Foundations of Computer Science - Fall 2024
Implementation of a 1-Tape Universal Turing Machine

By Ertug Umsur & Joshua Espinoza Diaz

Introduction:

The Universal Turing Machine (UTM) is a theoretical construct introduced by Alan Turing in 1936, and was used to demonstrate the foundational concept of computational universality. A UTM is characterized by its capability to simulate the program of any Turing machine (including itself) given an encoded description of its states, alphabet, transition functions, starting state, accepting state and rejecting state. Our project focuses on building the framework to implement a theoretical single-tape Universal Turing Machine.

This document is divided into two sections. Section 1 will cover the encoding scheme that we used to transform the formal definition of any Turing Machine and an input string, into a single string that will function as the input for our UTM; it also covers our implementation of an encoder program written in python.

Section 2 focuses on the implementation of such machine, starting with a brief explainer of our approach on how we developed the algorithm; then covering a description of the "helper" functions that we used for the final implementation of the UTM, which we refer to as "gadgets", and finally the explanation of how the algorithm was implemented as a finite automaton.

Table of Contents:

Part I: Encoding

Turing Machine (TM) encoding
Notation:
We define a Turing Machine M as a 7-item tuple that contains a set of states (Q), an input alphabet ($\mathbf{\Sigma}$), a tape alphabet (Γ), a set of transitions (δ), the start state (q_s), the accepting state (q_{acc}), and the rejecting state (q_{rej}).
We denote <M> as the "raw" encoding of M into a bitstring that only contains the information from the definition of M.
We denote [Mw] as the "complete" encoding of M and an input string w into a bitstring that contains <M>, the encoding of w (<w>) and some extra information that will be defined later on. In other words, [Mw] is the encoding of M and w that our Universal Turing Machine will accept as input to simulate M.

To encode M into <M>, it will be necessary to first translate the items in the definition of M into a bitstring form. The process to do so is as follows:

- Q = $\{q_1, q_2, q_3, \dots q_n\}$ is encoded by identifying each state q_k as k, and converting k into its unary form (1 = 1, 2 = 11, 3 = 111, etc., for all n states). We then take all the unary representations of the states in Q and assemble them in a single bitstring with a single 0 in between states to differentiate them: $\{q_1, q_2, q_3, \dots q_n\} = 1\,0\,11\,0\,111\dots0\,1^n$. We call this final bitstring as the encoding of Q (<Q>). In Q, the states q_1, q_2, q_3, must correspond to q_s, q_{acc}, q_{rej} respectively.
- $\mathbf{\Sigma}$ = $(s_1, s_2, s_3, \dots, s_n)$ is encoded by assigning each of the symbols in the alphabet to an index ($s_1 = 1, s_2 = 2, s_3 = 3, \dots, s_n = n$), and then translating those indexes into their unary form. $\mathbf{\Sigma}$ is then assembled by putting together the unary representations of all the symbols in $\mathbf{\Sigma}$ into a bitstring that will use 0 to separate one symbol from another one: $\{s_1, s_2, s_3, \dots s_n\} = 1\,0\,11\,0\,111\dots0\,1^n$. This final bitstring will be called the encoding of $\mathbf{\Sigma}$ (<$\mathbf{\Sigma}$>).
- Γ = $\{s_1, \dots s_n, \gamma_1, \gamma_2, \dots \gamma_k\}$ is encoded with the same process as the encoding of $\mathbf{\Sigma}$. All of the symbols in Γ are assigned an index, keeping the symbols from $\mathbf{\Sigma}$ (since $\mathbf{\Sigma} \subseteq$ Γ) at the beginning of the set Γ with the same indexes used to encode $\mathbf{\Sigma}$, while the unique tape symbols will be indexed starting from $n + 1$ up to $n + k$. The indexes are then represented in their unary form, and the final bistring for Γ is assembled by putting together all the unary representations of the symbols in Γ, and using 0 to separate each individual symbol: $\{s_1, s_2, \dots s_n, \gamma_1, \gamma_2, \dots \gamma_k\} = 1\,0\,11\dots0\,1^n\,0\,1^{n+1}\,0\dots\,1^{n+k}$. This bitstring will be called as the encoding of Γ (<Γ>).
- In a set of transitions δ = $\{d_1, d_2, d_3, \dots d_n\}$, is d_k is defined as a 5-item tuple of the form $\{p, a, q, b, \pm 1\}$, where p represents the current state, a represents the current symbol in the tape, q represents the next state from the transition, b represents the symbol to

replace a with, and $\pm\ 1$ represents the movement of the tape head (left or right). To encode δ, all d_k transitions in δ are encoded by representing $p,\ a,\ q$ and b with their encodings generated for the encoding of Q and Γ, and a $\pm\ 1$ is represented in unary form as a 1 if the movement is to the right, or 2 if the movement is to the left. These encodings are assembled into a single bistring with a 0 separating each element of the tuple to form an encoded transition of d_k (<d_k>). Then, δ is assembled by putting together all the bit strings representing the transitions in δ, and using 00 to separate each transition: δ = $\{d_1,\ d_2,\ \ldots\ d_n\}\ =\ \ < d_1 >\ 00\ < d_2 >\ 00 \ldots\ 00 < d_n\ >$.

- The start state (q_s), accepting state (q_{acc}), and rejecting state (q_{rej}) are encoded by representing them with their corresponding encodings generated for <Q>.

With all the elements in M encoded, the final encoding <M> is formed by taking the encoding of its elements and separating them with 00, except for the encoding of δ, which will have a 000 instead of a 00 at the beginning and end of it to: a) Differentiate it from the rest of the elements in M (since we will check δ constantly while simulating M) and b) Avoid confusion between different transitions in δ (which are separated by 00) and the other elements of M. Following these rules, M is encoded in <M> as follows:

<M> = <Q> 00 <**Σ**> 00 <Γ> 000 <δ> 000 <q_s> 00 <q_{acc}> 00 <q_{rej}>

To encode M and the input string w into a bitstring [Mw] that can be used to simulate M in our Universal Turing Machine (UTM), we need to add a couple of things to <M>:

- First, we add a buffer of n 0's at the beginning of the bitstring <M>; where n is the largest index for a state in Q that was used when generating <Q> and we add the sequence 0000 after the buffer. The reasoning behind this added sequence will be described later on for our explanation of our UTM implementation. We end by flipping the first bit of the buffer to a 1 to declare our starting state (by convention q_s has been defined as 1 in unary).
- Secondly, we add at the end of <M>, the sequence 0000 (this will also be explained later on) and we add the encoding of the string w that will be the input of M to simulate.
    - The encoding of w (<w>) is obtained by representing each of the symbols in the string with their index representation in unary form assigned with the encoding of the input alphabet **Σ**, and assembling all the symbols together with a 0 in between symbols to differentiate them.

This generates [Mw], which is the encoding of M and w that will be accepted by the UTM as follows:

[Mw] = $1\ \ 0^{n-1}\ \ 0000$ <M> 0000 <w>

TM Encoding implementation in python

GitHub Repo of our implementation: https://github.com/SirTrap/FOCS_Final

How the code works: The code has a *MachineConverter* class which is initialized by taking a string input with a dictionary format (the format used in the class to represent Turing Machines).

So, copying any dictionaries we worked on in the class before would work. Then, the *init* function converts this string into a dictionary using the ast library and the function ast.literal_eval which evaluates the string without the quotes. Then, all of the Turing Machine information like “states” and “alphabet” are saved in variables.

```
self.states = self.diction["states"]
self.alphabet = self.diction["alphabet"]
self.tape_alphabet = self.diction["tape_alphabet"]
self.start = self.diction["start"]
self.accept = self.diction["accept"]
self.reject = self.diction["reject"]
self.delta = self.diction["delta"]
```

The class also has a second function (*convert2string*) which converts the dictionary into a unary string format following the encoding described before for our implementation. The function first assigns numbers to the items for the first 3 tuples that describe M (states, input alphabet and tape alphabet in a dictionary and then it creates unary strings for each tuple using a for loop that starts from 1 and ends at *len(tupleName)*.

```
index = 1
for item in tupleName:
    if item not in items_dict:
        items_dict[item] = index
        print(item)
        index += 1

uItems = ""
for i in range(len(self.states)):
    uItems += "1"*(i+1)
    uItems += "0"
uItems = uItems[:-1]
```

Note: In our code implementation, the start, accept and reject states are defined as 1, 2 and 3 following our state indexing convention explained for the encoding of Q. Therefore, the variable *indexs* (*index* variable used for the states) starts at 4 instead of 1.

