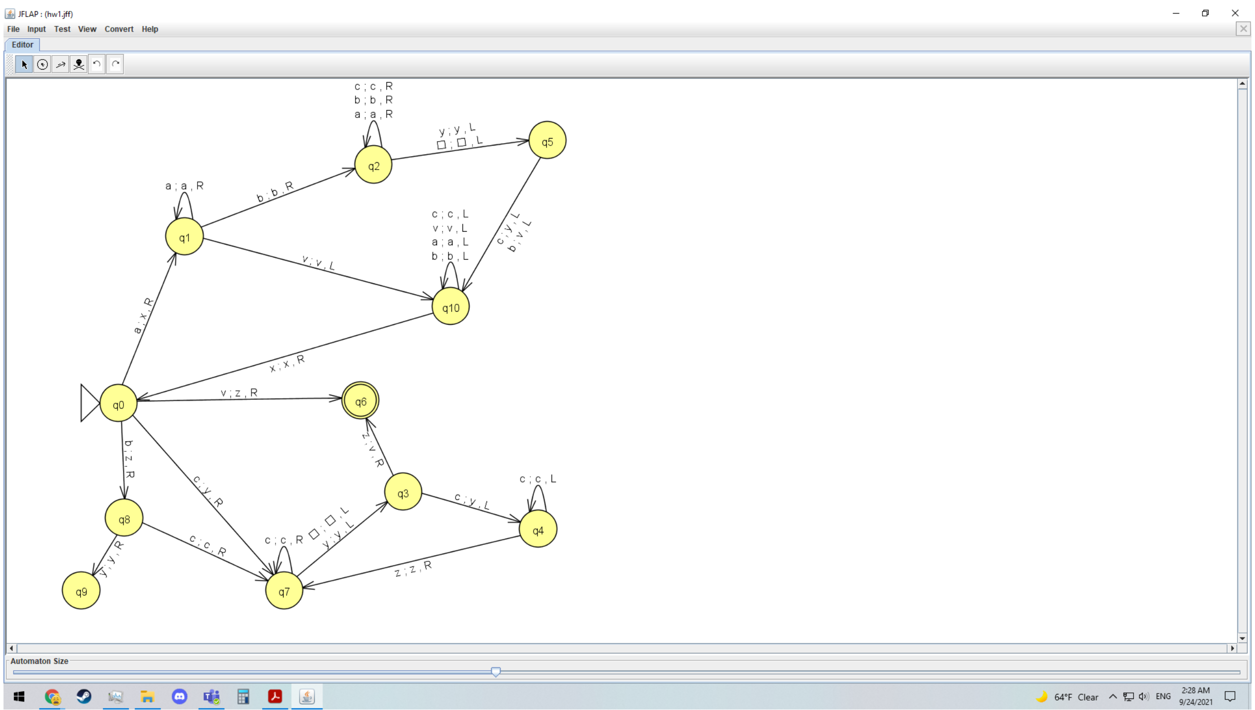
1)
a)

The formal-level specification(Not true if it is the model or formal description)

Now we give the formal description of M2 = (Q, Σ, Γ, δ, q1, qaccept, qreject):
 • Q = {q1, q2, q3, q4, q5,q6,q7,q8,q9,q10 qaccept, qreject},
• Σ = {a,b,c}
• Γ = {x,y,z,v}.
• We describe δ with a state diagram
• The start, accept, and reject states are q1, qaccept, and qreject, respectively

