Theory of Computation: Final Project

Nandini Patel, Het Patel, Niraj Patel

Dr. Luis Rueda

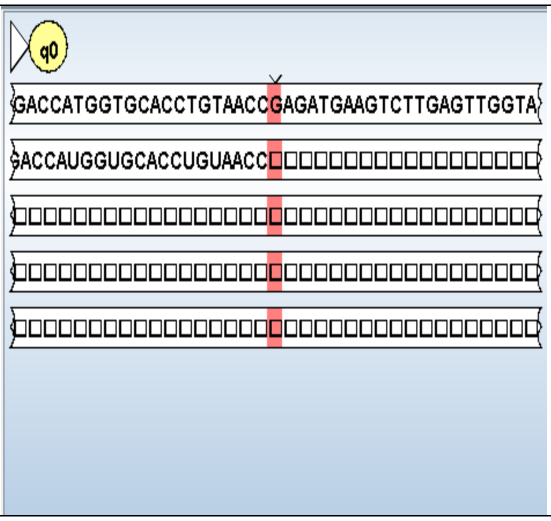
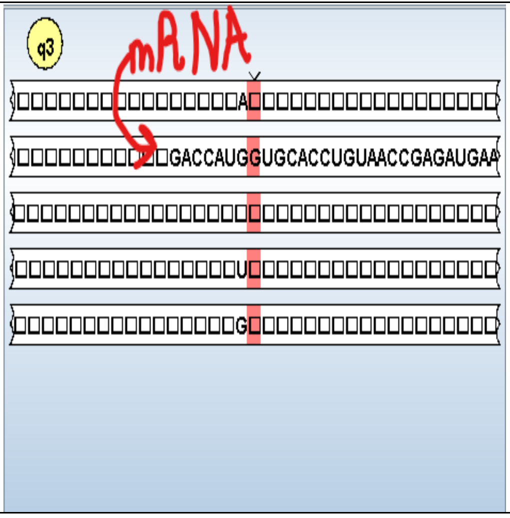
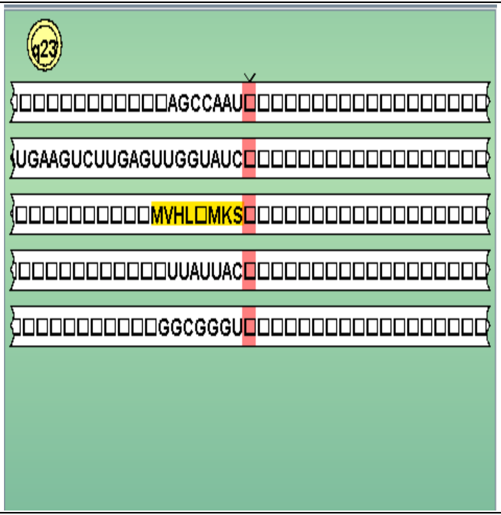
December 15, 2020

Group Work Distribution

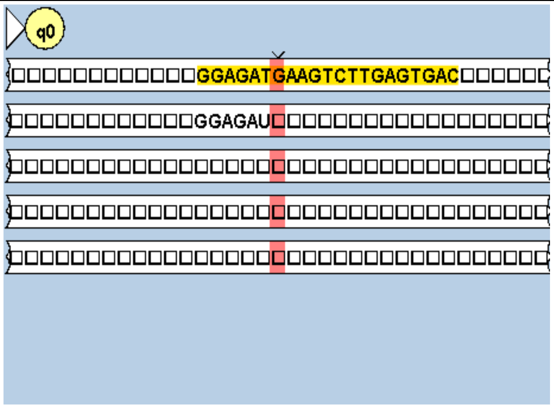
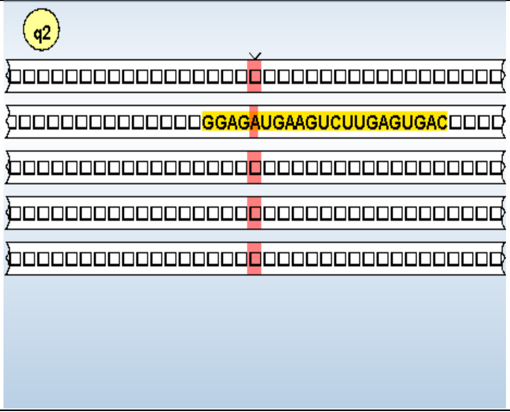
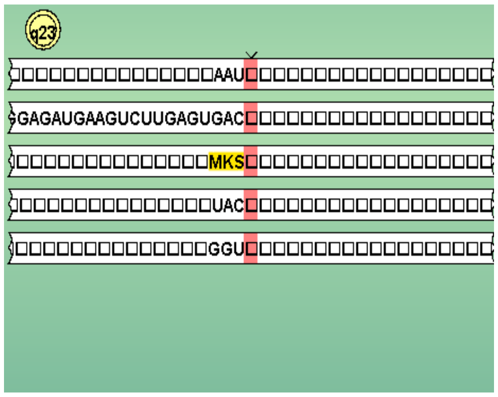
For the group final project, all the members worked together on all the aspects of the project. The group held daily meetings of 1 hour, in which we brainstormed the Turing Machine for Part A, the implementation of the TM in JFlap. For the 2 program, one member coded the program will two others helped him. The reason behind this was in previous CS class we learned pair programming. So, we applied that but with 3 people making more efficient. For the report Niraj worked on Part A, Nandini and Het worked on Part B.

