Matrix Vector Multiplication

COSC 312 HW 8b: Extra Credit TM Project

Tatiana Dehoff, Finn Sparks, Jackson Mowry

Sun Nov 17 2024

1 Project Description and Summary

We have built a matrix vector multiplication machine that accepts a matrix$_{\mathrm{n}\times\mathrm{n}}$ and a vector$_{\mathrm{n}\times 1}$, producing the result of the calculation. Numbers are represented in unary notation with a single '1' for each digit. All numbers in the input must be $\in \mathbb{W}$, that is the set of all whole numbers including 0. The machine supports any arbitrary sized matrix and vector, as long as the result of multiplication for the two operands is defined (i.e. we cannot multiply a 2×2 matrix by a 3×1 vector).

Our input space consists of {`[`, `]`, `=`, `*`, `,`, `;`, `1`} to represent the matrix/vector notation.

1.1 Members

- Tatiana Dehoff: Polished the final draft of the Turing machine. Added a dimension error check. Organized the implementation and applied coloring for clarity and better readability.
- Finn Sparks: Worked to transition from the rough draft of the machine to the polished final version.
- Jackson Mowry: Implemented an initial rough draft of a matrix vector multiplicaiton machine. Wrote this report giving a summary and overview of our turing machine project.

All members met in person to collaborate on the main portion of the machine, working together from the initial right draft into the final version that would function correctly for the most cases.

1.2 Matrix/Vector Notation

Matricies and vectors are represented in a standard notation where **,** separates digits within the same row, and **;** separates rows in a matrix or vector. [1,2,3;4,5,6;7,8,9] is represented as...

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

...and [1;2;3] is represented as...

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

To convert base_{10} digits into unary digits we create a corresponding number of 1's in the notation. Once again converting [1,2;3,4] gives us...

$$\begin{bmatrix} 1 & 11 \\ 111 & 1111 \end{bmatrix}$$

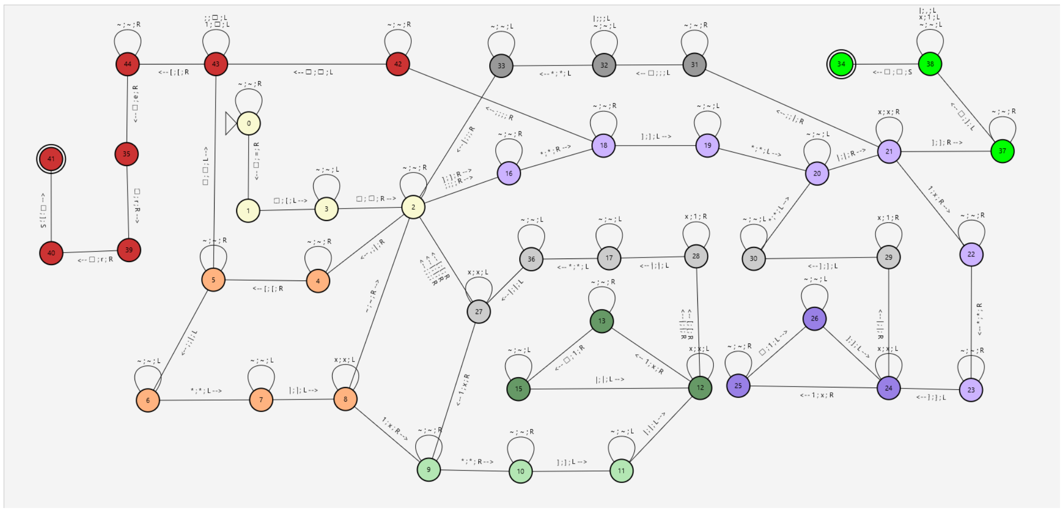

Figure 1: Our final Turing Machine

2 Turing Machine Functional Design

2.1 General Design

To begin our machine moves as far right as it can to write out `=` and `[` to prepare to write the resulting vector. After moving to the right it moves back to the start of input to begin multiplying each component in the matrix by its corresponding component in the vector. Once a single row of the matrix has been processed the machine will move into the result vector, write out a `;` to indicate the end of a digit, before finally resetting to the next row of the matrix. Once the machine reaches the end of the matrix (indicated by a `]`) it will traverse the far right and close off the result vector with a corresponding `]`.