This creates a unary string in the format: $1\ 0\ 11...\ 0\ 1^n$ (where n is the size of the tuple). For the delta transition functions; first, all of the transition functions are converted into their corresponding index generated for the previous steps in our encoding, using the dictionaries created earlier to do so.

```
mod_delta = []
```

```
for (starts, letter, ends, tletter, add) in self.delta:
    mod_delta.append((
                        States_dict[starts],
                        Letters_dict[letter],
                        States_dict[ends],
                        Letters_dict[tletter],
                        1 if add == 1 else 2
                      ))
```

Then, the indexes are switched into strings with their unary representation. At the end, the machine unary string is formed by adding all unary strings for all the elements of M, and finally, the starting state and the string input unary strings are also created and added to the machine string. The function then outputs the finalized string.

```
ucs = states_dict[cs] * "1" + (len(self.states) - states_dict[cs]) * "0"

encoding = ucs + "0000" + ustates + "00" + ualphabet + "00" +
           utapealphabet + "000" + udelta + "000" + "1001100111" +
           "0000" + ustring

return encoding
```

Part II: Universal Turing Machine Implementation

UTM as a 3-tape machine

The basic algorithm for a UTM is a 3-Tape Turing Machine. In this implementation, for a Turing Machine M, one of the tapes is reserved to keep track of the current state of M while it is being simulated, another tape is used to hold the encoding of M for reference during the simulation, and the last tape is used to represent the simulated tape of M. For our implementation, we decided to allocate each tape in that specific order; the first tape to keep track of the state of M, the second tape to hold the encoding of M (<M>), and the third tape to simulate the tape of M (this setup is illustrated in Image 1).

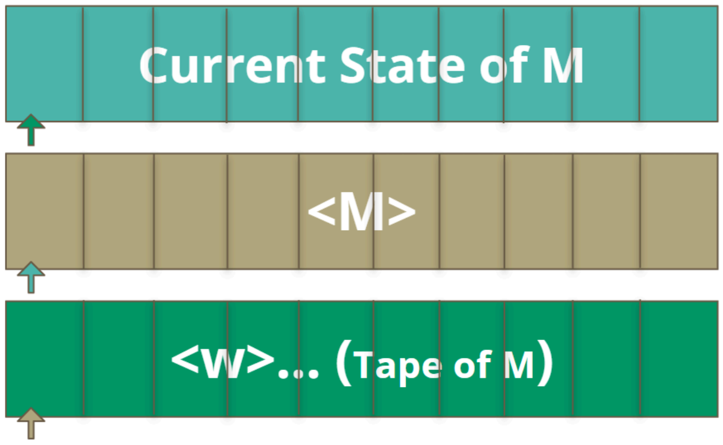

Image 1: *3-Tape UTM representation*

The algorithm for this UTM implementation can be described as follows:

1. To determine the next transition that M should take:
    a. Move the head tape of tape 1 to the beginning of the tape.
    b. Move the tape head of tape 2 to the next transition in M (the first transition if this is the first iteration of the loop).
    c. Compare the state in tape 1 with the state in the current transition of M. If they are different, this is not the correct transition, re-run step 1. If they are the same, this might be the correct transition, continue to step 2.
2. Now that the current state and the state in the current transition are equal:
    a. Move the tape head of tape 2 to the next item in the current transition (symbol a).
    b. With the tape pointer of tape 3 over a symbol in the tape of M (over the first symbol of the input string w if this is the first iteration), Compare the symbols in tape 2 and tape 3. If they are not the same, this is not the correct transition, return to step 1. If they are the same, this is the correct transition (current state and symbol match), proceed to step 3.
3. Simulate the change of states in M:
    a. Move the tape head in tape 2 to the next item of the current transition (state q), and replace the contents of tape 1 with the state described in tape 2 (move from state p to state q).

    b. Move the head in tape 2 to the next item of the current transition (symbol a), and replace the symbol in tape 3 with the symbol in tape 2 (change symbol a to symbol b).
    c. Move the head in tape 2 to the next item of the current transition (Movement of the tape head in the tape of M), and move the pointer in tape 3 based on the contents of tape 2.
4. Check the current state in tape 1.
    a. Return the tape head in tape 1 to the beginning of the tape and read the current state. If the current state is the accepting state of M, accept the string. If the current state is the rejecting state of M, reject the state. If the state is any other state, return to step 1 and continue simulating M.

3-Tape UTM to 1-Tape UTM transformation

To replicate the behaviour of a 3-Tape Turing Machine in a 1-Tape TM, we needed to find a way to: a) Combine the 3 tapes into a single tape while keeping the section corresponding to each of the original tapes recognizable; and b) Replicate the behaviour of multiple pointers keeping track of their respective tapes.

The solution for the first requirement was introduced at the end of the encoding section of this document. To keep the three tapes recognizable between each other, we concatenate the three tapes (where tape 1 will also be referred as the buffer introduced during the encoding process) and add a special symbol to differentiate them. Our UTM has an input alphabet of 0s and 1s, where the 1's represent information, and the 0's are used to form differentiation symbols: 0, to separate elements in a set; 00 to separate the elements in M and the transitions in the transition function δ, and 000 to differentiate δ from the other items from M (added at the beginning and end of δ). Since our objective is to introduce a new differentiation symbol to separate the three tapes between each other, we use the sequence 0000 as a new symbol to achieve this purpose. This is how during the explanation of the encoding, we obtained the definition of [Mw] from <M> and <w>:

[Mw] = $1\ 0^{n-1}\ 0000$ <M> 0000 <w>

Here it is important to point out that the first tape is expressed initially as a sequence of n 0's (or the "buffer") so that it can hold up to the state in M encoded with the largest index n.

To achieve the second requirement, we will introduce the notion of pointers; which are special symbols introduced in each of the tapes to "represent" what each of the tape heads in our 3-Tape TM would perform.

Note: for TMs, a tape head and a tape pointer refer to the same thing, but for this document, we will treat them as different concepts that refer to the conceptual "head" that is used in a TM, and the symbolic "pointer" that will represent that concept in when merging multiple tapes into a single one.

Although the pointers for the three tapes could be represented with the same symbol from the tape alphabet of our UTM; for practical reasons, we will make these symbols different. This will allow us to easily identify each of the tapes by the pointer contained inside it, which will prove useful when traversing the UTM tape between sections to replicate the behaviour of a 3-Tape UTM. The introduction of the pointers is illustrated in Image 2.

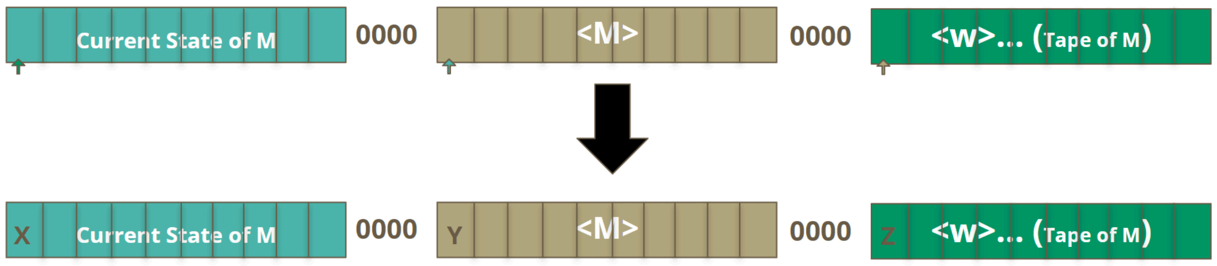

Image 2: *Tape heads to pointers representation.*

Gadgets

The implementation of a UTM requires several sequences of states that will perform the same task multiple times. This can be complex to understand at a large scale like the one needed for a 1-Tape UTM. To simplify this issue, we will introduce the notion of a Gadget.
A gadget is a finite automaton that describes a sequence of actions performed by a Turing Machine, but with no accepting or rejecting state, but rather an ending state (or states) that can be connected to a different section of a TM. The start transition is also labeled with the associated TM transitions that the gadget "accepts" as inputs when it is connected to a different section of a TM. In essence, a gadget is a function that contains a specific action that a complete TM can reuse at different points in its execution.