Part A

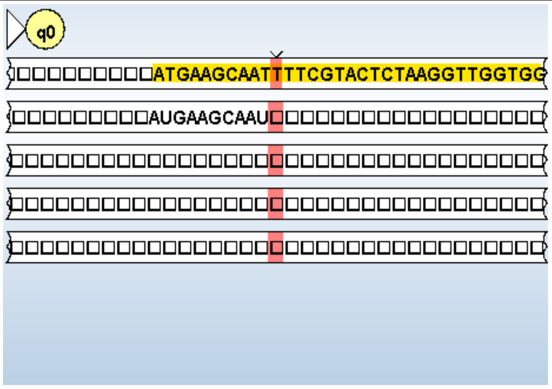
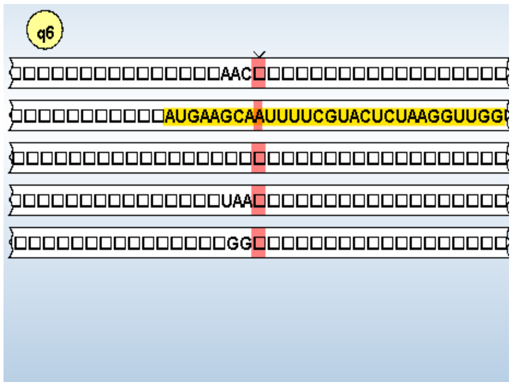
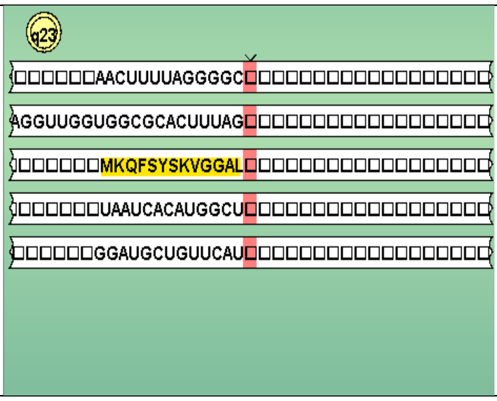
1. Final project was done in a group. The group decided to use a multi-tape. This way each tape will have its own head which can move freely. An overview of the design is the first tape will be the input DNA, then the second tape will have the mRNA, and lastly the third tape will have mRNA to Protein. The other tapes will be used to successfully go from the DNA→ mRNA → Protein. The appropriate JFlap (.jff) file is attached.
2. **NOTE:** In our project implementation our group have decided to implement using it option b. This is the option where the program will go through the whole string no matter if it finds one start stop. So, in our implementation if the mRNA has a Start…Stop...Start and it does not have a Stop for the last Start it will result in a failure state. So, for example the following DNA would fail our program. The Start is highlighted in blue and stop is in red. Ex: GACCATGGTGCACCTGTAACCGAGATGAAGTCT. This type of DNA would be rejected. Below are some of the test cases that we ran.
   The following is list of Accepting DNA and the pictures are in order
   - GACCATGGTGCACCTGTAACCGAGATGAAGTCTTGAGTTGGTATC → MVHL MKS

DNA	mRNA	Protein
 q0 GACCATGGTGCACCTGTAACCGAGATGAAGTCTTGAGTTGGTA GACCAUGGUGCACCUGUAACC□□□□□□□□□□□□□□□□□□	 q3 □□□□□□□□□□□□□□□□□A□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□GACCAUGGUGCACCUGUAACCGAGAUGAA □□□□□□□□□□□□□□□□□U□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□G□□□□□□□□□□□□□□□□□□	 q23 □□□□□□□□□□□AGCCAAU□□□□□□□□□□□□□□□□□□ UGAAGUCUUGAGUUGGUAUC□□□□□□□□□□□□□□□□□□ □□□□□□□□□□MVHL□MKS□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□UUAUUAC□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□GGCGGGU□□□□□□□□□□□□□□□□□□

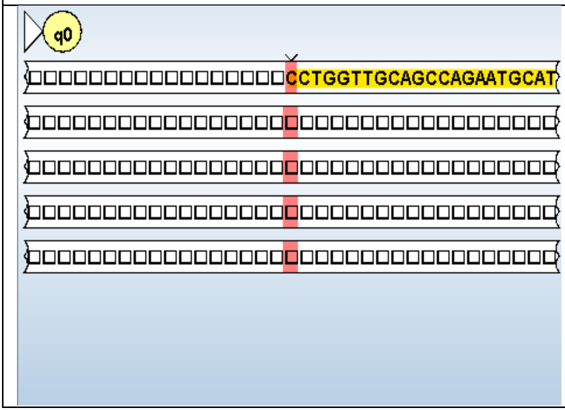
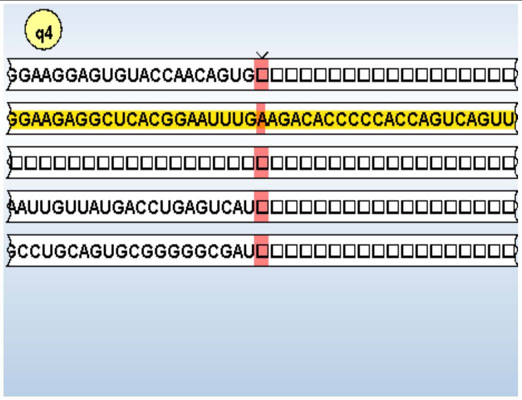
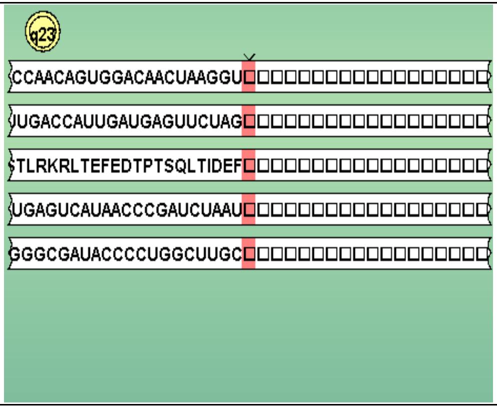
- GGAGATGAAGTCTTGAGTGAC → MKS

DNA	mRNA	Protein
 q0 □□□□□□□□□□□□GGAGATGAAGTCTTGAGTGAC□□□□□□ □□□□□□□□□□□□GGAGAU□□□□□□□□□□□□□□□□□□	 q2 □□□□□□□□□□□□□□GGAGAUGAAGUCUUGAGUGAC□□□□	 q23 □□□□□□□□□□□□□□AAU□□□□□□□□□□□□□□□□□□ GAGAUGAAGUCUUGAGUGAC□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□MKS□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□UAC□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□GGU□□□□□□□□□□□□□□□□□□

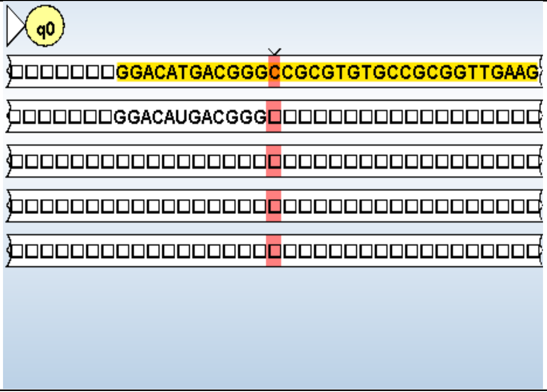
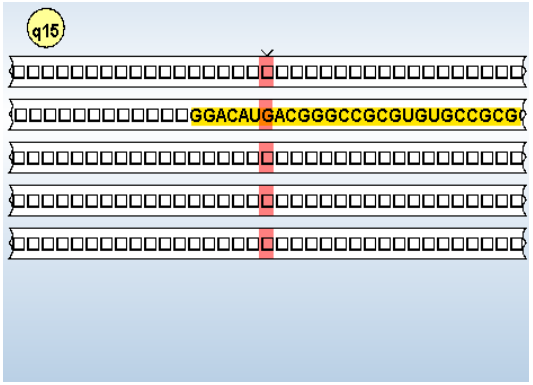
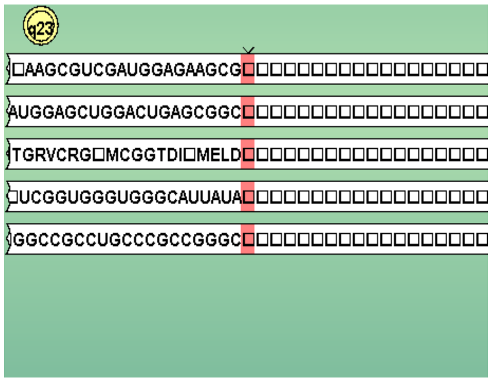
- ATGAAGCAATTTTCGTACTCTAAGGTTGGTGGCGCACTTTAG → MKQFSYSKVGGAL