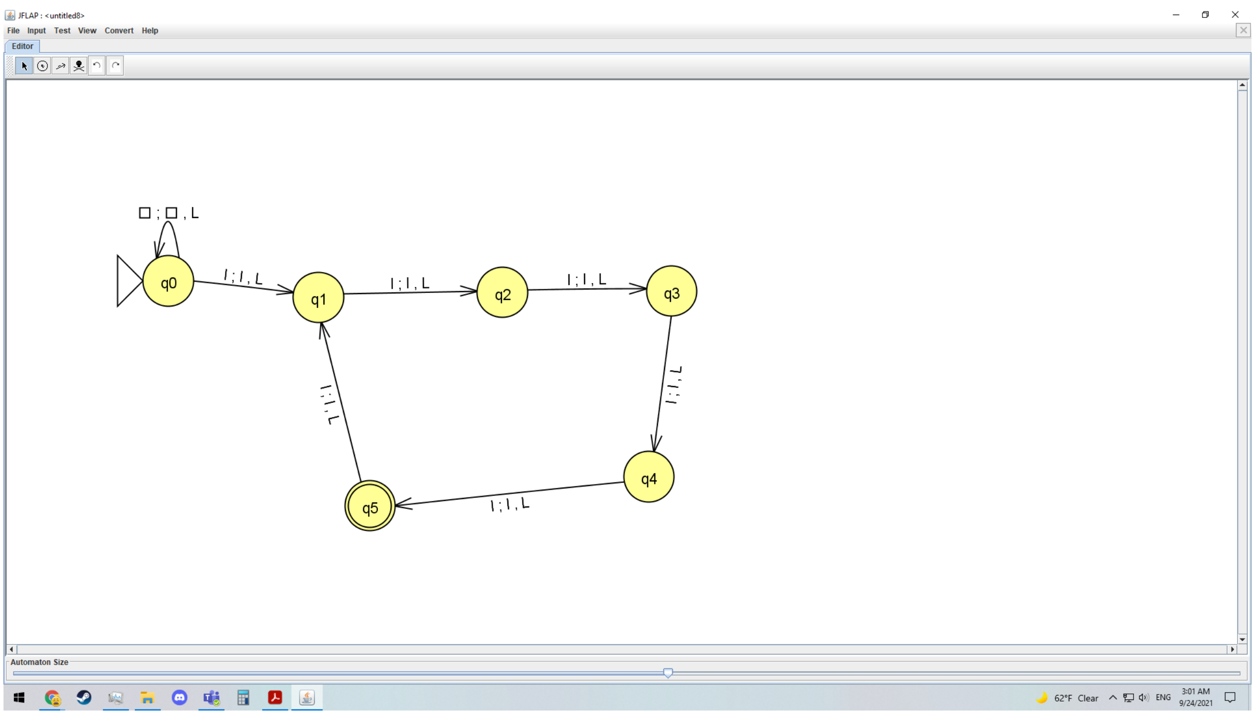


b)Basic idea of how my program works
If b is the first letter (like if there are no a) then change to z and move right
If the letter is a, then change to x and move right
If a is the letter move right along the string until you get to b (if not b it rejects) or v
If that letter is v repeat from step 2
If that letter is b move to the right
If that letter is blank then move left
If that letter is y keep moving left until you get to b or z
If that letter is c then change it to y and move to the left along the string until you get to x
If that letter is y and move to the left along the string until you get to x
If x move to the right
Repeat from step 2
x=finished a
y=finished b

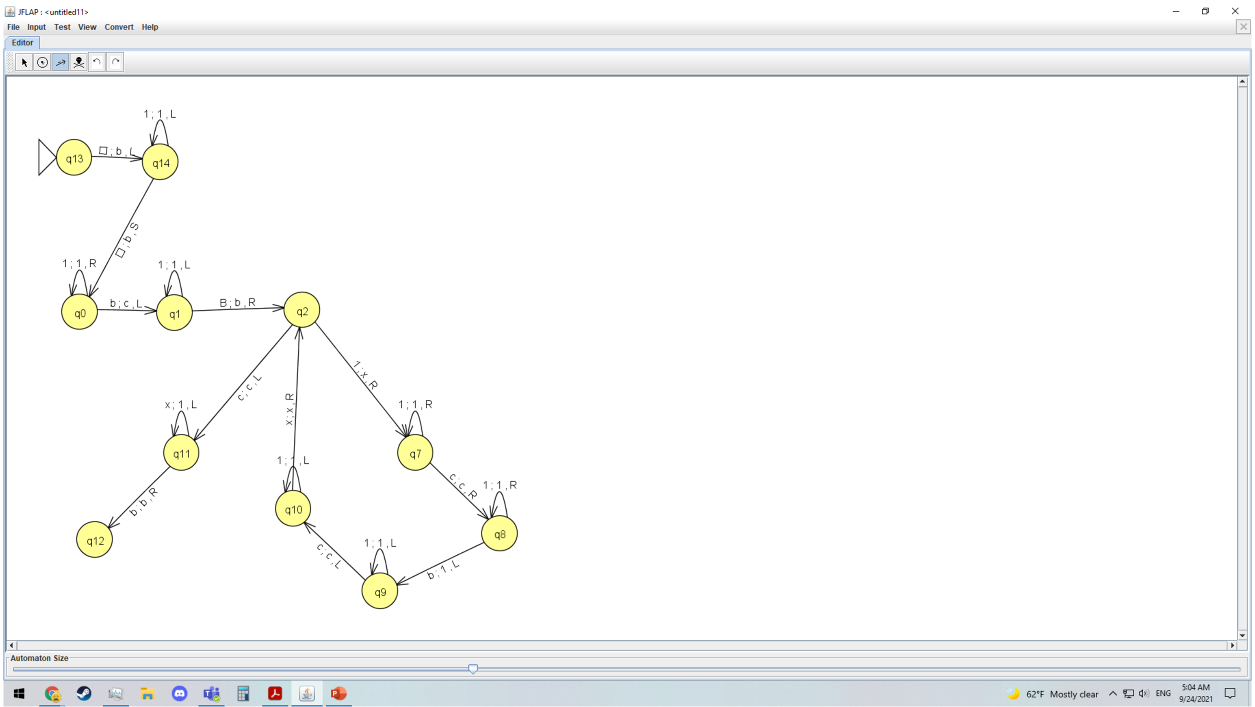
z=no a to the left
v=no c to the right

String in language
Ab (xb,xv, accept)
Bc (zc, zy, accept)
aaabc (xaabc,xaaby, xaavy,xxavy,xxxvy, accept)