<u>Notation</u>:

- **A,B** refer to pointers found in the tape of our UTM. The action that a gadget is set to perform is over the "information" of the specified pointer (rather than the pointer itself). A pointer A or B is set to "contain" the "information" located to its right side. In our UTM with input alphabet {0,1}, "information" is represented as a sequence of 1s used to encode something in unary form.
- **a,b** refer to symbols in the tape of our UTM. The action that a gadget is set to perform applies to the symbol itself. This is useful to denote actions that will be applied to the "pointer" itself rather than the information it contains.
- **P** refers to the placeholder symbol that will be used to perform an action. These act in a similar way as temporary registers for a register machine in the sense that the placeholder symbol is used to keep track of the progress of an action, and must be returned to its original state (before being changed to P) at the end of the execution of a gadget so the information is not altered.
- **C** refers to a tape symbol that is different from the ones used to define a gadget (in our case, any other symbol that is not A, B, a, b, P, 1 or 0). When a gadget is implemented in a TM, the transitions marked with the symbol C must be replaced with any of the symbols in the machine's tape that are not: a) Part of the input alphabet, b) The place holder symbol (P) or c) The input symbols of a gadget. For instance, for a TM with unique tape symbols x, y and z, if a gadget is set to perform an action over x and y (x and y are inputs to the gadget), every transition that involves a C must be replaced by z.

We will now present and explain the gadgets developed for our UTM implementation. For reference, in the gadget diagrams, a gray state represents an ending state of the gadget, and a transition arrow colored orange has the same behaviour of any other transition, but is used to differentiate between overlapping transition arrows in our diagrams.

Gadget 1: Copy Machine.

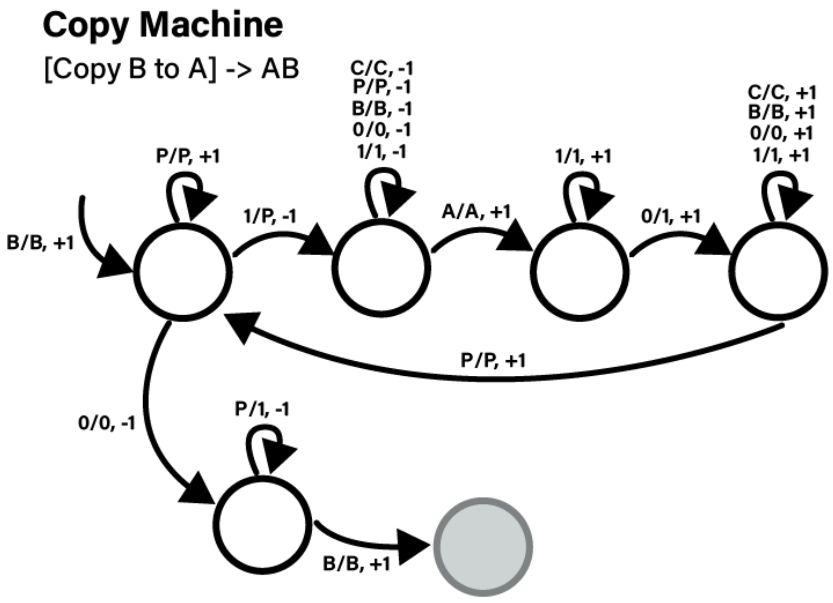

Image 3: *Diagram of the Copy Machine*.

This gadget copies the content of a pointer B to the pointer A by replacing a sequence of 0s to the left of A with the information from pointer B. This gadget assumes: a) Pointer A starts empty (there is no sequence of 1's next to it but rather a sequence of 0's), b) That the pointer A has enough 0's to its right to hold the information copied from B, and c) Pointer A is located to the left of pointer B in the TM's tape.
To do this, the gadget starts when the tape head of the TM is exactly over the symbol for pointer B. Once it identifies pointer B, the gadget looks for the next bit of information with a 1 located next to B (if it is the first iteration, the first 1 should be right next to B), and marks it with a P. it then moves all the way over to the left of the tape until if finds pointer A. After it finds it, the gadget will skip over the previously copied bits of information until it reaches a bit with no information (0), and changes it to a 1 to "copy" the bit from pointer B. After this is completed, the gadget will return to the last bit of information (P) copied from pointer B and loop back to continue copying the information. The loop ends when the information in pointer B ends (there is a 0 that interrupts the sequence of 1s) and then it returns from the last copied bit to the pointer B, changing the placeholder symbols back to 1's to restore the information in the tape.

Note: This is the machine that will be used to copy the next stage for M into the first tape of the UTM, and the reason behind the deliberate decision to make the representation of tape 1 a "buffer" of n 0's, allowing it to hold up to the state with the largest index n assigned from the encoding of M.

Gadget 2: Compare Machine.

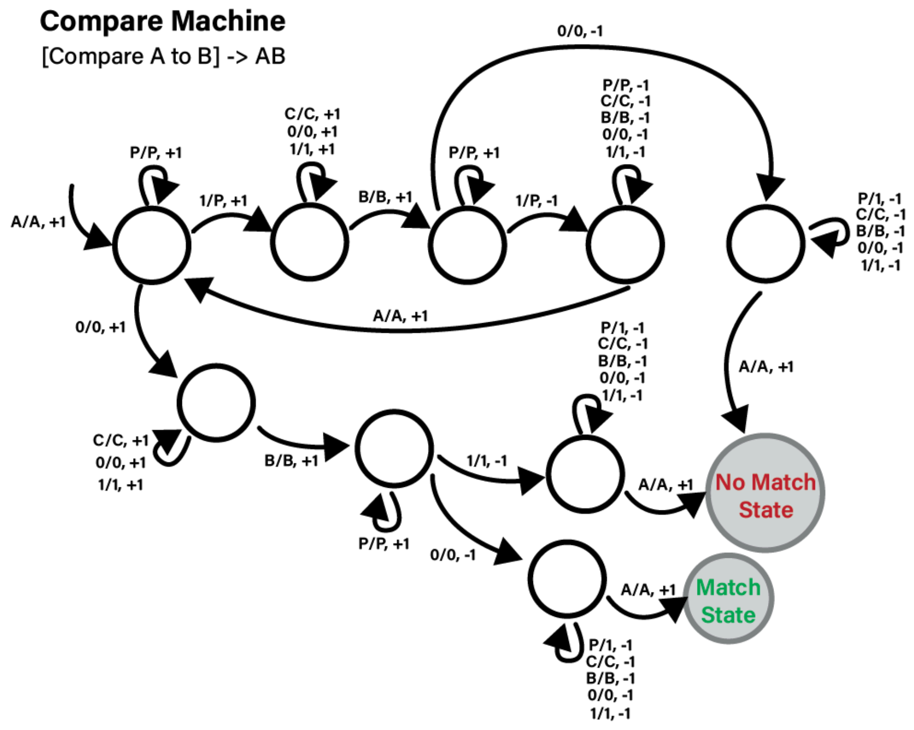

***Image 4:** Diagram of the Compare Machine*.

This gadget compares the contents of pointer A with the contents of pointer B and ends in a final state if the information in the two pointers is the same, or in a different final state if the information differs. This gadget assumes: a) Pointer A is to the left of pointer B, b) The information in pointer A and B ends with a 0 or a space symbol (-)
The gadget starts exactly when the tape head of the TM is over A. It then skips all of the previously compared items and finds the next bit of information to compare. Once it finds it, it changes it to a P and moves to the right side of the tape until it finds the pointer B. It then skips over the previously compared information in pointer B and finds for the next bit of data to compare. Once it finds it, it switches it to a P and returns back to the left until it finds pointer A and loops back to compare the rest of the digits. The loop can be interrupted in two ways. Once it finds the end of the information from pointer B, it moves back to the left changing any P symbols back to their original state until it finds pointer A and ends in the No-Match state.
The other way to stop the loop is when the gadget finds the end of the information from pointer A. At this point the gadget goes over to pointer B and checks what the last bit after the previously compared bit of information marked with P indicates. If the next bit is a 1, that means that pointer B still has more information, so it returns to pointer A reverting all the P's to their original state and ends in the No-Match state. If the next bit is a 0, that means pointer A and pointer B have the exact same length and thus, contain the same information. So, it moves back to the left reverting any P's to their original state until it reaches pointer A and ends in the Match state.