DNA	mRNA	Protein
 q0 □□□□□□□□□ATGAAGCAATTTTCGTACTCTAAGGTTGGTGG □□□□□□□□□AUGAAGCAAU□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□	 q6 □□□□□□□□□□□□□□□AAC□□□□□□□□□□□□□□□□□□ □□□□□□□□□□AUGAAGCAAUUUUCGUACUCUAAGGUUGG □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□UAA□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□GG□□□□□□□□□□□□□□□□□□	 q23 □□□□□□AACUUUUAGGGGC□□□□□□□□□□□□□□□□□ AGGUUGGUGGCGCACUUUAG□□□□□□□□□□□□□□□□□ □□□□□□MKQFSYSKVGGAL□□□□□□□□□□□□□□□□□ □□□□□□UAAUCACAUGGCU□□□□□□□□□□□□□□□□□ □□□□□□GGAUGCUGUUCAU□□□□□□□□□□□□□□□□□

- CCTGGTTGCAGCCAGAATGCATGACTTCAGGAGGACTGTGAAGGAGG TCATCAGTGTGGTCAAAGTGTGTGAGTCCACGCTGCGGAAGAGGCTCA CGGAATTTGAAGACACCCCCACCAGTCAGTTGACCATTGATGAGTTCT AGCACGACTTCAGG → MHDFRRTVKEVISVVKVCESTLRKRLTEFEDTPTSQLTIDEF

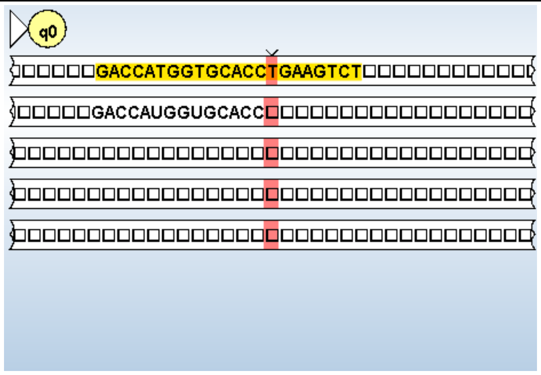
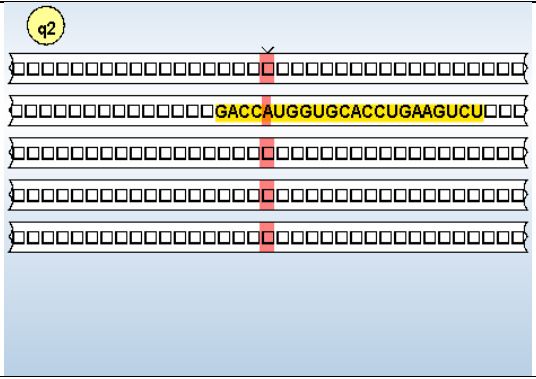
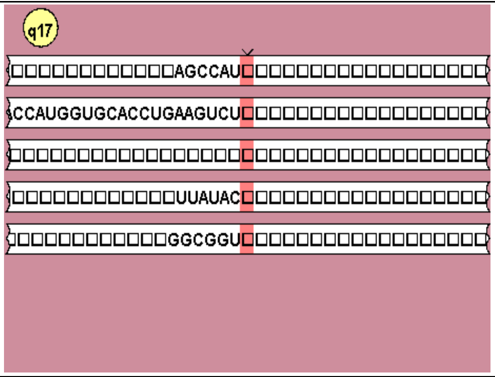
DNA	mRNA	Protein
 q0 □□□□□□□□□□□□□□□□□□CCTGGTTGCAGCCAGAATGCAT □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□	 q4 GGAAGGAGUGUACCAACAGUG□□□□□□□□□□□□□□□□□ GGAAGAGGCUCACGGAAUUUGAAGACACCCCCACCAGUCAGUU □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ AAUUGUUAUGACCUGAGUCAU□□□□□□□□□□□□□□□□□ GCCUGCAGUGCGGGGGCGAU□□□□□□□□□□□□□□□□□	 q23 CCAACAGUGGACAACUAAGGU□□□□□□□□□□□□□□□□□ UGACCAUUGAUGAGUUCUAG□□□□□□□□□□□□□□□□□ TLRKRLTEFEDTPTSQLTIDEF□□□□□□□□□□□□□□□□□ UGAGUCAUAACCCGAUCUAAU□□□□□□□□□□□□□□□□□ GGGCGAUACCCCUGGCUUGC□□□□□□□□□□□□□□□□□

- GGACATGACGGGCCGCGTGTGCCGCGGTTGAAGATGTGCGGCGGCAC GGACATCTAATGCAATGGAGCTGGACTGAGCGGC → MTGRVCRG MCGGTDI MELD

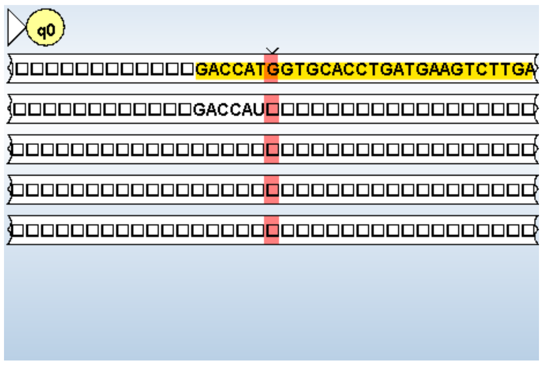
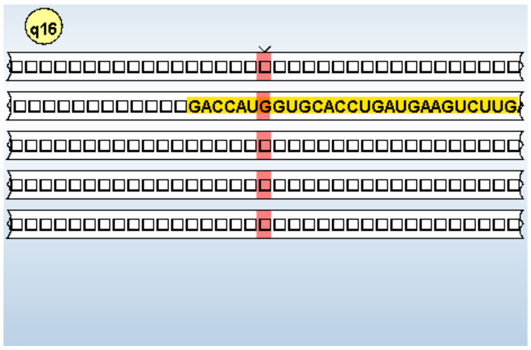
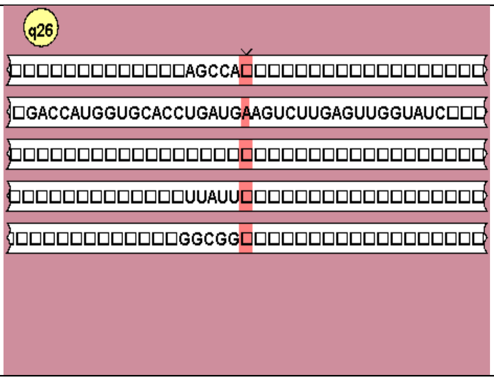
DNA	mRNA	Protein
 q0 □□□□□□□GGACATGACGGGCCGCGTGTGCCGCGGTTGAAG □□□□□□□GGACAUGACGGG□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□	 q15 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□GGACAUGACGGGCCGCGUGUGCCGCG □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□	 q23 □AAGCGUCGAUGGAGAAGCG□□□□□□□□□□□□□□□□□ AUGGAGCUGGACUGAGCGGC□□□□□□□□□□□□□□□□□ TGRVCRG□MCGGTDI□MELD□□□□□□□□□□□□□□□□□ UCGGUGGGUGGGCAUUAUA□□□□□□□□□□□□□□□□□ GGCCGCCUGCCCGCCGGGC□□□□□□□□□□□□□□□□□

