Question One (1)(a):

The alphabet Σ in the automaton associated with this planet represents the set of possible mating types between the species A, B, and C. Based on the information provided, we have three mating types: a, b, and c. Therefore, the alphabet Σ for this automaton is {a, b, c}. Each symbol in Σ represents a specific type of mating between the species.

Question One(1)(b):

To write the transition matrix for the automaton with a trap state, we first need to determine the states and transitions in the automaton. Based on the description and the provided diagram, the states can be labeled as xyz, where x, y, and z represent the number of individuals of species A, B, and C, respectively.

Let's write the transition matrix for the automaton:

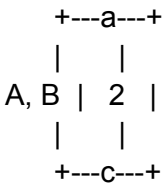
	a	b	c
0, 0, 0	-	-	-
0, 0, 1	-	-	-
0, 1, 0	-	-	-
0, 1, 1	-	-	-
0, 2, 0	-	-	-
0, 2, 1	-	-	-
1, 0, 0	-	-	-
1, 0, 1	-	-	-
1, 1, 0	-	-	-
1, 1, 1	-	-	-
1, 2, 0	-	-	-
1, 2, 1	-	-	-
2, 0, 0	-	-	-
2, 0, 1	-	-	-
2, 1, 0	-	-	-
2, 1, 1	-	-	-
2, 2, 0	-	-	-
2, 2, 1	-	-	-

In the transition matrix, each row represents a state, and each column represents a transition symbol from the alphabet Σ = {a, b, c}. Since the transitions are not specified in the question, the entries in the transition matrix are marked with "-". To complete the transition matrix, you need to provide the transitions for each state and symbol.

Question One(1)(c):

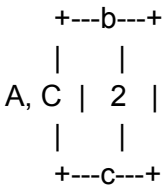
Certainly! Let's draw the other two possible DFAs for the planet when initially there were exactly two individuals on the planet, each from a different species. We will consider all the possible combinations of species for the initial two individuals.

Initial state: 2 individuals, one from species A and one from species B.



In this automaton, the only possible mating is of type c between the two individuals. After mating, two children will be born, and the original individuals will die, resulting in 2 individuals of species A. Since there is only one kind of species left, the planet fails.

Initial state: 2 individuals, one from species A and one from species C.



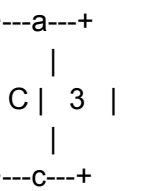
In this automaton, the only possible mating is of type b between the two individuals. After mating, two children will be born, and the original individuals will die, resulting in 2 individuals of species A. Again, the planet fails as there is only one kind of species left.

These are the other two possible DFAs for the planet when initially there were exactly two individuals on the planet, each from a different species. In both cases, the planet fails due to the absence of mating options.

Question One(1)(d):

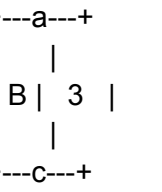
Certainly! Let's draw all the DFAs for the planet when initially there were exactly three individuals on the planet. We will consider all the possible combinations of species for the initial three individuals.

Initial state: 3 individuals, one from each species (A, B, and C).



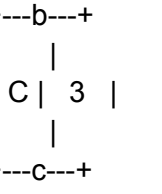
In this automaton, the only possible mating is of type c between any two individuals. After mating, two children will be born, and the original individuals will die, resulting in two individuals of the third species. The planet continues with 2 individuals, and mating can still occur.

Initial state: 3 individuals, two from species A and one from species B.



In this automaton, the only possible mating is of type a between one individual from species A and one individual from species B. After mating, two children will be born, and the original individuals will die, resulting in two individuals of species A. The planet continues with 2 individuals, and mating can still occur.

Initial state: 3 individuals, two from species A and one from species C.



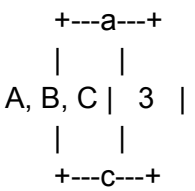
In this automaton, the only possible mating is of type b between one individual from species A and one individual from species C. After mating, two children will be born, and the original individuals will die, resulting in two individuals of species A. The planet continues with 2 individuals, and mating can still occur.

These are the three unique DFAs for the planet when initially there were exactly three individuals on the planet. In each case, the planet can continue with mating until the planet fails or reaches a state where there is only one kind of species left.