Gadget 3: Insert Machine.

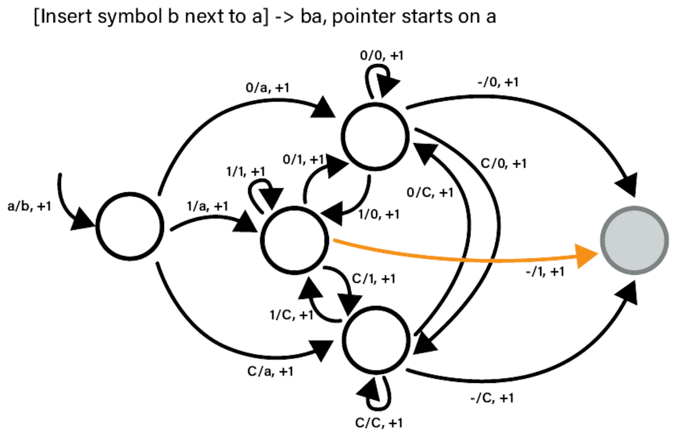

Image 5: *Diagram of the Insert Machine*.

This gadget inserts a symbol b in the position of symbol a and moves the rest of the tape 1 cell to the right to give the space to symbol b. This gadget assumes: That the pointer starts exactly over symbol a.
The gadget starts by replacing the symbol a with symbol b, then moves 1 cell to the right. Depending on what the next symbol is (0, 1 or C) it goes to a specific state, this will allow the gadget to “remember” the last symbol it replaced. The first symbol is then changed by symbol a, then it moves to the right and changes the next symbol by the first symbol. This is repeated until the machine finds a space, meaning it reaches the end of the string in the tape, and replaces the space with the last copied symbol. The machine ends with the tape head at the end of the string.
Note: This machine is used to introduce the pointers to the encoded [Mw] in the UTM.

Gadget 5: Insert Content Machine.

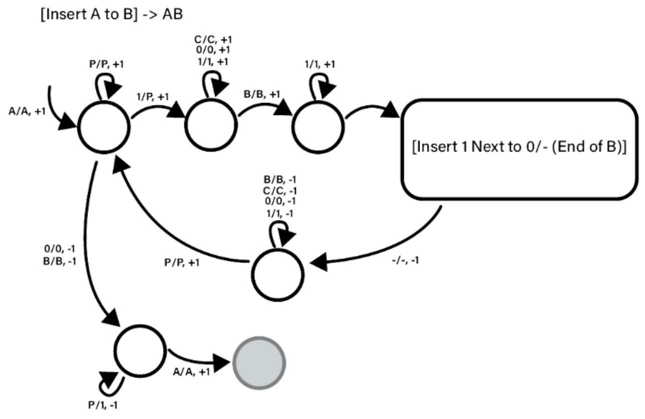

Image 6: *Diagram of the Insert Content Machine*.

This gadget inserts the content from pointer A to pointer B. This gadget assumes: a) Pointer A is left to pointer B, and b) Pointer B is empty.

The logic for this gadget is similar to the one similar to the copy machine. The sequence starts with the tape head exactly over pointer A, it then moves to the first bit of information that has not been previously copied and once it finds it, it replaces it for a P and moves over to the right until it finds pointer B. It then skips all of the information bits that have been copied before until it reaches the end of a bit (marked by a 0 if B is inside the string or a space symbol (-) if B is the last pointer of the tape string). Once in the end, it uses the previously described gadget: **Insert Machine,** and uses it to insert the bit of information at the end of pointer B. Once it finishes, it returns back to the last copied information bit in pointer A and loops back to continue inserting the information in A to B. Once it reaches the end of the information string in pointer A, the machine loops back to the start of pointer A, returning any P's to their original state. The gadget ends on pointer A.
The diagram for this gadget can be found in Image 6, which uses the gadget representation of the Insert Machine. A complete diagram incorporating the full schematic for the Insert Machine can be found in Image 7.

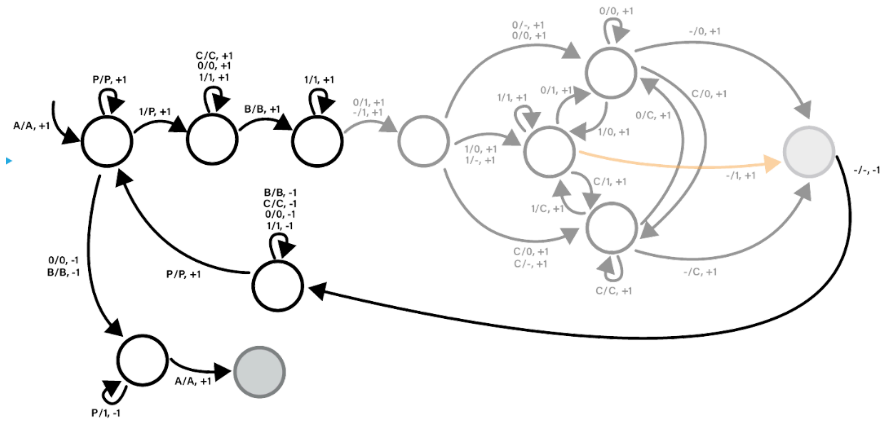

Image 7: *Extended diagram of the Insert Content Machine*.

Gadget 6: Collapse Machine.

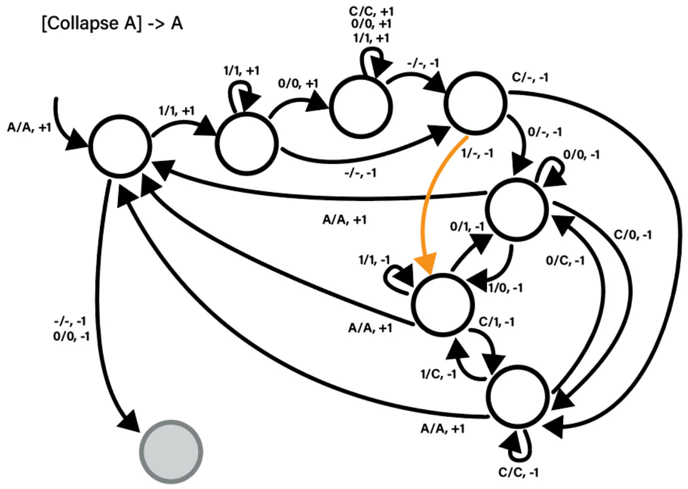

Image 10: *Diagram of the Collapse Machine*.

The collapse machine mimics the behaviour of the insert machine with the opposite effect of the latter machine. This gadget deletes the information contained in pointer A. It assumes that pointer A has information in it.
The sequence starts with the tape head over pointer A, and it moves to the right skipping all of the information bits from A. Once it finds the end of the information for pointer A, it moves over to the end of the tape and once it finds the first space symbol, it returns to the left, replacing the current symbol with the previous symbol until it reaches the pointer A. Effectively pushing the tape 1 cell to the left. This process is repeated until pointer A is empty (there is a 0 or a space next to it) and ends with the tape head in the symbol next to pointer A.

Gadget 7: Replace Machine.