The following is list of Rejecting DNA and the pictures are in order

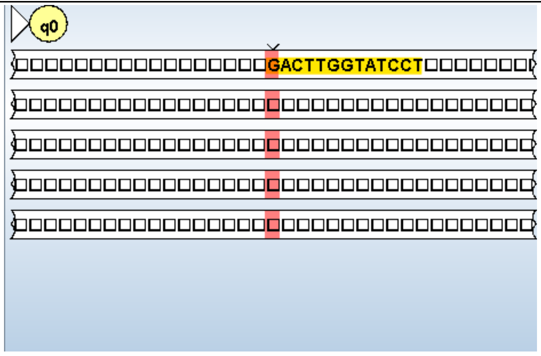
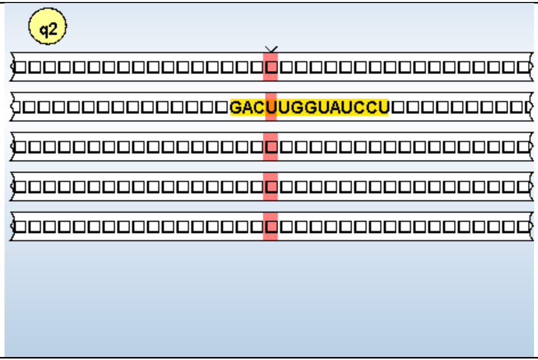
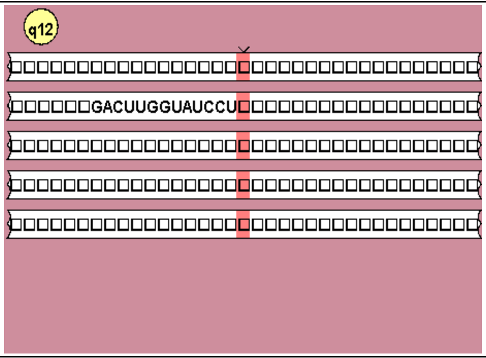
- GACCATGGTGCACCTGAAGTCT → Fail State

DNA	mRNA	Fail State
 q0 GACCATGGTGCACCTGAAGTCT GACCAUGGUGCACC	 q2 GACCAUGGUGCACCUGAAGUCU	 q17 AGCCAU CCAUGGUGCACCUGAAGUCU UUAUAC GGCGGU

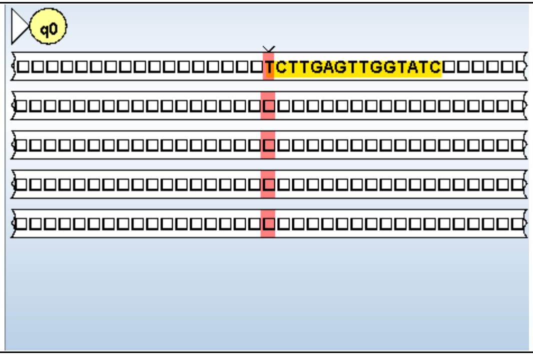
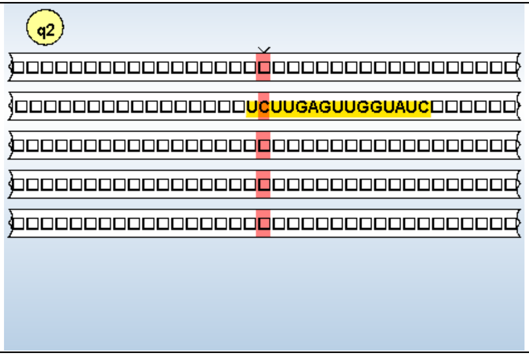
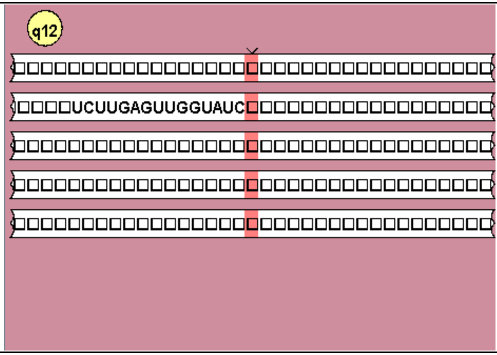
- GACCATGGTGCACCTGATGAAGTCTTGAGTTGGTATC → Fail State

DNA	mRNA	Fail State
 q0 GACCATGGTGCACCTGATGAAGTCTTGA GACCAU	 q16 GACCAUGGUGCACCUGAUGAAGUCUUGA	 q26 AGCCA GACCAUGGUGCACCUGAUGAAGUCUUGAGUUGGUAUC UUAUU GGCGG

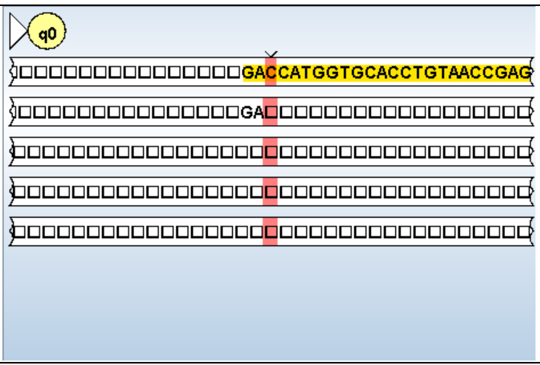
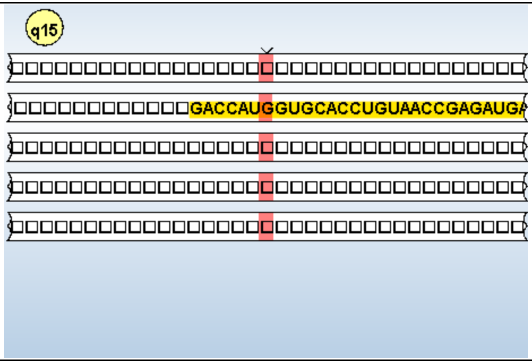
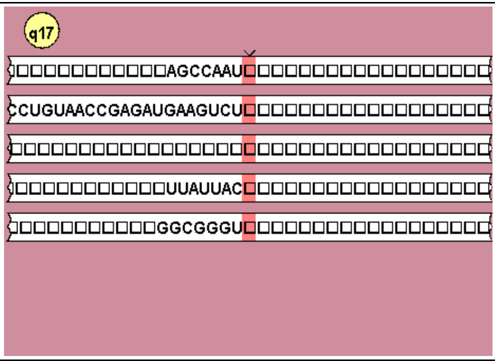
- GACTTGGTATCCT → Fail State