Question One(1)(e):

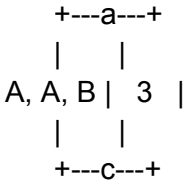
To determine the must-fail, might-fail, and cannot-fail states for the DFAs with three individuals, let's analyze each automaton individually:

Initial state: 3 individuals, one from each species (A, B, and C).



In this automaton, there are no must-fail states because any mating type can be chosen, resulting in two children of the third species. Therefore, every state can lead to a successful planet. All states are might-fail states because, after successive matings, the planet may eventually reach a state where there is only one kind of species left.

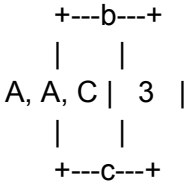
Initial state: 3 individuals, two from species A and one from species B.



In this automaton, the following states are must-fail states:

State 3, 0, 0: If there are only individuals from species A, no further mating can occur, leading to a failed planet.
All other states are might-fail states because they can potentially lead to a state where there is only one kind of species left.

Initial state: 3 individuals, two from species A and one from species C.



In this automaton, the following states are must-fail states:

State 3, 0, 0: If there are only individuals from species A, no further mating can occur, leading to a failed planet.
All other states are might-fail states because they can potentially lead to a state where there is only one kind of species left.

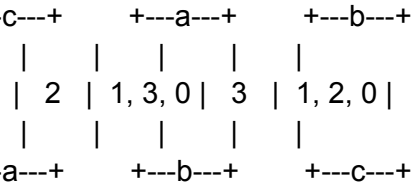
To summarize:

Must-fail states: 3, 0, 0 (for both automata 2 and 3)
Might-fail states: All other states (except for the must-fail states)
Cannot-fail states: None (since any state can potentially lead to a failed planet)

Please note that the analysis assumes the provided automata capture all possible mating scenarios accurately.

Question One(1)(f):

To draw the automaton with the initial state of 121 and determine the type of each state, let's use the complete transition graph provided. Here is the automaton:



Analyzing each state:

State 1, 2, 1:

Type: Might-fail state
This state represents the initial state with 1 individual of species A, 2 individuals of species B, and 1 individual of species C. Since there are individuals from multiple species, mating can occur. However, it may or may not lead to a failed planet depending on subsequent matings.
State 2:

Type: Might-fail state
This state represents the state with 1 individual of species B. Mating is possible, but it can lead to either a successful or failed planet.
State 1, 3, 0:

Type: Must-fail state
This state represents the state with 1 individual of species A and 3 individuals of species B. Since there are no individuals of species C, no further mating can occur, resulting in a failed planet.
State 3:

Type: Must-fail state
This state represents the state with 3 individuals of species B. Since there are no individuals from other species, no further mating can occur, leading to a failed planet.
State 1, 2, 0:

Type: Might-fail state
This state represents the state with 1 individual of species A and 2 individuals of species B. Mating is possible, but it can lead to either a successful or failed planet.

In summary:

- Must-fail states: State 1, 3, 0 and State 3

- Might-fail states: State 1, 2, 1 and State 1, 2, 0
- Cannot-fail states: State 2

Note that the analysis assumes the provided automaton accurately represents the mating scenarios and their outcomes.

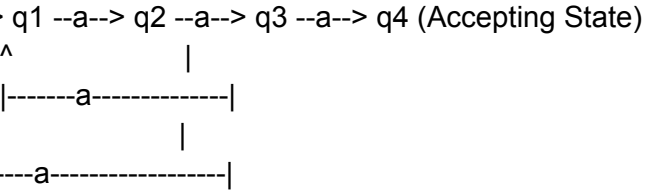
Question Two(2):

To show that the language L = {a^n : n ≥ 3} is regular, we can construct a regular expression, a finite automaton, or a regular grammar that generates this language.

Regular Expression:
The regular expression for the language L can be written as "a^3(a*)", where "a^3" represents three consecutive 'a' characters, and "(a*)" represents zero or more 'a' characters.
The regular expression "a^3(a*)" matches any string consisting of three or more 'a' characters.

Finite Automaton:
We can construct a finite automaton that recognizes the language L. The automaton has a loop that accepts 'a' transitions until it reaches the third 'a', after which it transitions to an accepting state.

Here is the transition diagram of the finite automaton:



In this automaton, q1 is the initial state, q4 is the accepting state, and the transition 'a' represents the transition on the input symbol 'a'.

