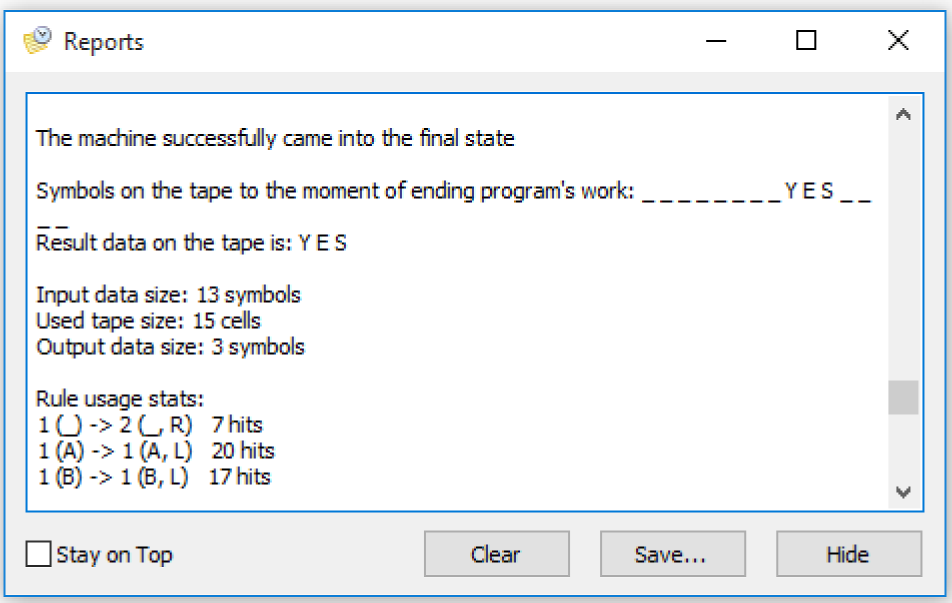
```
Moved to state: 2
Moved to state: 2
Moved to state: 2
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 4
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Moved to state: 1
Input accepted in state: 5
```

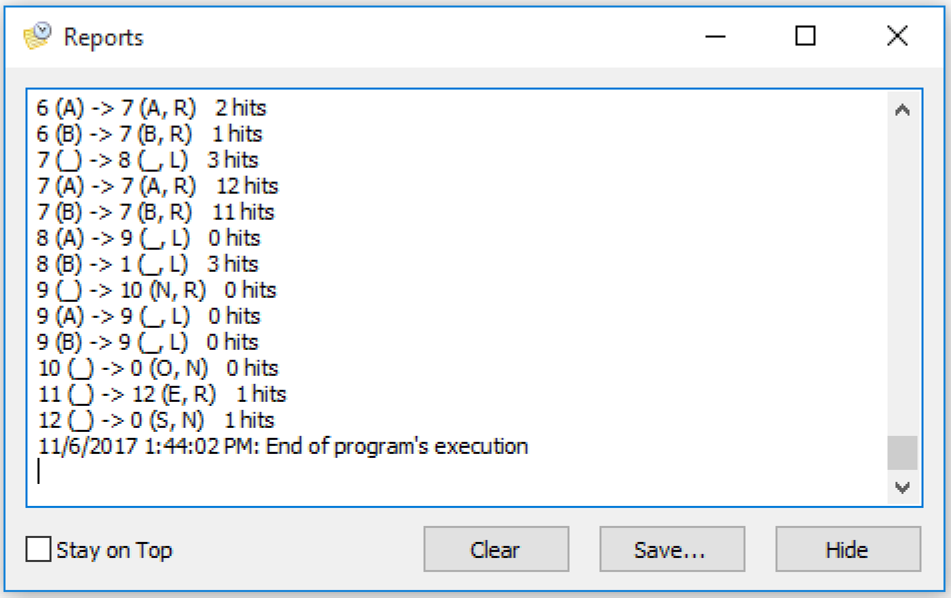
UBER TURING MACHINE:

For palindrome detection:

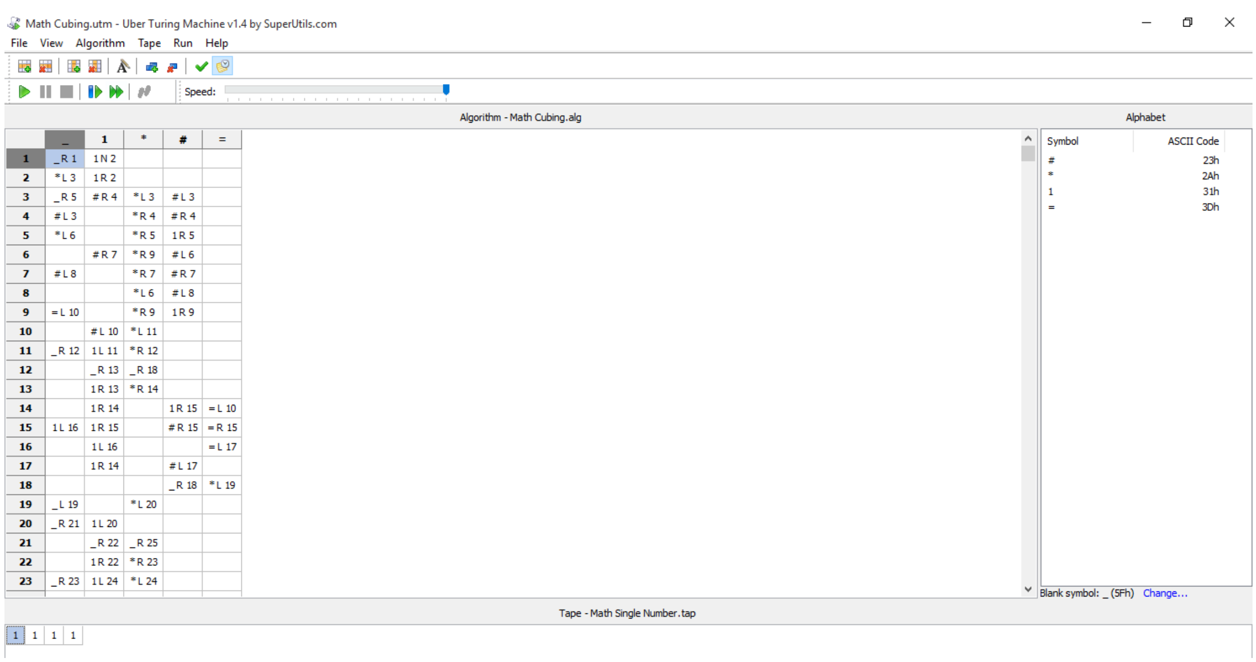


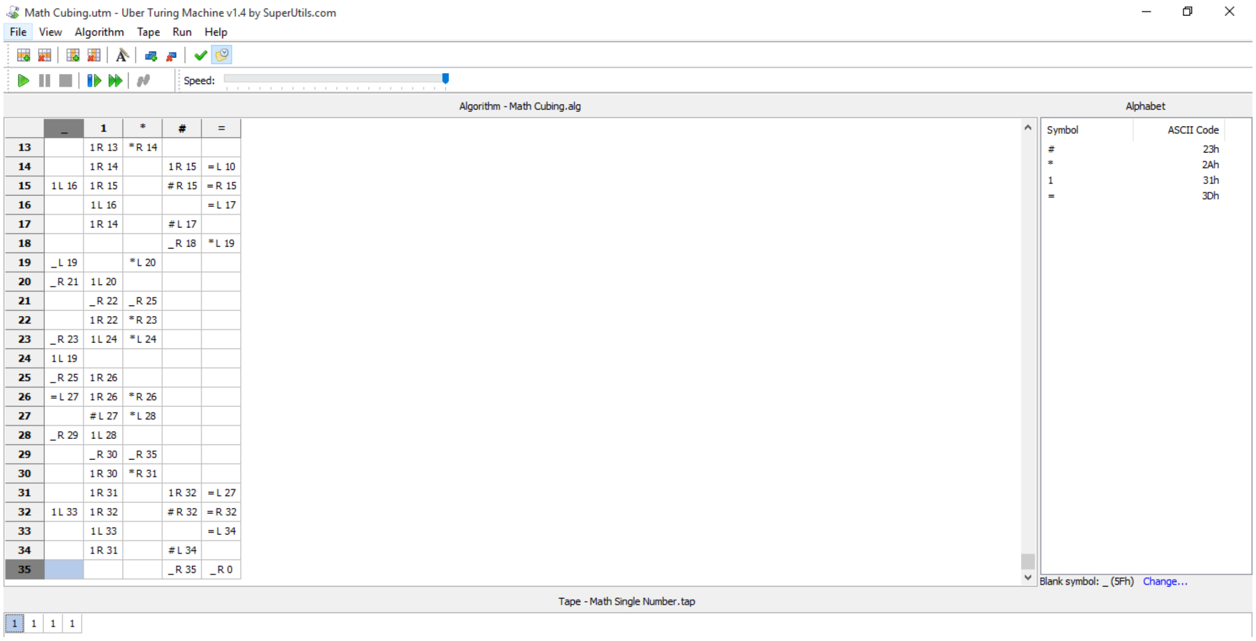
Output:



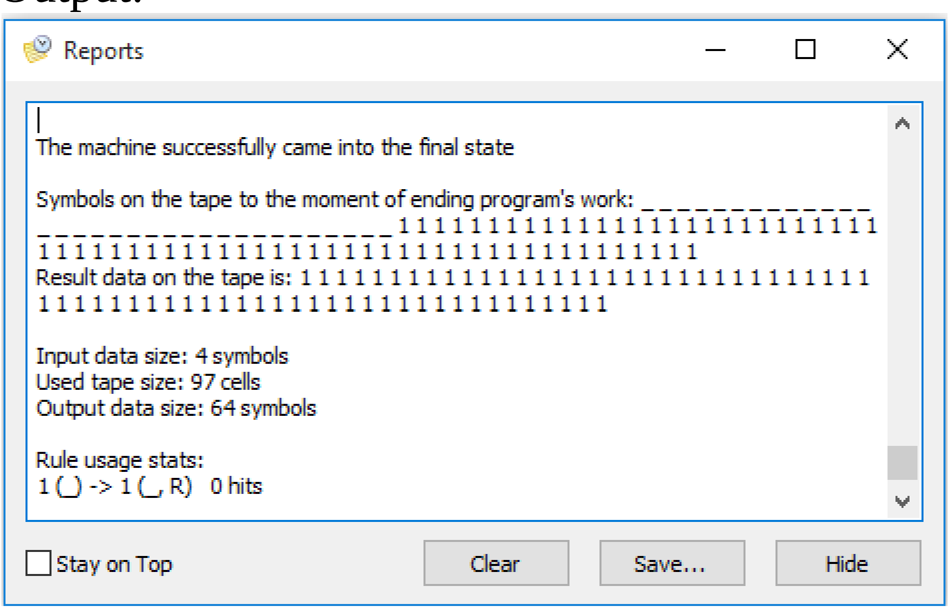


Finding the cubes of the given number:



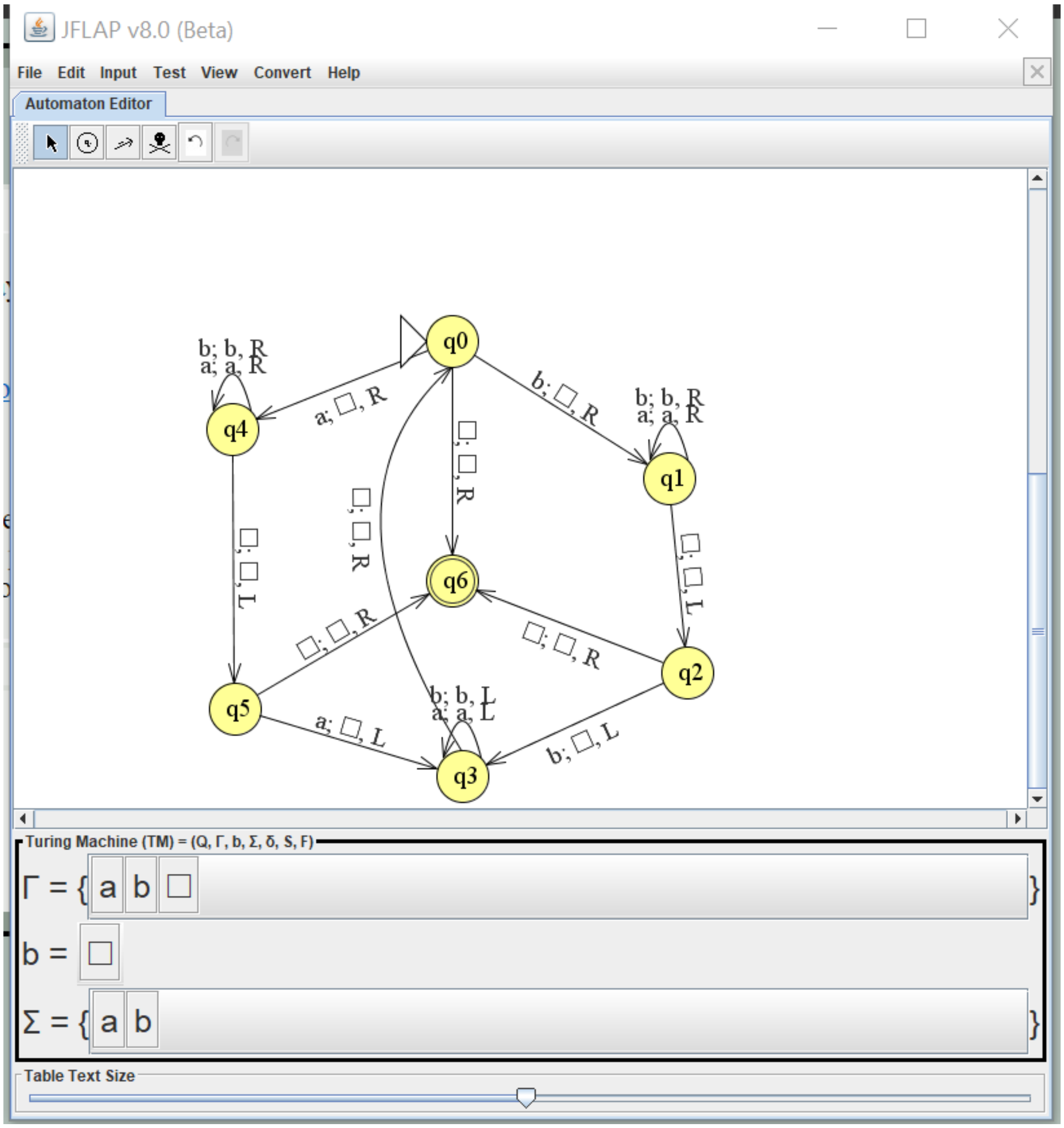

Math Cubing.utm - Uber Turing Machine v1.4 by SuperUtils.com
File View Algorithm Tape Run Help
Speed:
Algorithm - Math Cubing.alg
_ 1 * # =
13 | | 1 R 13 | * R 14 | |
14 | | 1 R 14 | | 1 R 15 | = L 10
15 | 1 L 16 | 1 R 15 | | # R 15 | = R 15
16 | | 1 L 16 | | | = L 17
17 | | 1 R 14 | | # L 17 |
18 | | | | _ R 18 | * L 19
19 | _ L 19 | | * L 20 | |
20 | _ R 21 | 1 L 20 | | |
21 | | _ R 22 | _ R 25 | |
22 | | 1 R 22 | * R 23 | |
23 | _ R 23 | 1 L 24 | * L 24 | |
24 | 1 L 19 | | | |
25 | _ R 25 | 1 R 26 | | |
26 | = L 27 | 1 R 26 | * R 26 | |
27 | | # L 27 | * L 28 | |
28 | _ R 29 | 1 L 28 | | |
29 | | _ R 30 | _ R 35 | |
30 | | 1 R 30 | * R 31 | |
31 | | 1 R 31 | | 1 R 32 | = L 27
32 | 1 L 33 | 1 R 32 | | # R 32 | = R 32
33 | | 1 L 33 | | | = L 34
34 | | 1 R 31 | | # L 34 |
35 | | | | _ R 35 | _ R 0
Alphabet
Symbol ASCII Code
23h
* 2Ah
1 31h
= 3Dh
Blank symbol: _ (5Fh) Change...
Tape - Math Single Number.tap
1 1 1 1