DNA	mRNA	Fail State
 q0 GACTTGGTATCCT	 q2 GACUUGGUAUCCU	 q12 GACUUGGUAUCCU

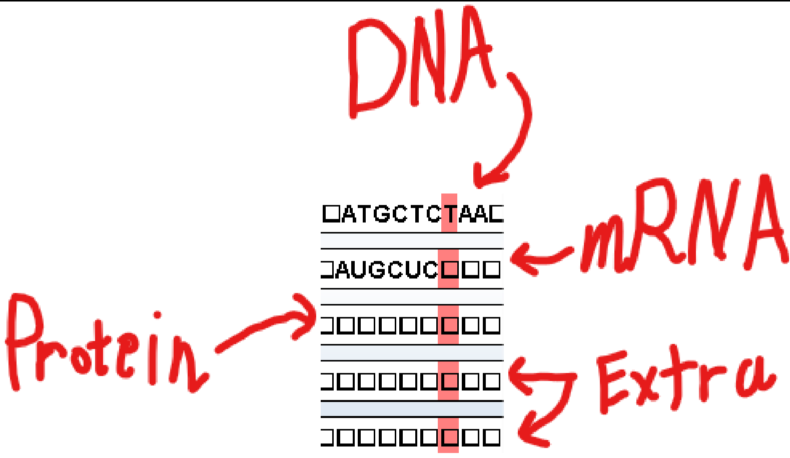
- TCTTGAGTTGGTATC → Fail State

DNA	mRNA	Fail State
		

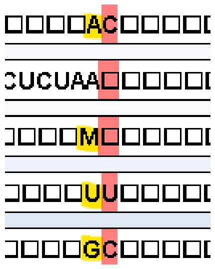
- GACCATGGTGCACCTGTAACCGAGATGAAGTCT → Fail State

DNA	mRNA	Fail State
		 q17 AGCCAAU CUGUAACCGAGAUGAAGUCU UUAUUAC GGCGGGU

3. Our Turing machine is designed with 5 different tapes. In this Turing machine the heads of each tape move independently, meaning that the heads can be at different locations of the respective tapes. The input (DNA) is the first tape, the middle tape is the DNA → mRNA and the third tape are our output which is mRNA → Protein. The other 2 tapes are used as extra, and you can see more specifically how in 5. Which describes each state and its transitions? The general order that the TM follows is that the input is loaded in the first tape, then converted to mRNA and then using the extra tapes as an additional help it converts the mRNA to Protein.

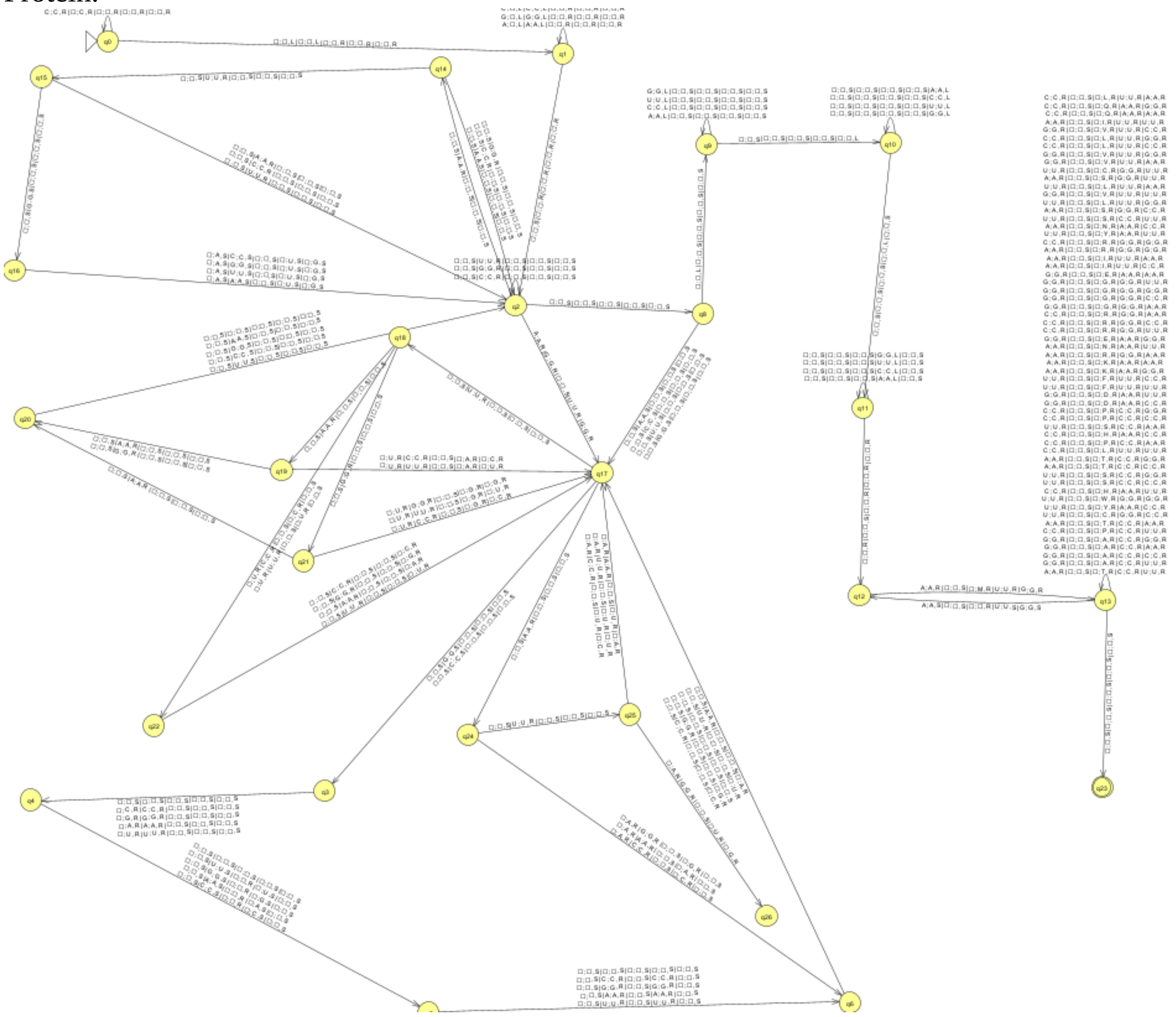


When the TM starts, the TM goes through the first tape and copies each letter to second tape and changes each T to U so therefore we get the mRNA in the second tape. It then traverses back to the beginning of the second tape and the first while replacing the DNA with all Blanks in the first tape. Then it goes through the second tape (mRNA) and looks for AUG codon, once it first that it starts placing the first letter of the codon in the 1st tape, the second letter in the 4th, tape, and the 3rd letter in the 5th



tape. Once all the mRNA is done it traverses the head of 1st, 4th, 5th, tape to start and maps each letter by letter to its Protein. Observe the picture attached above. So, for example if the first tape had A, fourth tape had U and the fifth tape had G, then the TM will put M on the third tape and move the heads of 1st, 4th, 5th to the right one more. This will happen until the heads of 1st, 4th, 5th is not pointing to blank.