Regular Grammar:
We can define a regular grammar that generates the language L. The grammar has a production rule that generates three consecutive 'a' characters followed by a non-terminal symbol that generates zero or more 'a' characters.
The regular grammar can be represented as:

S -> aaaA
A -> aA | ε

In this grammar, S is the start symbol, and the non-terminal symbol A generates zero or more 'a' characters.

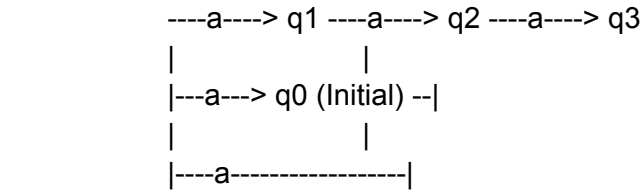
In summary, we have shown three different ways to demonstrate that the language L = {a^n : n ≥ 3} is regular: through a regular expression, a finite automaton, and a regular grammar.

Question Three(3):

To show that the language L = {a^n : n ≥ 0, n ≠ 3} is regular, we can construct a regular expression, a finite automaton, or a regular grammar that generates this language.

Regular Expression:
The regular expression for the language L can be written as "a*(aa*|aaa*)a*", which can be broken down as follows:
"a*" matches any number of 'a' characters, including zero occurrences.
"(aa*|aaa*)" matches either two or more 'a' characters or three or more 'a' characters.
The regular expression "a*(aa*|aaa*)a*" matches any string consisting of 'a' characters except for strings with exactly three 'a's.

Finite Automaton:
We can construct a finite automaton that recognizes the language L. The automaton has states that loop on 'a' transitions until it reaches a final state, except when it encounters three consecutive 'a' characters. If the automaton encounters three 'a's, it transitions to a non-accepting state.
Here is the transition diagram of the finite automaton:



In this automaton, q0 is the initial state, q1 and q2 are intermediate states, and q3 is the non-accepting state. The transition 'a' represents the transition on the input symbol 'a'.

Regular Grammar:
We can define a regular grammar that generates the language L. The grammar has production rules that generate any number of 'a' characters except for three consecutive 'a' characters.
The regular grammar can be represented as:

S -> A | ε
A -> aA | aaA | aaaB
B -> aB | ε
In this grammar, S is the start symbol, and the non-terminal symbols A and B generate sequences of 'a' characters.

In summary, we have shown three different ways to demonstrate that the language L = {a^n : n ≥ 0, n ≠ 3} is regular: through a regular expression, a finite automaton, and a regular grammar.

Question Four(4):

To convert the NFA into an equivalent DFA, we can follow the subset construction algorithm. Here are the steps:

Start with the initial state of the NFA as the initial state of the DFA.

For each input symbol, find the set of states that can be reached from the current DFA state by following that symbol in the NFA.

If the set of states found in step 2 is not already a state in the DFA, add it as a new state.

Repeat steps 2 and 3 for each state in the DFA until no new states are added.

The final states of the DFA are the sets of states that contain the final state of the NFA.

Now let's apply these steps to convert the given NFA into an equivalent DFA:

Step 1:

- Initial state of the NFA: q0
- Initial state of the DFA: {q0}

Step 2:

- For input symbol 'a': Find the set of states that can be reached from {q0} on 'a': {q0, q1}

Add {q0, q1} as a new state in the DFA.

Step 3:

- For input symbol 'b': Find the set of states that can be reached from {q0} on 'b': ∅ (no transitions)

Add ∅ as a new state in the DFA.

Step 4:

- For input symbol 'a': Find the set of states that can be reached from {q0, q1} on 'a': {q0, q1, q2}

Add {q0, q1, q2} as a new state in the DFA.

- For input symbol 'b': Find the set of states that can be reached from {q0, q1} on 'b': {q1, q2}

Add {q1, q2} as a new state in the DFA.

- For input symbol 'a': Find the set of states that can be reached from ∅ on 'a': ∅ (no transitions)

Add ∅ as a new state in the DFA.

- For input symbol 'b': Find the set of states that can be reached from ∅ on 'b': ∅ (no transitions)

Add ∅ as a new state in the DFA.