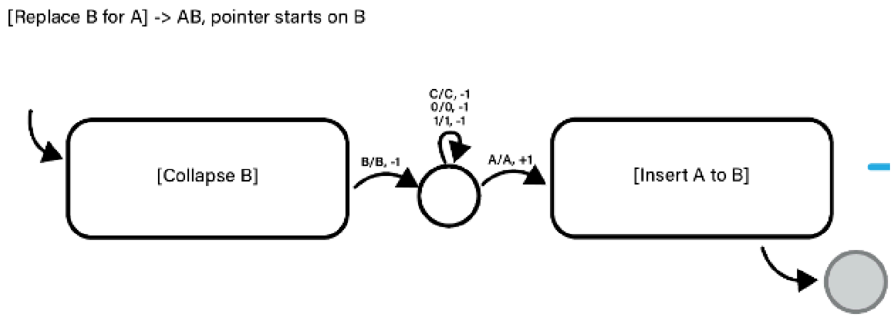

Image 11: *Diagram of the Collapse Machine*.

This gadget replaces the information in pointer B with the information from pointer A. It assumes that A is left to B and the tape head starts over pointer B.
The sequence starts by using the **Collapse Machine** to empty the contents of pointer B, then it moves back to the left until it finds pointer A and inserts the contents of pointer A into pointer B using the **Insert Content Machine**.
The simplified diagram of this gadget is shown in Image 11. An extended version of this diagram can be found in Image 12.

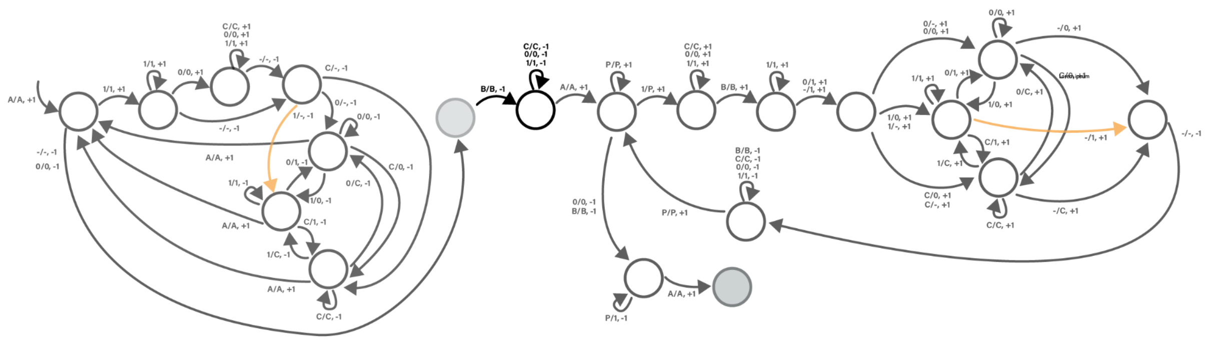

Image 12: *Extended diagram of the Collapse Machine*.

Gadget 8: Move pointer machine.

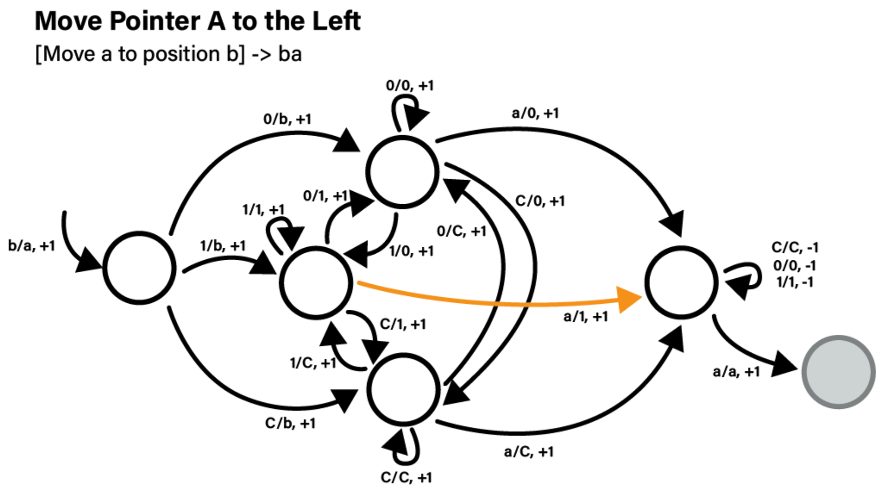

Image 13: *Diagram of the Move pointer to the left Machine*

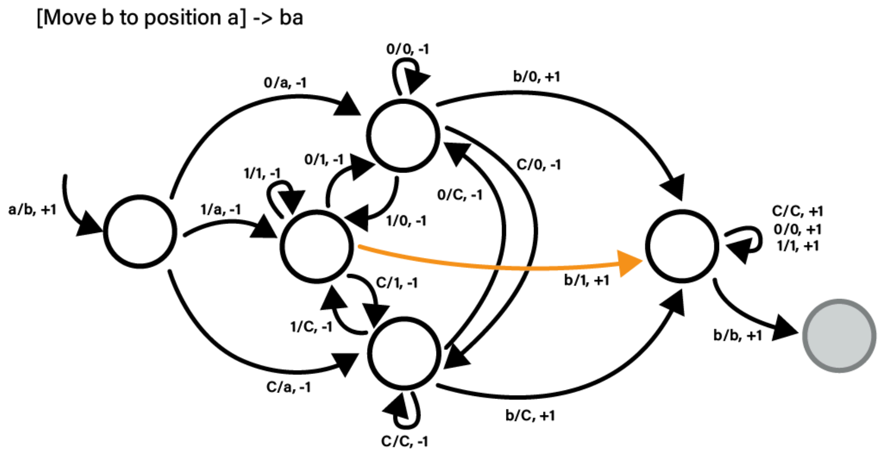

Image 14: *Diagram of the Move pointer to the right Machine*

Finally, the two gadgets used to move the symbols for the pointers to a different location. These machines assume that the tape head starts in the position the pointer will be moved to. The two gadgets have a similar configuration. They start by replacing the current symbol by the pointer symbol, and depending on the machine, they start replacing the symbols to the right (for the move to the left machine) or to the left (for the move to the right machine) effectively moving the rest of the tape 1 cell away to the direction contrary to the movement of the pointer. This continues until it reaches the previous position of the pointer that was moved and replaces it with the last symbol that was moved. Then the gadgets return to the new location of the pointer and end with the tape head on the pointer.

1-Tape Universal Turing Machine Implementation.
With all the gadgets developed, we can start with the implementation of the 1-Tape UTM. Earlier in this section, the algorithm for a 3-Tape UTM was described. This algorithm was extended and updated to work for the 1-Tape UTM.

Our 1-Tape UTM is defined as a Turing Machine with a set of Q states, an input alphabet {0,1}, a tape alphabet {0,1,X,Y,Z,P}, a set of transition functions δ, a start state q_s, an accepting state q_{acc} and a rejecting state q_{rej}.

Algorithm for a 1-Tape UTM

1. Add the pointers to the encoding of the Turing Machine M.
2. Move the Y pointer to the first transition in the transition function "delta."
3. Return to X and compare X and Y. If they are the same go to step 5 If they are not, go to step 4.
4. Move the Y pointer to the next transition of delta and return to step 3. If there are no transitions left, reject the string.
5. Move the Y pointer to the next item in the current transition (symbol a).
6. Compare Y and Z. If they are the same, go to step 7, if they are not, return to step 4.
7. Empty the buffer in X (turn the current state to a string of 0's) and move the Y pointer to the next item in the current transition (state q). Then, copy Y to X ().
8. Move the Y pointer to the next item in the current transition (symbol b) and replace Z with Y.
9. Move the Y pointer to the next item in the current transition (Movement of the tape head in M). If the transition indicates that the tape head has to move to the right, move the Z pointer to the right. If the transition indicates a movement to the left, move the Z pointer to the left. (If the tape head is at the beginning of the tape, do not move it).
10. Return to X and read the current state of M. If the state is the accepting state, accept the string; if the state is in the rejecting state, reject the string. If the state is any other state, go to step 11.
11. Move the Y pointer to the first term of the transition function "delta" and loop back to step 2 to continue simulating the machine M.