4. This is the transition diagram from the JFlap which converts the DNA → mRNA → Protein.



5. The following table describes each states' job and its transitions:

State	Description
Q0	Goes through the DNA on first tape and changes all the T to U for the mRNA on the second tape
Q1	Makes the header for tape 1 and tape 2 to the start of the DNA and mRNA. It also erases the DNA from tape 1
Q2	Goes through the mRNA on second tape, until it finds an "A"

State	Description
Q2→Q14	Goes to Q3 when it finds an "A" other wises loops in state Q2
Q14→Q2	Goes back to Q2 if it finds 'C', 'G', or 'A' since then that indicates its not a starting codon
Q14→Q15	Goes to Q15 when it finds an 'U'
Q15→Q2	Goes back to Q2 if it finds 'C', 'G', or 'A' since then that indicates its not a starting codon
Q15→Q16	Goes to Q16 when it finds an "G"
Q16→Q2	Goes to Q17 and puts "A" in the first tape, "U" in the second tape, and "G" in the third tape
Q2→Q17	Moves the second tape one left
Q17→Q24	Goes to Q24 if the symbol on second tape is "A"
Q24→Q25	Goes to Q25 if the symbol on the second tape is "U"
Q25→Q26	Goes to Q26 if the symbol on second tape is "G" which means we found a second starting codon inside another starting codon
Q25→Q17	Goes to Q17 if we find "U", "A", or "C". This means that inside the starting codon we checked for another starting on, and we found something like AUU, AUA, AUC but not AUG.
Q24→Q6	If we find a "A", "G", or "C" on second tape it means we have the next three letters as AGX, AAX, ACX, where X is the next symbol waiting to be read. A is then added to first tape and the respective one is added to the 4th tape.
Q6→Q17	Reads the symbol from the 2nd tape and adds it to the 5th tape
Q17→Q3	If it reads "G" or "C" on the second tape
Q3→Q4	Reads the next input on the second tape and put it on the first tape
Q4→Q5	Reads the next input on the second tape and puts it on the 4th tape

State	Description
Q17→Q18	If it reads a "U" on the second tape it means it could be a stopping codon
Q18→Q22	If it reads "C", or "U" on seconds tape it means it is no longer a stopping codon since we have UC, or UU. Puts U in the first tape and the read letter in the 4th tape
Q22→Q17	Reads the next letter from the second tape and puts it in the 5th tape
Q18→Q19	If it reads "A" form the second tape it means it could be the codon UAA, or UAG
Q19→Q17	If it reads "C" or "U" form the second tape it means it is no longer a stopping codon since it is UAC, or UAU. It then puts U in the first tape, A in the 4th tape, and the read letter in the 5th tape
Q18→Q21	If it reads "G" from the second tape it means it could be the UGA stopping codon
Q21→Q17	If it reads "G", "U", or "C" from the second tape it means it is no longer a stopping codon. Since it is UGG, UGU, UGC. It then puts U in the first tape, G in the 4th tape and the read letter in the 5th tape
Q19→Q20	If it reads A or G from the second tape it means we successfully read a stopping codon and they could be UAA, or UAG
Q21→Q20	If it reads A from the second tape it means we successfully read a stopping codon and it is UGA
Q20→Q2	If goes to Q2 and then starts looking for another opening codon and loops
Q2→Q8	Goes to Q8 when no more mRNA is left to be read
Q8→Q9	Moves the first tape in one left position
Q9	Loops itself until first tape does not encounter a Blank while going left. So, goes to the start position of the first tape
Q9→Q10	Moves the 5th tape in one left position when first tape encounters a blank when going left

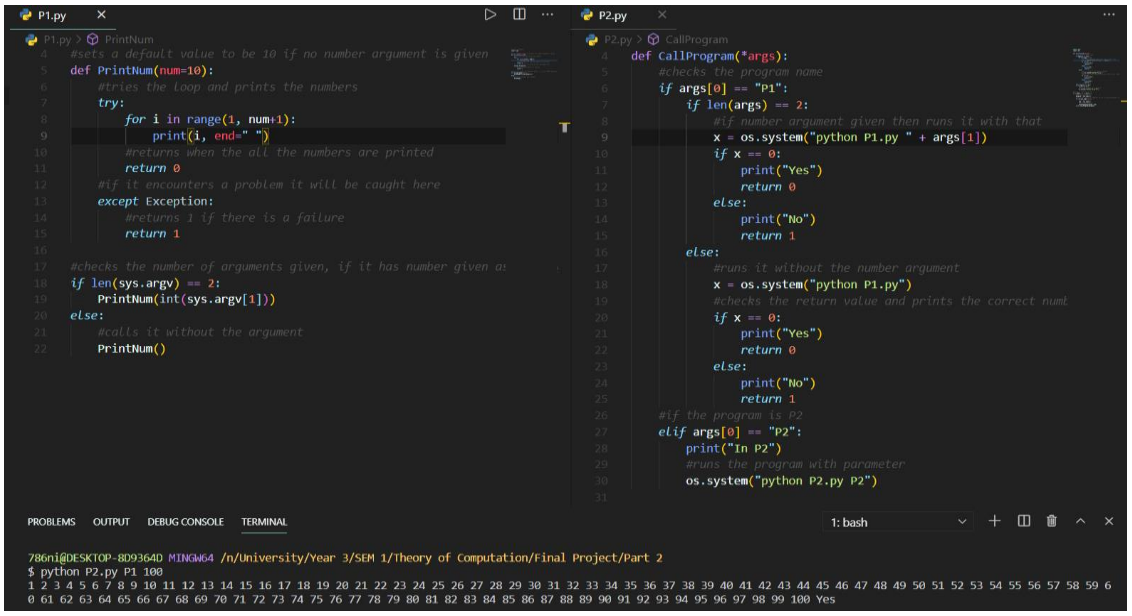
State	Description
Q10	Loops itself until the fifth tape does not encounter a Blank while going left. So, goes to the start position of the fifth tape
Q10→Q11	Moves the 4th tape in one left position when fifth tape encounters a blank when going left
Q11	Loops itself until the fourth tape does not encounter a blank while going left. So, goes to the start position of the fourth tape
Q11→Q12	When the 4th tape encounters a blank and moves pointers of tape 1, 4, 5 right
Q12→Q13	Compares the letters that the heads of tape 1, 4, 5 are pointing at. If tape 1 is A and tape 4 is U and tape 5 is G. It means it is the starting codon and it then places a M on the 3rd tape
Q13	Loops itself, compares the letters that the heads are point on tape 1, 4, 5 and matches it to the correct protein.
Q13→Q12	If in the loop, tape 1 has A and tape 4 has U and tape 5 has G it means it is another starting codon, so it leaves a Blank in the 3rd tape
Q13→Q23	If the first, fourth, and fifth tape reach a Blank it means all the mRNA has been read and the respective protein is printed on the 3rd tape