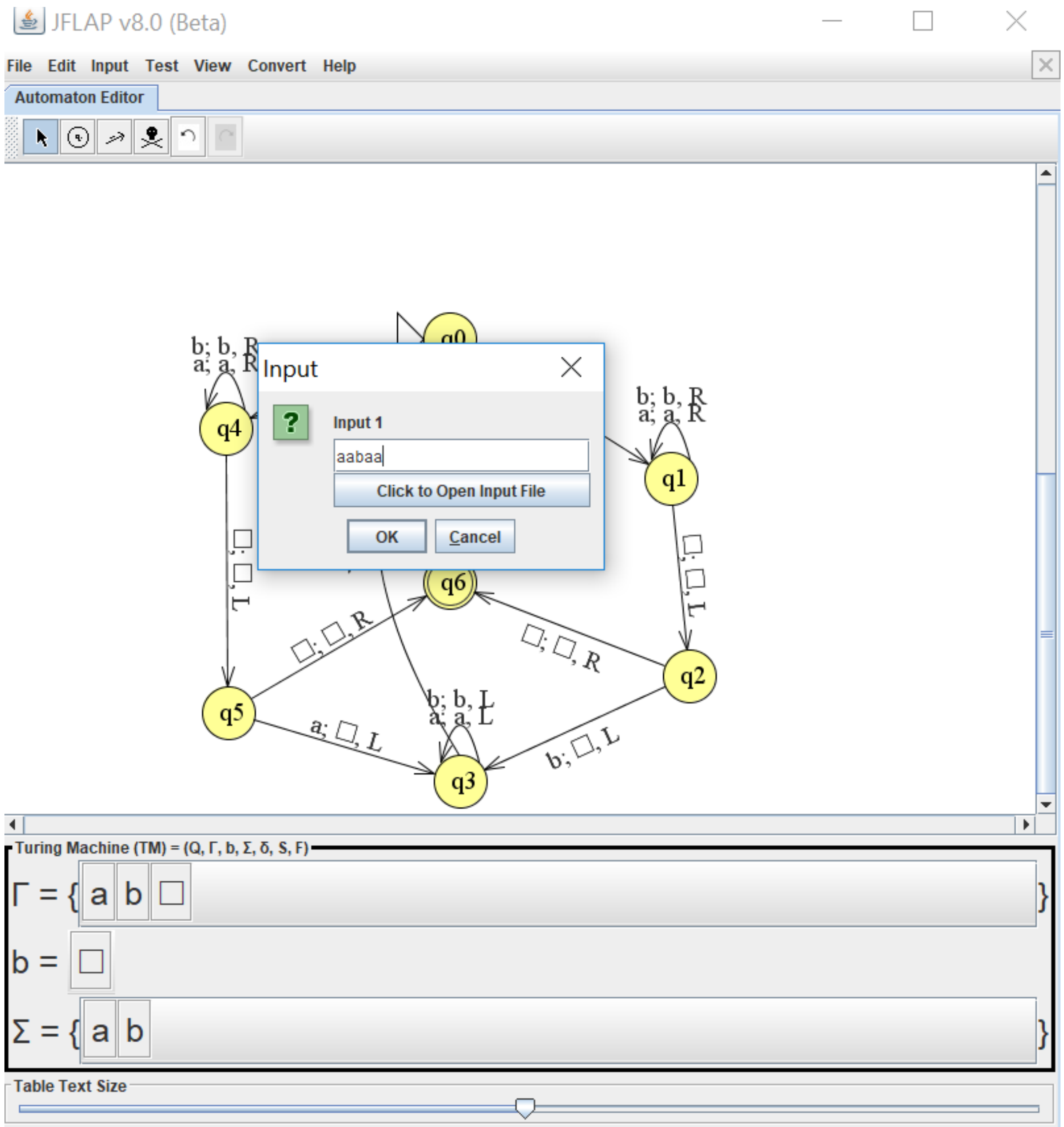
Output:


Reports
The machine successfully came into the final state
Symbols on the tape to the moment of ending program's work: _ _ _ _ _ _ _ _ _ _ _ _ _
_ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Result data on the tape is: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Input data size: 4 symbols
Used tape size: 97 cells
Output data size: 64 symbols
Rule usage stats:
1 (_) -> 1 (_, R) 0 hits
Stay on Top
Clear
Save...
Hide