Each of these steps will be described with detail in the following sections with their diagrams of the portions of the UTM they cover. When a portion of the UTM uses one of the gadgets, the word that describes the specified machine will be written in bold letters for quick referencing.

Step 1: Add the pointers to the encoding of the Turing Machine M.

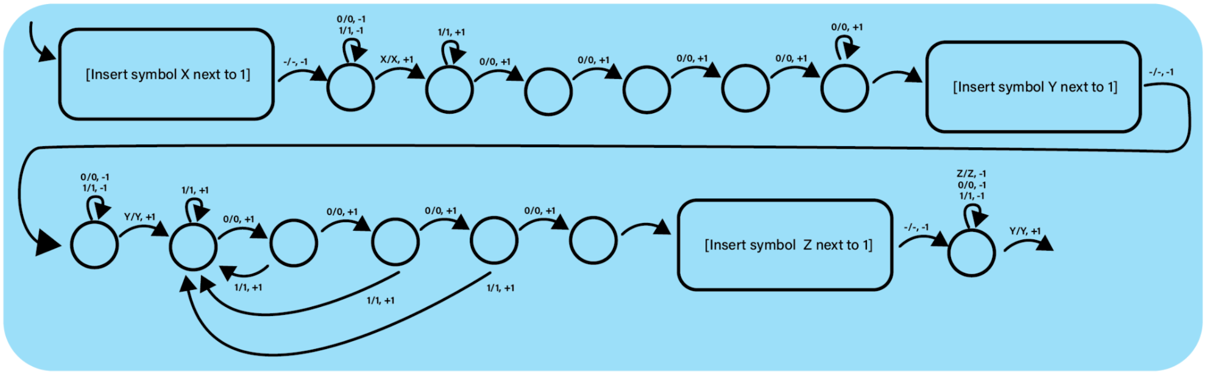

Image 15: *Diagram for Step 1, 1-Tape UTM*

The tape head starts over the left-most cell of the UTM tape. This is the position where pointer X should be, so the UTM **inserts** the symbol for pointer X in this cell. Once the pointer X is in the tape, the UTM will return from the end of the tape to the position of the newly added pointer X and then it is going to skip all of the information bits (1's) from the first "encoded tape" and once if finds a 0, it will transition to the next stage only if it can find at least 4 consecutive 0's (remember that 0000 is the symbol used to differentiate between tapes). Once it finds the end of a sequence of at least 4 0's, the tape head will be over the encoding of the second tape, here is the position where pointer Y should start, so the UTM **inserts** the symbol Y in the current position, marking the beginning of the second tape. Once it completes that, the tape head is returned from the end of the tape to the new location of pointer Y. Then, the UTM skips all of the symbols to the right of pointer Y until it finds a sequence of 4 0's (indicator of the 3rd tape). The UTM keeps track of the number of 0's that have been found by assigning a state for each of the possible number of 0's in a row and returning to the original state if a 1 appears and breaks the sequence. Once it finds the location of the third tape, it inserts the symbol Z for the third pointer. In the end, the tape head is moved back to the left where the Y pointer is located and ends on the Y pointer.

Step 2: Move the Y pointer to the first transition in the transition function "delta."

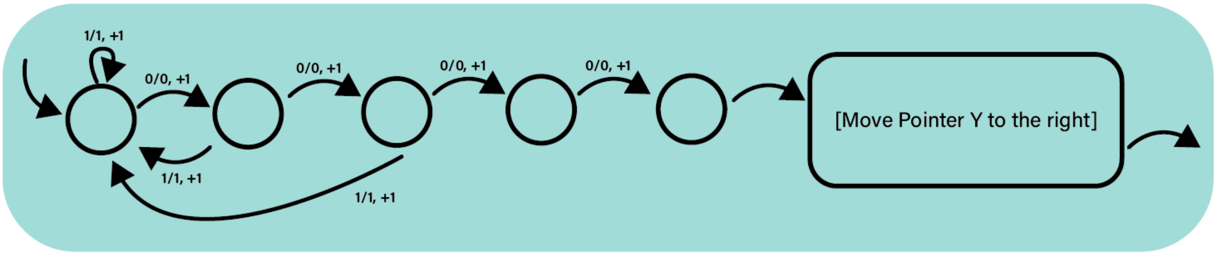

Image 16: *Diagram for Step 2, 1-Tape UTM*

Now that the tape head is located over the Y pointer, the UTM through the contents of "tape 2" until it finds a sequence of exactly 3 0's. As explained for the encoding, this sequence differentiates the set of transition functions (“delta”) from the other elements of the simulated machine M. Now positioned at the beginning of the set of transitions, the UTM **moves the pointer** Y to this new location.

Step 3: Return to X and **compare X and Y**. If they are the same go to step 5 If they are not, go to step 4.

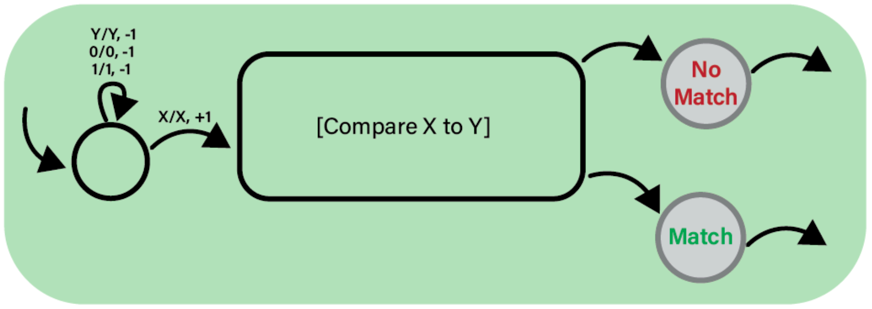

Image 17: *Diagram for Step 3, 1-Tape UTM*

Now, the UTM returns back to the beginning of the tape until it finds pointer X, and in that position, it **compares** the content in pointer X with the content in pointer Y. As a reminder, this is effectively comparing the current state of the simulated Turing Machine M with the state p in the first transition of "delta."

Step 4: Move the Y pointer to the next transition of delta and return to step 3. If there are no transitions left, reject the string.

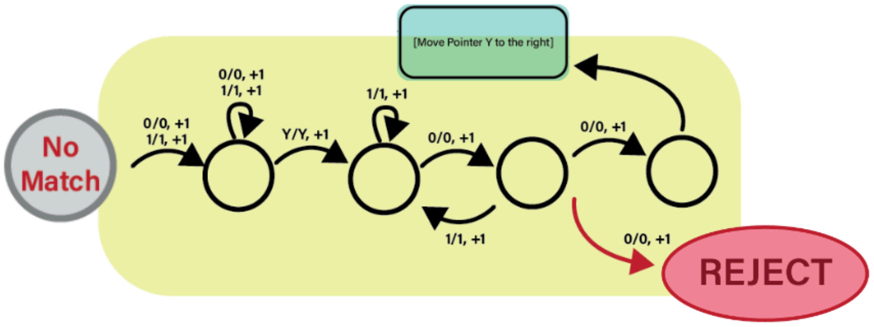

Image 18: *Diagram for Step 4, 1-Tape UTM*

If in step 3, the end state of the UTM is in the No-Match state, that means that the analysed transition is incorrect, so it returns the tape head to the pointer Y, and then looks for the next appearance of a sequence 00, which indicates the next transition for the transition set, then it **moves** the pointer Y to the current location, to then return to step 3. However, if the sequence is insead 000, that means we reached the end of the transition states, meaning that there was no transition that matches the current state of the Turing Machine M and the current symbol in the tape; following the behaviour of a TM, this string is immediately rejected.

Step 5: Move the Y pointer to the next item in the current transition (symbol a).