Part B

*Please refer to P1.py and P2.py to see the code

2. Program Analysis

a. *python P2.py P1 100*


P1.py

```python
#sets a default value to be 10 if no number argument is given
def PrintNum(num=10):
    #tries the loop and prints the numbers
    try:
        for i in range(1, num+1):
            print(i, end=" ")
        #returns when the all the numbers are printed
        return 0
    #if it encounters a problem it will be caught here
    except Exception:
        #returns 1 if there is a failure
        return 1

#checks the number of arguments given, if it has number given a
if len(sys.argv) == 2:
    PrintNum(int(sys.argv[1]))
else:
    #calls it without the argument
    PrintNum()
```

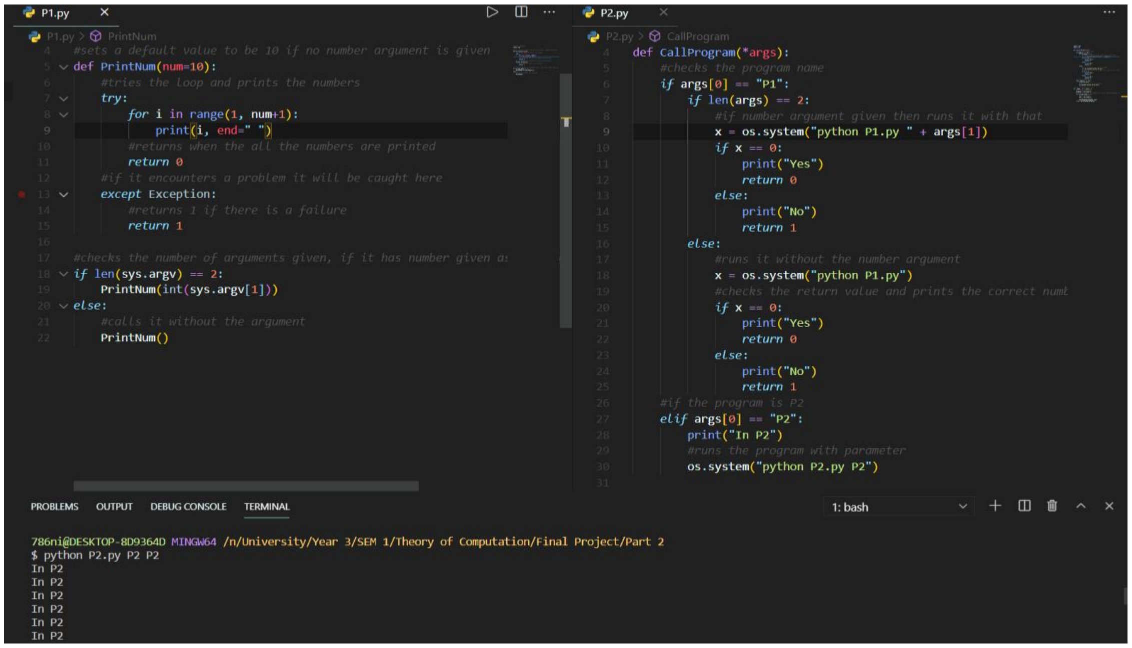
P2.py

```python
def CallProgram(*args):
    #checks the program name
    if args[0] == "P1":
        if len(args) == 2:
            #if number argument given then runs it with that
            x = os.system("python P1.py " + args[1])
            if x == 0:
                print("Yes")
                return 0
            else:
                print("No")
                return 1
        else:
            #runs it without the number argument
            x = os.system("python P1.py")
            #checks the return value and prints the correct numb
            if x == 0:
                print("Yes")
                return 0
            else:
                print("No")
                return 1
    #if the program is P2
    elif args[0] == "P2":
        print("In P2")
        #runs the program with parameter
        os.system("python P2.py P2")
```

```
786ni@DESKTOP-8D9364D MINGW64 /n/University/Year 3/SEM 1/Theory of Computation/Final Project/Part 2
$ python P2.py P1 100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 6
0 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 Yes
```

Here, we can see P2 running. Inside P2, P1(P) is running by taking 100(w) as the input for that program. P1 outputs the first 100 positive integers. Once that is finished, it returns to P2, in which "yes" is printed, and it exits with code 0.

b. *python P2.py P2 P2*


P1.py

```python
#sets a default value to be 10 if no number argument is given
def PrintNum(num=10):
    #tries the loop and prints the numbers
    try:
        for i in range(1, num+1):
            print(i, end=" ")
        #returns when the all the numbers are printed
        return 0
    #if it encounters a problem it will be caught here
    except Exception:
        #returns 1 if there is a failure
        return 1

#checks the number of arguments given, if it has number given a
if len(sys.argv) == 2:
    PrintNum(int(sys.argv[1]))
else:
    #calls it without the argument
    PrintNum()
```

P2.py

```python
def CallProgram(*args):
    #checks the program name
    if args[0] == "P1":
        if len(args) == 2:
            #if number argument given then runs it with that
            x = os.system("python P1.py " + args[1])
            if x == 0:
                print("Yes")
                return 0
            else:
                print("No")
                return 1
        else:
            #runs it without the number argument
            x = os.system("python P1.py")
            #checks the return value and prints the correct numb
            if x == 0:
                print("Yes")
                return 0
            else:
                print("No")
                return 1
    #if the program is P2
    elif args[0] == "P2":
        print("In P2")
        #runs the program with parameter
        os.system("python P2.py P2")
```

```
786ni@DESKTOP-8D9364D MINGW64 /n/University/Year 3/SEM 1/Theory of Computation/Final Project/Part 2
$ python P2.py P2 P2
In P2
In P2
In P2
In P2
In P2
In P2
```

