Matt Mahan & Matt Rundle

CSE 30151 Spring 2014

Project 3

Turing Machine Simulator

Language and Testing Platform

The Turing machine simulator is implemented in C++, and was developed and tested on the Red Hat Linux machines in the computer cluster of the Fitzpatrick Engineering library. Over the course of the project's development, a private Github repository was used for version control and to facilitate collaborative design and testing.

High Level Program Description

Machine Descriptions:

The Turing machine descriptions are stored in a special "Machine" class which contains all necessary data structures. The Machine class consists of:

- three (3) strings that store the starting state, the accepting state, and the rejecting state
- three (3) vectors that store the list of states, the input alphabet, and the tape alphabet
- one (1) two-dimensional vector that stores a list of transitions for the machine, with each transition being a vector containing the individual transition elements

Upon reading the machine description, the Machine is passed into a "Processor" which verifies that the machine is deterministic (that is, it iterates through the two-dimensional vector of transitions and ensures that no two different transitions have the same starting state and the same current tape symbol.)

Input / Tape Handling:

Each line of the keyboard input from the user (which can also be redirected into the program from a file) is broken into string components using commas as a delimiter. These components were then stored in a vector. After each input line is read, the vector (representing a "tape" of the turing machine) is pushed into a larger, two-dimensional vector that contains all of the tape inputs. If a user input is encountered that is not reflected in the input alphabet provided in the machine description, the input is disregarded and the user is notified that the input wasn't accepted. The machine will still continue to read input after throwing out an invalid string.

Simulation:

To simulate the Turing machine after both the machine description and user inputs are read and stored, the tapes and Machine are passed into a "Processor" class. For each input tape, the Processor class initializes the Turing machine, places the reading head at the beginning of the tape, and then proceeds to simulate the execution of the machine based on the transition rules provided in the machine description. If the machine cannot make any more transitions for the current state and tape symbol, it automatically sets the current state to the reject state. After each transition, the program checks to see whether the current state is either the accept or reject states. If so, it breaks out of the transition loop, notifies the user, and continues with the next tape input. If there are no more tape inputs, the program exits.

Functionality and Implementation of Designed TMs

Unary Addition

This machine accepts a string $a^ib^jc^k$ such that i+j=k and k > 0. The input file has seven input strings, of which the machine accepts the first four and rejects the last three.

INPUT	OUTPUT
a,b,c,c	ACCEPT
a,a,b,b,c,c,c,c	ACCEPT
b,b,c,c	ACCEPT
a,c	ACCEPT
a,a,b,c	REJECT
	REJECT
c	REJECT

Unary Subtraction

This machine accepts a string $a^i b^j c^k$ such that i-j=k and i > 0. The input file has ten input strings, of which the machine accepts the first four and rejects the last six.

INPUT	OUTPUT
a,a,b,c	ACCEPT
a,a,a,a,a,b,b,c,c,c	ACCEPT
a,c	ACCEPT
a,b	ACCEPT
	REJECT
a	REJECT
a,b,b	REJECT
a,b,c	REJECT
b	REJECT
a,a,a,b,b,b,c	REJECT

String Reversal

This machine accepts any string of a's and b's. It outputs that string, followed by a space, followed by the reverse of the input string. The input file has four inputs, all of which accepts (all valid strings are accepted).

INPUT	OUTPUT	
a	ACCEPT	a, ,a
	ACCEPT	
a,b,b,b,a,a	ACCEPT	a,b,b,b,a,a, ,a,a,b,b,b,a
b,a,a	ACCEPT	b,a,a, ,a,a,b

Major Difficulties and Implementation Decisions

Due to the similarity of this project to previous simulation projects (in terms of the reading and storage of machine descriptions and user inputs), we experienced no major difficulties in our path to completing the project. Likewise, most major implementation decisions (such as using elements of the C++ Boost library to handle string tokenization of comma delimited input) had been made previously in past assignments.