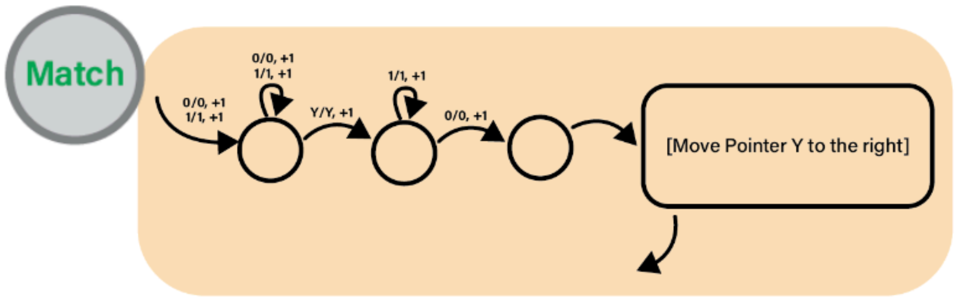

Image 19: *Diagram for Step 5, 1-Tape UTM*

If in step 3, the end state is the Match state, this means that the current transition could be the correct transition to perform. So, the tape head moves to the right looking for a 0 that separates the elements in the current transition. As a reminder a transition is defined as the tuple $\{p, a, q, b, \pm\ 1\}$. The UTM just did the comparison between the current state and p and they match, so it will move to the next term in the tuple (symbol a) separated by a single 0 and it **moves** the pointer to this location.

Step 6: Compare Y and Z. If they are the same, go to step 7, if they are not, return to step 4.

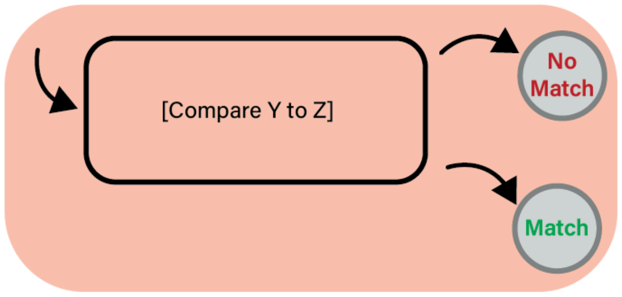

Image 19: *Diagram for Step 6, 1-Tape UTM*

With the tape head over Y, **compare** the symbol in pointer Y with the symbol in the pointer Z (the current symbol in the tape of M). This is essentially comparing the symbol in the current transition and the symbol in the tape of M (where the tape head of M would be at this moment) to see if this is the correct transition to take. If the two symbols match, that means that this is the correct transition, so it moves forward to the next step, but if they are different, this is the incorrect transition, so it returns to step 4 to find the next possible transition from the set of transitions.

Step 7: Empty the buffer in X (turn the current state to a string of 0's) and **move** the Y pointer to the next item in the current transition (state q). Then, **copy** Y to X.

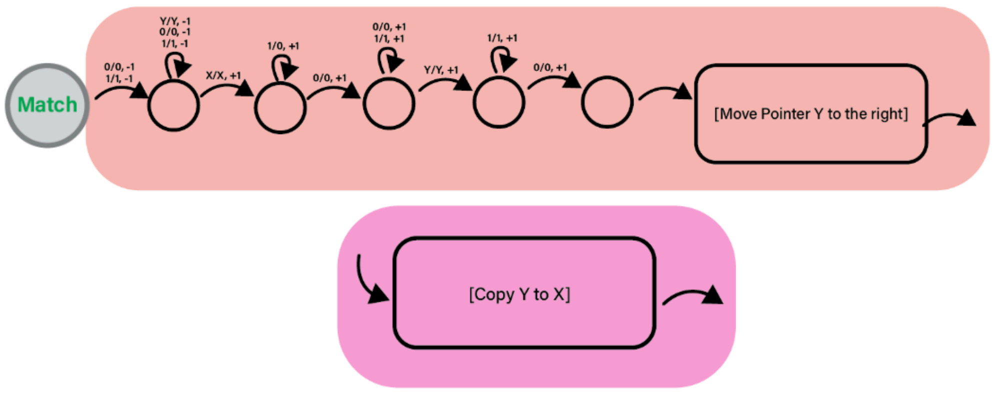

Image 20: *Diagram for Step 7, 1-Tape UTM*

If this is the correct transition (p is equal to the current transition, and a is equal to the current symbol in the tape of M), return the tape head of the UTM to the pointer X, and move to the right transforming all of the 1's that encode the current state of M into 0's, in other words, clearing the "buffer" (this is needed since the copy operation assumes that the buffer is empty.

Then, continues moving to the right until it reaches the pointer Y. Now we need to get to the next item in the current transition (q) to determine the next step. So the tape head continues moving to the right until it finds a 0, that indicates the next term in the transition. Once it finds it, the pointer Y is **moved** to this location. With the tape head over the pointer Y, the UTM **copies** the contents of the pointer Y to pointer X, effectively changing the state of the simulated Turing Machine M.

Step 8: Move the Y pointer to the next item in the current transition (symbol b) and **replace** Z with Y.

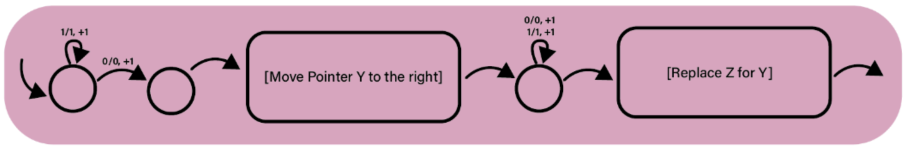

Image 21: *Diagram for Step 8, 1-Tape UTM*

Now, the tape head moves to the right of the pointer Y to find the next item in the current transition (symbol b). Once it finds it, it **moves** the pointer Y to this position and the tape head continues moving to the right until it finds the pointer Z. In this position, the UTM **replaces** the symbol in pointer Z for the symbol in pointer Y (replacing the current symbol a in the tape of M with the new symbol b from the transition).

Step 9: Move the Y pointer to the next item in the current transition (Movement of the tape head in M). If the transition indicates that the tape head has to move to the right, **move** the Z pointer to the right. If the transition indicates a movement to the left, **move** the Z pointer to the left. (If the tape head is at the beginning of the tape, do not move it).

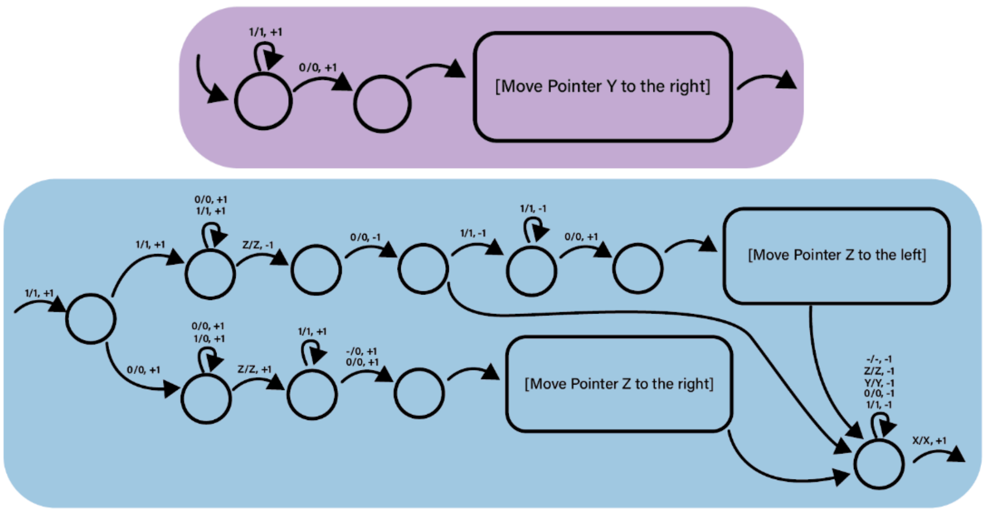

Image 22: *Diagram for Step 9, 1-Tape UTM*

Now, the tape head of the UTM will move to the right to find the next element of the current transition (which is the movement that the head tape for the simulated Turing Machine M should perform). Once it finds it, the UTM **moves** the Y pointer to this new location. If the content in Y is a two in unary, this means that the tape should move to the left, so the tape head goes over to the position of pointer Z, and checks if there is a single 0 behind Z. If there is only one 0, the tape head continues moving to the left until it finds the beginning of the previous cell in the tape of M and then **moves** the pointer Z to this new location accordingly. If there are more than one 0's, that means this is the beginning of the tape and thus, the pointer should not be moved.