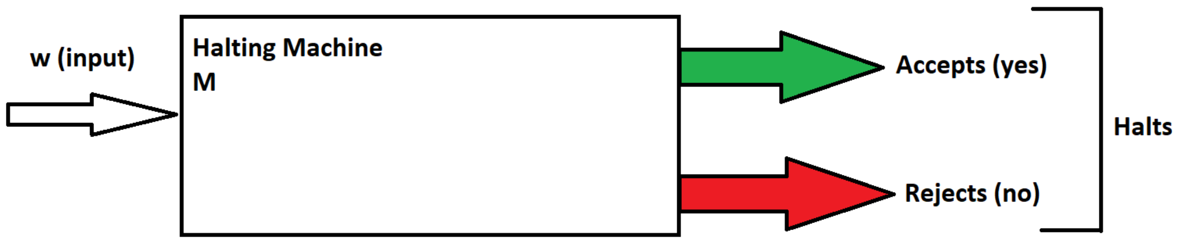
Here, we are running P2 again. This time, the value of P is P2, meaning P2 will be running the P2 program and w is P2, meaning P2 takes itself as the parameter. For this reason, the program never returns, and we can see this above as the program keeps printing "... in P2". In other words, it is recursively enumerable.

c. Program 2 is similar to a Universal Turing Machine (UTM). UTM takes as input the code for some TM M and some binary string w and accepts if and only if M accepts w. Similarly, we can see that P2 takes the following as parameters: a program P and a string w containing the input to P. P2 then return yes if and only if P exits successfully. Let us take the results from both (a) and (b) for further proof. In (a), P2 is running with P1 and 100 as parameters representing M and w for a UTM. P2 first executes itself. It then runs P1 with input 100, printing the first 100 positive integers successfully. In (a), our machine (P2) can decide whether to accept or reject P1. However, this is not the case for any P as we can see in the results obtained in (b). In (b), P2 is running with P2 and P2 as parameters again, representing M and w for a UTM. As per our conclusion in (b), this case is recursively enumerable proving that it will keep running infinitely. As per the definition, L_u is recursively enumerable meaning it is undecidable and has no guarantee of halting. It recognizes when a program can stop but cannot decide or guarantee that a given program will stop. Essentially, P2 creates a new instance each time and calls itself. Therefore, this is like a UTM as it is running its own language as input but never returning anything.

3. Halting Problem

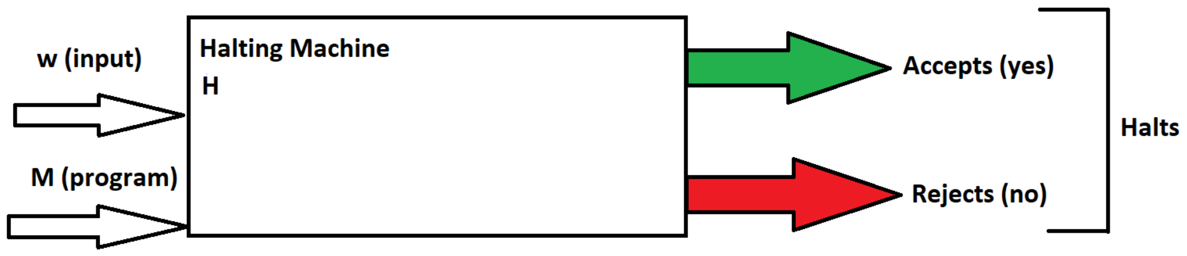
a. One of the biggest problems in logic has been understanding whether there is an automatic way to decide where given an input, we can know that it will surely give us an output. In other words, it is about finding if there is any answer for any given problem (input) that we give to our machine. H(M) or the halting problem states it cannot be decidable. To prove this, one would need to show that no possible program can give the answer that a program will halt or not; however, it is not possible to run through every program.

Suppose we have a machine M. It will take w as input and provide some output. Here, we are assuming, we can solve the halting problem. The machine will solve some problem given some input. As a result, it will say either "yes" or "no".

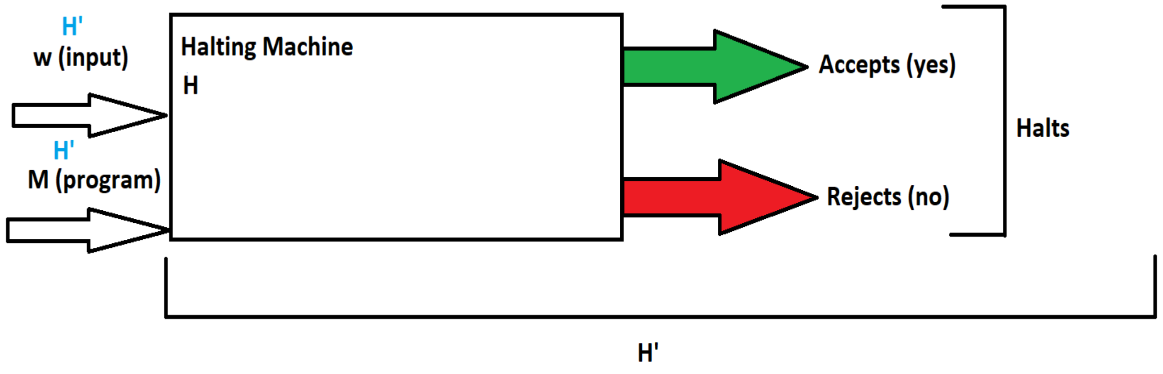


The question that we now need to ask is that will H(M) shown above, give us a result for every program that we run. In other words, telling us whether it will halt or it will not halt. Now, instead of input shown above, the machine will take two things as parameters: a program and input.

Again, we are assuming that we have a machine that solves the halting problem. More specifically, it will take a program and input giving us an answer whether it will halt or not.



If this machine outputs “yes”, we can make it loop forever and if it is “no”, then it will halt. Now let us define a formal definition of H(M). TM M is the set of inputs w such that M halts w regardless of whether M accepts or not. H(M) is the set of pairs (M, w) such that w is H(M). Let us understand what will happen if we give the whole machine defined above into itself. We will give the machine H’ as its name. Now, w will be H’ and M will be H’ and the machine is determining whether H’ will halt given H’.



b. If we run H’ on itself it leads to two cases, both of which are contradiction to our assumption:

- Case 1: H(M) = (H’, H’) => Halts. This means that our machine will loop forever as mentioned above. If H’ does halt, we get yes but then it would loop forever so it does not halt.
- Case 2: H(M) = (H’, H’) => Does not halt. If it does not halt, we get no but then it halts as defined previously.

Either way, we get a contradiction proving that H(M) is recursively enumerable. We started by assuming it can solve the halting problem but since we got a contradiction, we have an incorrect assumption. Therefore, there is no possible machine or program that can solve the halting problem which then, makes H(M) undecidable.

Now, we can see that P2 represents H(M) based on its behaviour. Both sections of 2 asked to run P2. In 2(a), P was P1 and w was 100. Here, we saw that P2 halts because it accepted P1 and the input string, w. According to the definition, H(M) halts if and only if the program accepts the input. Hence, it is an example of H(M). Similarly, for 2(b) P was P2 and w was P2. Here, we saw that P2 looped infinitely because it was unable to decide whether it should accept itself as an input or not. This resembles H(M)’s behaviour again as discussed in 3(a). We can now conclude that P2 is an undecidable problem like the halting problem.