EECS 510 final project

Tyler Oswald

1 Introduction

This project focuses on building a language that represents triangle numbers. These numbers are defined by the sum $\sum_{i=1}^{n} i$. The motivation behind computing these triangle numbers is that they are often useful in fields like discrete mathematics. For example, the number of edges in a complete graph can be found with T_{v-1}, where v is the number of vertices and T is the triangle function.

Valid strings in this language will be generated by an unrestricted grammar that accepts all unary inputs with length greater than 0. These generated strings will be computed and accepted by a two-tape Turing machine that halts for every input. The first tape will hold the input passed to the machine. The second tape will hold the result of the computation.

2 Alphabet

Since this language is represented in unary, the input alphabet is relatively simple, consisting only of 1s:

$\Sigma = \{1\}$

Additionally, we will need an alphabet to represent the tape that the Turing machine uses:

$\Gamma = \{1, X, \square\}$

Both the input and output tapes will use the same alphabet.

3 Semantics

The formal language for the triangle numbers is:

$L = \{\, 1^{\frac{n(n+1)}{2}} \mid n > 0 \,\}$

This language represents the output of the Turing machine for any given input in the domain below:

$\{\, 1^n \mid n > 0 \,\}$

Essentially, the Turing machine will map any non-empty unary input to an output string that is contained in the language of triangle numbers. Additionally, this machine produces only one correct output for each input string. For example, an input of 111 would compute 111+11+1 and produce the output string 111111, which is in the language L.

4 Grammar

The unrestricted grammar below starts with a given input string in unary and uses production rules to generate the triangle number for that input. This grammar requires several helper variables in addition to the input.

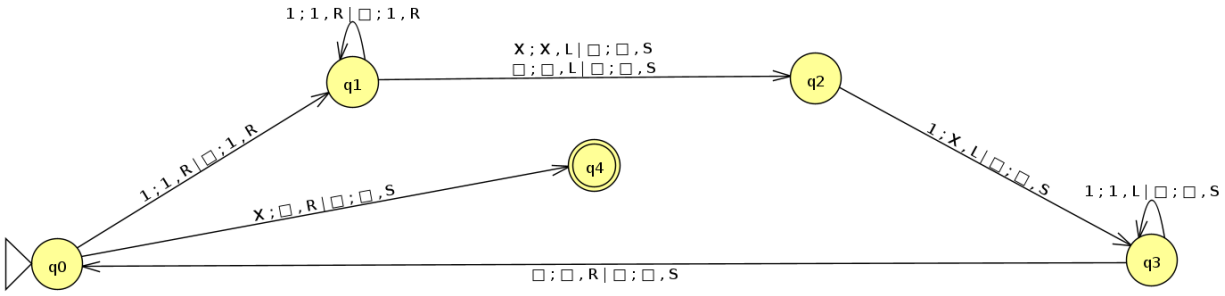
Example input for finding the triangle number of 11: _S11D

```
S1 -> B1S
1SD -> RD
_RD -> _S
1R -> R1
B1 -> 1B
BRD -> RDB
_RD -> _
_B -> 1_
1_ -> 1
```

There are several example derivations of triangle numbers using this grammar given in the grammar folder of this project.

5 Machine

This machine uses `R`, `L`, and `S` for right, left, and stay head movements respectively. Additionally, each transition follows the format (read; write, move), with tape 1 before the | and tape 2 after.
Note: This machine was drawn in JFLAP using its transition format.



6 Data structure

For this project, the machine is encoded in a file data structure. The file has one line for each different type of data stored. This means that the states, alphabet, tape alphabet, start state, accepting state, and transitions are all stored in the file for a total of 6 lines. Entries for the states and the tape alphabet are separated by whitespace on their respective lines. The alphabet, start state, and accepting state are just a single entry for each. Transitions are stored all on one line with each transition following the format below:

(Start state, Read tape 1, Read tape 2, New state, Write tape 1, Write tape 2, Move head 1, Move head 2)

Each transition is separated by whitespace. Additionally, blanks are represented as 0s in this data structure. This makes representing the tape easier in the simulator.

7 Simulator

This project also contains a Python program that simulates a two-tape Turing machine. This simulator can be used to verify the correctness of the machine and generate triangle numbers. The program reads in the states and transitions from the file data structure, then loads them into a class object. The machine is simulated by reading the tapes and picking valid transitions from a given state until reaching an accepting state or having no valid transitions available. The former reports that the string was accepted, while the latter indicates the string was rejected.