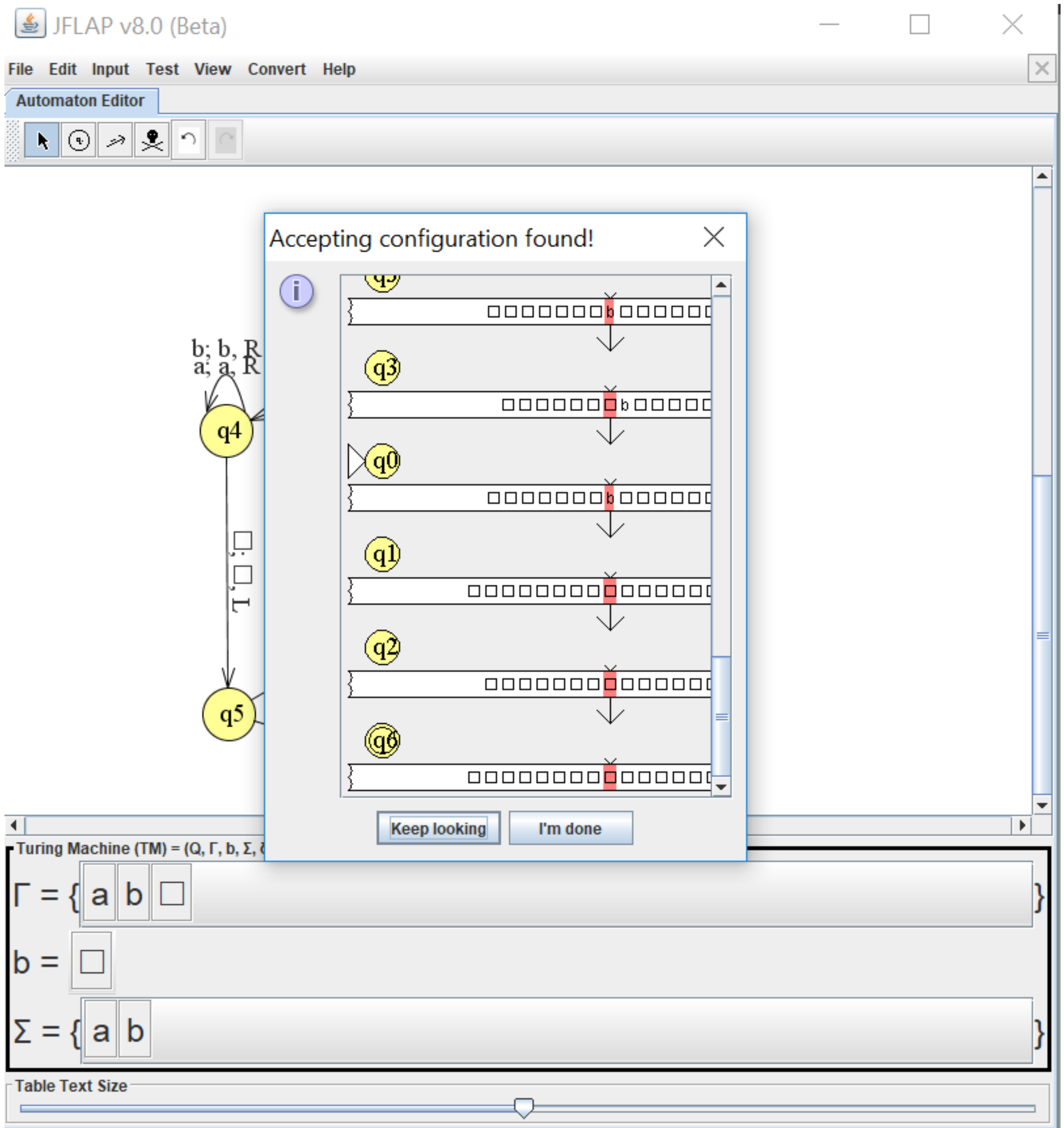
JFLAP:

For palindrome detection:





JFLAP v8.0 (Beta)
File Edit Input Test View Convert Help
Automaton Editor
Input
Input 1
aabaa
Click to Open Input File
OK
Cancel
q0
q1
q2
q3
q4
q5
q6
b; b, R
a; a, R
□; □, L
□; □, R
b; □, L
a; □, L
b; b, L
a; a, L
Turing Machine (TM) = (Q, Γ, b, Σ, δ, S, F)
Γ = { a b □ }
b = □
Σ = { a b }
Table Text Size


JFLAP v8.0 (Beta)
File Edit Input Test View Convert Help
Automaton Editor
Accepting configuration found!
b; b, R
a; a, R
q4
q5
q3
q0
q1
q2
q6
Keep looking
I'm done
Turing Machine (TM) = (Q, Γ, b, Σ,
Γ = { a b □ }
b = □
Σ = { a b }
Table Text Size

CONCLUSION:

The turing machine is the most comprehensive, deep and accessible model of computation extant and its associated theories allow many ideas involving “complexity” to be profitably discussed. In providing a sort of atomic structure for the concept of computation, it has led to new mathematical investigations. One development of the last 30 years, is that of classifying different problems in terms of their complexity. It gives a platform-independent way of measuring this complexity. Now-s-days computer can be used to simulate working of a Turing machine, and so see on the screen. It can have various applications such as enumerator, function computer. We have written a code which depicts the algorithm for 0^2^n. we have also used uber-turing machine and JFLAP.

REFERENCES:

- ✓ THEORY OF COMPUTATION – D.P ACHARJYA,
- ✓ https://introcs.cs.princeton.edu/java/52turing/
- ✓ Emil Post (1936), "Finite Combinatory Processes—Formulation 1", Journal of Symbolic Logic, 1, 103–105, 1936. Reprinted in The Undecidable, pp. 289ff.