Due 5/15 (Friday of finals week)

Construct a Turing machine that adds two ternary integers (base 3). An input will be of the form X#Y, where X and Y are elements of $\{0, 1, 2\}^+$. In particular, $X = x_n x_{n-1} \ldots x_1 x_0$, and $Y = y_m y_{m-1} \ldots y_1 y_0$, with x_i, y_i in {0, 1, 2}, $X = (x_n \times 3^n) + (x_{n-1} \times 3^{n-1}) + \ldots + (x_1 \times 3^1) + (x_0 \times 3^0)$, and $Y = (y_m \times 3^m) + (y_{m-1} \times 3^{m-1}) + \ldots + (y_1 \times 3^1) + (y_0 \times 3^0)$. Your Turing machine must be a single tape, one way infinite, deterministic Turing machine. When the Turing machine completes, the tape should contain Z, where Z = X + Y. You do not need to delete X#Y from the tape, you can simply position the Turing machines read/write head at the beginning of Z. For this assignment you will probably want to use blocks (subroutines in JFLAP) to build your Turing machine. You can also make use of the S directive for read/write head motion (L – left, R – right, S – stay).
Your Turing machine cannot make use of the blank spaces to the left of the input string. JFLAP has some unhappiness with filenames containing special characters and I don't know all the symbols that cause problems (I stick with alphanumeric and the underscore symbol, and have had problems with $, #, and the blank space in block file names). When you create a block, I would suggest you create it as a separate file, test it, and then insert it into your main program (or a block that goes into your main program). JFLAP seems to only keeps a single copy of each block in the main file, so if you insert a block multiple times into your program, and then edit and save one of the copies from your main program, then all copies of the blocks within your main program will be updated. I've heard from students that if you are editing a block from within you main program and you save the block before closing it, it will overwrite your file with the block you are editing (you need to exit block edit mode prior to saving). Keep backup copies of all of your file (often). You can do a save as while editing a block to save the block as a separate file.

It took me about three hours to implement my version of the program, and then an additional hour to test it. For my final test I generated 1000 random strings of the form X#Y where X and Y have length between 4 and 10 symbols from {0, 1, 2}. I then ran my program against the strings and verified that the result of adding X and Y was correct (I wrote a java program to do the verification).

Below is some additional information about my implementation.

My Turing machine uses 8 blocks. The blocks perform the following actions.

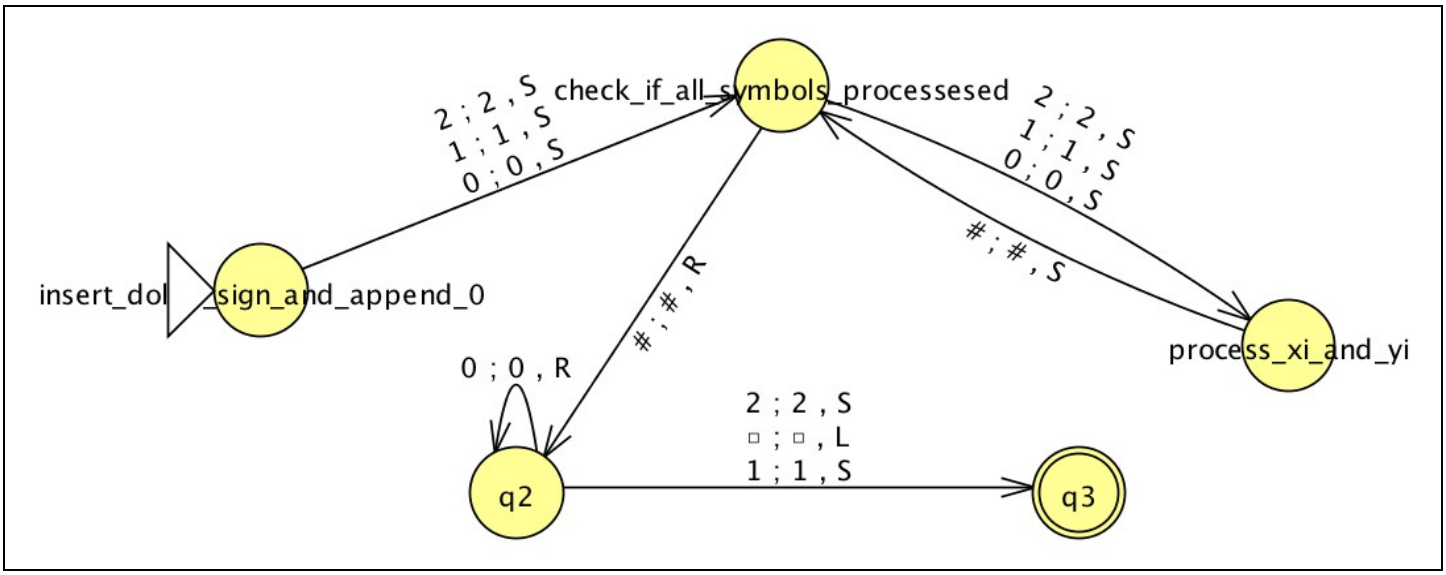
- insert_dollar_sign_and_append_0 – block that inserts a “$” at the left end of the input, shifting all of the input to the right one position, and appends a “#0” at the right end of the string
  - The block has 9 states and rewinds the tape leaving the read/write head under the first symbol of X#Y.
  - The “#” appended to the input is to mark where Z (Z = X + Y) begins.
  - The “0” appended to the input is the carryover from the addition of the previous digits of X and Y (the initial carryover is 0).
- insert_0 – block that inserts a 0 at the current location of the read/write head, shifting all symbols to the right one position
  - The block has 6 states.
  - Used when inserting the carryover after adding x_i and y_i (and the carryover from the previous digits of X and Y).
- insert_1 – block that inserts a 1 at the current location of the read/write head, shifting all symbols to the right one position.
  - The block has 6 states.
  - Used when inserting the carryover after adding x_i and y_i (and the carryover from the previous digits of X and Y).
- check_if_all_symbols_processed – block to determine all digits of X and Y have been

processed.
  - The block has 4 states.
- find_next_xi – block that finds the next unprocessed digit of X
  - The block has 3 states.
- find_next_yi – block that finds the next unprocessed digit of Y
  - The block has 4 states.
- find_next_ci – block that finds the carryover from the addition of the previous digits of X and Y
  - The block has 2 states.
- process_xi_and_yi – block that adds x_i and y_i and the carryover from previous operations
  - This block has 8 states and uses blocks insert_0, insert_1, find_next_xi, find_next_yi (3 copies), and find_next_ci (5 copies).
  - This does the vast majority of the work.

I found that using the find_next_xi, find_next_yi, and find_next_ci to position the read/write head at the next unprocessed digit of X and Y and the carryover from the previous addition allowed me to keep the overall number of states relatively small.

My Turing machine has a tape alphabet of {0, 1, 2, x, #, $, blank space}. The “x” is used to mark symbols of X and Y as having been processed.

Below is an image of my main program. Once my program marks the left end of the tape and appends the “#0” to the right end of the input string, it simply calls process_xi_and_yi until all of the digits of X and Y have been processed.



E-mail the JFLAP file to me (david.garrison@binghamton.edu) by 11:59:59.999pm on the date due. The filename must be your last name followed by “_p7.jff” (as an example, my filename would be “garrison_p7.jff”).