If the pointer Y has a single 1 (meaning, move Z to the right), the tape head of the UTM moves to the right until it finds the pointer Z, and continues moving to the right, skipping the information contained in Z. Once the tape head reaches the end of the current cell in the tape M, the UTM **moves** the pointer Z to this new location.

The three possibilities that were just described, end up by taking the tape head of the UTM and moving it to the left until it reaches the pointer X.

Step 10: Return to X and read the current state of M. If the state is the accepting state, accept the string; if the state is in the rejecting state, reject the string. If the state is any other state, go to step 11.

The tape head of the UTM (located in pointer X) now reads the content of the pointer X (or the "buffer"), if it finds that the current state is the state 2 or 3 (11 and 111 as written in unary), this means that the simulated Turing Machine M has reached an ending state. If the state is 2 (the accepting state), this means that M would have accepted the string w, so the UTM moves to the accepting state to agree with this behaviour. If the state is 3 (the rejecting state), this means that M would have rejected the string w, accordingly, the UTM moves to the reject state to recreate M's behaviour. If the current state in X is anything else, this means that the simulation continues, so the UTM skips over the rest of the information in the buffer X and proceeds to step 11.

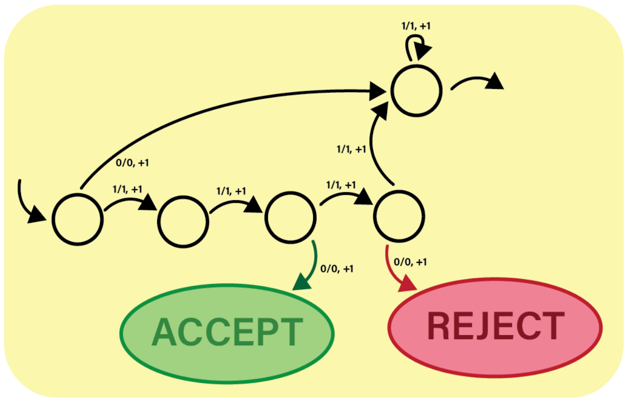

Image 23: *Diagram for Step 10, 1-Tape UTM*

Step 11: Move the Y pointer to the first term of the transition function "delta" and loop back to step 2 to continue simulating the machine M.

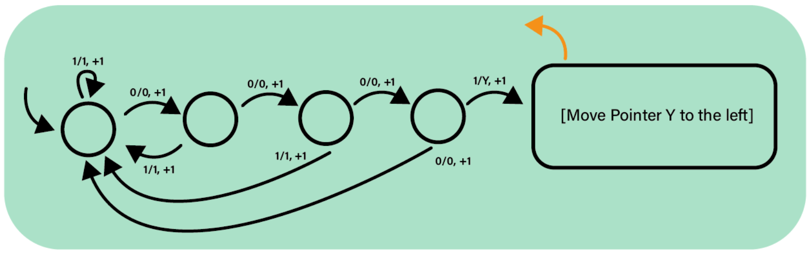

Image 24: *Diagram for Step 11, 1-Tape UTM*

With the tape head of the UTM at the end of the buffer, the UTM continues moving to the right, skipping over the information bits with a 1 and “counting” the number of consecutive 0’s it finds on its way by assigning a state of the UTM to identify how many 0’s the tape head has found. Once the tape head finds a sequence of exactly 3 0’s, the beginning of the transition function has been found. Now the UTM **moves** the pointer Y (that might be in the middle of any transition of the transition function) back to the beginning of the transition function, so we can start the loop of simulating the Turing Machine M all over again. At the end of this step, the UTM goes back to step 2 and continues the simulation.

This is the complete implementation of our 1-Tape Universal Turing Machine, Image 25 shows the illustration of all the steps assembled together. Because the full UTM has several states, the Image in this document may be too compressed to review, so we included a link to the full size diagram through Google Drive that can be downloaded for the full resolution:

https://drive.google.com/file/d/1htfQfajehWTLOyI5wVxVgVO3T6RiREFm/view?usp=sharing

Note: The link should provide commentator access to anyone with the link (which includes viewing). Should any issues arise, let us now to update the sharing preferences.

Full diagram - 1-Tape Universal Turing Machine

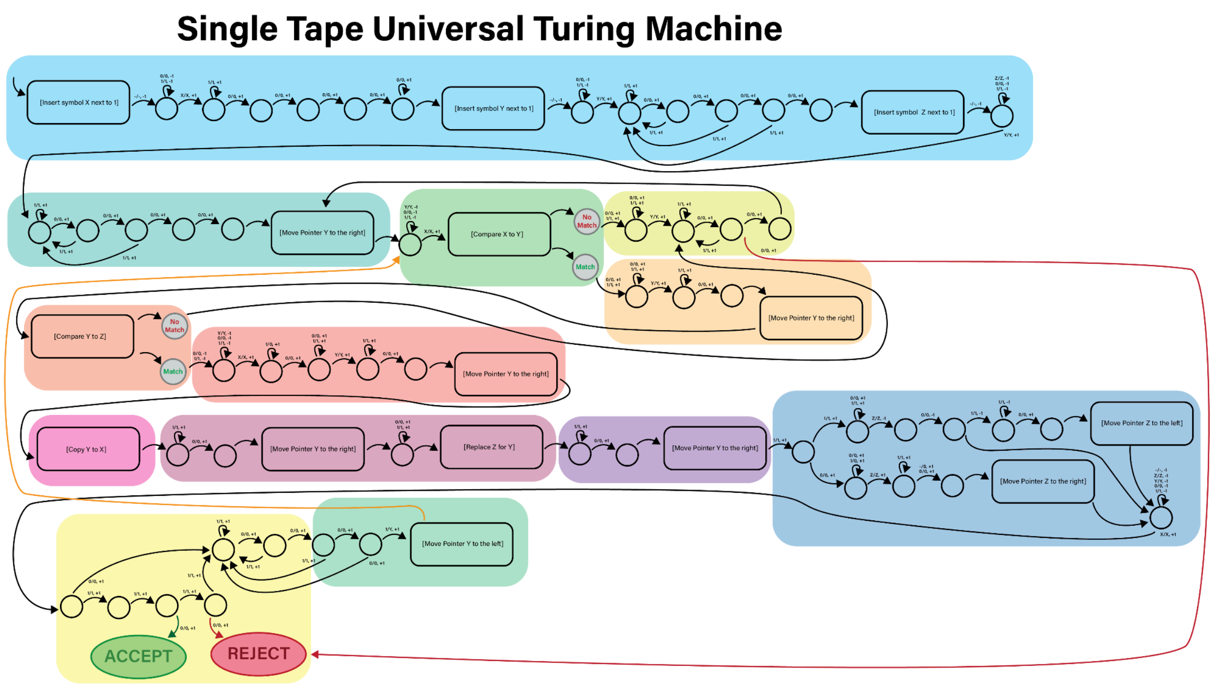
Single Tape Universal Turing Machine
[Insert symbol X next to 1]
[Insert symbol Y next to 1]
[Insert symbol Z next to 1]
[Move Pointer Y to the right]
[Compare X to Y]
No Match
Match
[Move Pointer Y to the right]
[Compare Y to Z]
No Match
Match
[Move Pointer Y to the right]
[Copy Y to X]
[Move Pointer Y to the right]
[Replace Z for Y]
[Move Pointer Y to the right]
[Move Pointer Z to the left]
[Move Pointer Z to the right]
[Move Pointer Y to the left]
ACCEPT
REJECT

Image 25: *Full diagram for the Universal Turing Machine*