2.2 Pale Yellow

Pale yellow represents our start state, which is non-reentrant. The machine will move to the far right, write a `=` and `[`, then move back to the left of the tape input. This portion of the machine sets up the tape to begin writing the answer at the other end of the tape.

2.3 Pale Orange

Once the machine has set up the tape to begin writing the answer it starts to multiply the corresponding componenets of the matrix and vector together. The first step in this process is to identify which components we need to multiply. The machine accomplishes this by first marking off a comma in the matrix, then moving to the vector and finding the first comma that has not been marked off. Now that the component has been identified we can move on to unary multiplication by either consuming a 1 then moving to the light green stage, or if we're already looking at a `0` we just move to the next component of the matrix.

2.4 Light Green

Now that we are performing multiplication we can move to the right, looking for a `*` to identify the delimiter between the matrix and vector. Then we move to the end of the vector, turning around when we hit the end, then searching backwards for the `|` symbol. This allows us to always work on the last element that hasn't been crossed off, as opposed to starting from the begining which would be much more complicated to determine which element in the vector is being worked on. The machine will then break out of this loop once it finds the `|`, moving to the dark green portion of the machine to write out a `1` in the result vector.

2.5 Dark Green

Now we can start writing out digits in the answer. Since we were just on top of a `|` symbol we know the digit to work on is sitting just to the left. We work on digits from right to left, meaning we may need to skip over some `x` characters in order to find the `1` that has not been crossed off. This is accomplished by looping on `x` and moving to the left. This loop is broken once we find a `1`, the machine then crosses this `1` off, replacing it with an `x`. Once a digit has been crossed off we need to write a `1` in the answer, which is accomplished by moving to the right until finding a blank, then writing a 1. The machine can then move back until finding a `|`, entering the beginning of the dark green loop.

The machine is only able to leave this dark green loop once it is done proccessing an entire component in the vector. This is indicated when we fine either another `|` or a `[` (indicating the start of the vector).

2.6 Gray

Once the machine is done processing the appropriate component of the vector it will enter the gray set of states. The machine will now be at the left end of the digit, which means it can revert `x` to `1` while moving to the right, breaking out of this loop once it finds a `|` delimiter. This set of gray states then moves us to the left (out of the vector), and back into the matrix. This left moving loop is broken once we find a pipe, indicated that we are back at the component of the matrix we are working on. We then scan through all the `x` characters of this component to find the beginning. If we're done with this component of the matrix the machine will enter back into the pale orange loop described above.

2.7 Light Purple

As we approach the end of a row in the vector our machine expects to see either a `;` or `]`. This indicates to the machine that it should now find the last component of the vector, as that is the corresponding element. It accomplishes this by moving to the right of the vector, and then coming back into the matrix looking for a pipe to indicate we are back the correct digit. After moving right off the pipe it should now be looking at either a `1` or an `x`, we consume any `x` characters until the machine is back on top of a `1`. This `1` is then marked off in the matrix, and the machine heads to the vector.

2.7.1 Green

As light purple is responsible for handling the last component in each row of the matrix, it must also be responsible for handling the accept state. As light purple is working on the last component in a row it is either looing for another 1 to multiply, or a `]`. Once it finds a `]` it knows the last element has been processed. The last step is to simply move to the far right, and place a `]` to close off the result vector and complete the calculation!

2.8 Dark Purple

Working on the last component of the vector is treated differently, and we work on a digit from the right-hand side. Marking off each digit and copying it into the result vector one at a time. Once the first pass of the component has completed the machine heads to a short gray path.