String not in language
Abc, (xbc, xby, xvy, reject)
Aacb(xaacb, reject)
Aabbb(xabbb,xabbv,xxbbv,reject)

2)
a)


b) (I assume this means that the statement “given a number x in unary notation, outputs the number 2^x in unary notation” means for example given a number 3 in unary notation would be 111 would output 2^3=8 in unary notation)



3) The general idea for the Turing machine is that we check whether graph G contains all edges connecting nodes in V' such that if yes, accept, otherwise, reject."

The Turing Machine that accepts the language {< G, k > | G is an undirected graph containing a clique of size at least k} has the following steps

1. (Assuming that the input G of an undirected graph is a finite string containing all edges in some format that we know x is the first vertex and y is the second vertex) Use the first edge as our starting vertex x
2. (Assuming there is some way to mark an edge with some symbol a) Mark every adjacent edge towards the first vertex with a symbol "a"
3. Go to vertex following an edge with symbol "a"
4. (Assuming that there is enough letters in the alphabet for this turing machine) Check if there is a vertex with an edge marked with symbol "a" if there is, then the current letter is the next letter in the alphabet and we mark every adjacent edge that does not have the symbol "a" with this current letter.
5. If there isn't a vertex with an edge marked with symbol "a" then go back to the previous vertex until you find one. If you go back to vertex "a" then try with the next vertex
6. If clique is not 2 then repeat step 3, k-2 more times with the the current letter

4)

i)If R is a reducible relation on some X and x is any element of X then that means x is related to something in X, for some variable w. We can then say that xRw, which by definition of reducibility since

x is in the reduced language L' if and only if w is in some language L that would mean that xRw implies that xRx. Therefore every element is related to itself and so the relation is reflexive.
ii) If R is a reducible relation on some X and x is any element of X then that means x is related to something in X, for some variable y. We can then say that since we know that a reducibility is a reflexive relation xRx using the definition of reducibility since x is in the reduced language L' if and only if w is in some language L then that would mean that xRw and wRx. Therefore every element is mirrored and the relation is reflexive.
5.)

In order to find out if the class of Turing-acceptable languages is closed under concatenation for languages L1 and L2 such that L1L2, we must be able to divide our input w into two parts, w1w2 and be able to run a turing machine M1 that accepts any input in L1 and rejects any input not in L1 on w1 and also run a turing machine M2 that accepts any input in L2 and rejects any input not in L2 on w2. This now means that we accept if M1 and M2 accept and reject if either one of the two reject. Now for some w ∈ L1L2, we can say then that there is a splitting point in our string where we know that M1 accepts w1, and M2 accepts w2 which means our nondeterministic machine will accept this input. On the other case, if w ∉ L1L2, then that would mean there is not a way to split the string so that M1 accepts w1 and M2 accepts w2 which would mean our nondeterministic machine will reject. This shows that it is able to be a decider for the concatenation of L1 and L2.
In order to find out if the class is closed under concatenation that we will try to prove if w ∈ L1*. If w is empty then we accept and if it is not, we try to find out how to break the input into different parts so that we can run M1 on each part such that it accepts if M1 accepts every part. We can therefore say if w ∈ L1* , then that means that it is possible to break our input such that M1 will accept each part and reject otherwise. This shows that it is able to be a decider for the concatenation of L1 and L2.This shows that it is able to be a decider for the Kleene star of L1.

6.)
No, just because B is a regular language does not automatically mean that A is a regular language. For example, if we assume that the language A={a^n b^n | n ≥ 0} then since we can see that f maps A to B so that means it is a computable function. This tests whether w is of the form a^n^b n and if it outputs the string a, and if not it outputs the string b. Therefore, A ≤ B and B is a regular language and A is a nonregular language

7.)
Since we are trying to prove that S is undecidable, if we try to prove by contradiction then we assume that S is decidable, and let R be its decider for the Turing machine. We can say that S is a decider for a Turing machine A where A = {M|M is a TM and M accepts w} which means that S accepts an input (M,w), where M is a Turing machine and w is some string. We can then create a turing machine with the following steps

When given some input x:
1. If the input x has the form ab we accept
2. If the input x has the form ba then run M on w and accept if M accepts w.
3. If the input x does not have the form ab or ba then we reject

Since we can see that R will only accept when M accepts w, so then that means that S is a decider for our Turing machine A. Since we can now say that A is undecidable then since this contradicts our assumption that S is decidable then by proof of contradiction we can therefore say that S cannot be decidable.