2.8.1 Gray

There is a short path here to revert `x` to `1` in the vector, before dumping us back into the matrix ready to process another digit. This then loops with the logic described above.

2.9 Dark Gray

Once an entire row has been processed in the matrix the machine now needs to head to the right and write a `;` to mark the end of a row in the vector. This simply loops on any non-blank symbol while moving to the right. Once we find a blank we write a `;` and then perform the same loop consuming anything while moving to the left. The machine finally ends up back in the matrix ready to multiply another row by the vector. At this point the machine has been nearly fully explored, and it will continue looping until it has processed every digit, refer back to the light purple section for the accept state.

2.10 Red

The red section of our Turing machine is responsible for handling dimension mismatches in the matrix-vector multiplication process. When the machine detects incompatible dimensions, it moves all the way to the right and erases the partially computed result, replacing it with 'err.' This route also results in the accept state.

2.11 Accept State

The accept state is reached when the machine has either successfully completed the computation or identified a dimension mismatch. For the successful computation, refer to the subsection **Green** above, as this is handled within the light purple path. The error route is described in the **Red** subsection.

3 How to Execute our Machine

The machine expects a matrix of dimensions n×n followed by `*` and then a vector of dimension n×1. All input values must be $\in \mathbb{W}$, in unary notation.

Base 10	Unary
0	
1	1
2	11
3	111
4	1111
5	11111
6	111111
7	1111111
8	11111111
9	111111111
10	1111111111

Each matrix/vector should be surround on either end with `[` and `]`, and the matrix should be separated from the vector with a `*` symbol. Componenets in the same row of the matrix are separated by `,` while rows of a matrix (and vector) are separated by `;`. Theoretically any dimension should be supported, however sizes larger than 4×4 or 5×5 tend to run very slowly.

3.1 Accepted Input Values

3.1.1 Example 1

Valid input: `[1,11;111,1111]*[11;111]`
Expected output: `=[11111111;111111111111111111]`
or in base 10…

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}2\\3\end{bmatrix}=\begin{bmatrix}8\\18\end{bmatrix}$$

3.1.2 Example 2

Input with 0 present as a component in the matrix
Valid input: `[,1;11,111]*[11;111]`
Expected output: `=[111;1111111111111]`
or in base 10…

$$\begin{bmatrix}0 & 1\\2 & 3\end{bmatrix}\begin{bmatrix}2\\3\end{bmatrix}=\begin{bmatrix}3\\13\end{bmatrix}$$

3.1.3 Example 3

Input with 0 present as a component in the vector
Valid input: `[1,1;11,111]*[;111]`
Expected output: `=[111;111111111]`
or in base 10…

$$\begin{bmatrix}1 & 1\\2 & 3\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix}=\begin{bmatrix}3\\9\end{bmatrix}$$

3.1.4 Example 4

Input with a 0 matrix
Valid input: `[,;,]*[1111;11111]`
Expected output: `=[;]`
or in base 10…

$$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\begin{bmatrix}4\\5\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$$

3.1.5 Example 5

Input with a 0 vector
Valid input: `[1,11;111,1111]*[;]`
Expected output: `=[;]`
or in base 10...

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$$

3.1.6 Example 6

Input with both 0 matrix and 0 vector
Valid input: `[,;,]*[;]`
Expected output: `=[;]`
or in base 10...

$$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$$

3.1.7 Example 7

Input of dimension 3
Valid input: `[1,11,111;1,11,111;1,11,111]*[1;11;111]`
Expected output: `=[11111111111111;11111111111111;11111111111111]`
or in base 10...

$$\begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}14\\14\\14\end{bmatrix}$$

3.1.8 Example 8

Mismatching dimensions
Valid input: `[1,11;111,1111]*[1;11;111]`
Expected output: `=[err]`
or in base 10...

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = [err]$$