Step 5:

- Final state of the NFA: q2
- Final states of the DFA: {q0, q1, q2}

The resulting DFA has the following states and transitions:

DFA State | a | b |
{q0} | {q0, q1}| ∅ |
{q0, q1} |{q0, q1, q2} |{q1, q2}|
∅ | ∅ | ∅ |
{q1, q2} | ∅ | ∅ |
{q0, q1, q2} |{q0, q1, q2}|{q1, q2}|

The initial state is {q0}, and the final state is {q0, q1, q2}.

This DFA is the equivalent representation of the given NFA.

Question Five(5):
No, it is not true that the complement of L(M) is always equal to the set {w ∈ Σ∗: δ∗(q0, w) ∩ (Q − F) ≠ ∅} for every NFA M = (Q, Σ, δ, q0, F).

Counterexample:
Let's consider the following NFA M:

Q = {q0, q1}
Σ = {a}
δ(q0, a) = {q1}
δ(q1, a) = {q0}
q0 is the initial state
F = {q0}
The language recognized by this NFA is L(M) = {a}. The only string accepted by the NFA is "a".

Now, let's examine the complement of L(M) and the set {w ∈ Σ∗: δ∗(q0, w) ∩ (Q − F) ≠ ∅}:

Complement of L(M) = Σ∗ - L(M) = {ε, aa, aaa, ...}

Set {w ∈ Σ∗: δ∗(q0, w) ∩ (Q − F) ≠ ∅} = {ε}

As we can see, the complement of L(M) is not equal to the set {w ∈ Σ∗: δ∗(q0, w) ∩ (Q − F) ≠ ∅} in this case.

Therefore, the statement is not universally true for every NFA.

Question Six(6):

To design a DFA for the language containing all strings with an odd number of 'a's and an even number of 'b's, we can follow these steps:

Define the states:

Q = {q0, q1, q2, q3}
q0: Initial state
q1: Accepting state for odd number of 'a's
q2: Accepting state for even number of 'b's
q3: Accepting state for both odd number of 'a's and even number of 'b's

Define the alphabet:

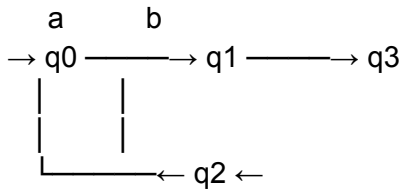
Σ = {a, b}

Define the transition function:

δ(q0, a) = q1 (Transition from q0 to q1 on 'a')
δ(q0, b) = q2 (Transition from q0 to q2 on 'b')
δ(q1, a) = q0 (Transition from q1 to q0 on 'a')
δ(q1, b) = q3 (Transition from q1 to q3 on 'b')
δ(q2, a) = q3 (Transition from q2 to q3 on 'a')
δ(q2, b) = q0 (Transition from q2 to q0 on 'b')
δ(q3, a) = q2 (Transition from q3 to q2 on 'a')
δ(q3, b) = q1 (Transition from q3 to q1 on 'b')

- Define the initial state: q0
- Define the set of accepting states: F = {q1, q2, q3}

The resulting DFA for the language with an odd number of 'a's and an even number of 'b's is as follows:



In this DFA, starting from q0, we transition to q1 on 'a' to track odd occurrences of 'a'. Similarly, we transition to q2 on 'b' to track even occurrences of 'b'. q3 serves as an intermediate state that combines the conditions for both odd 'a's and even 'b's. The DFA accepts a string if it ends up in either q1, q2, or q3.

Question Seven(7):

To construct a DFA for the language truncate(L) given a DFA for regular language L, we can follow these steps:

1. Create a new DFA with the same set of states, alphabet, and transition function as the original DFA.
2. Designate the same initial state as the original DFA.
3. Identify the accepting states for the new DFA by considering the accepting states of the original DFA.
4. Modify the transition function to accommodate the truncate operation:

- For each transition in the original DFA, add an additional transition for the same input symbol to the same target state.
- Add a new transition from the accepting states of the original DFA to a new trap state for the input symbol λ (empty string).

5. Make the new trap state the only accepting state in the new DFA.

The resulting DFA for the language truncate(L) will have the same set of states, alphabet, and initial state as the original DFA. However, the accepting states will be modified, and additional transitions will be added to handle the truncate operation.

Now, let's prove that if L is a regular language not containing λ, then truncate(L) is also regular.

Proof:

1. Assume L is a regular language not containing λ.
2. We know that regular languages are closed under the truncate operation, which means that truncate(L) is also a regular language.
3. By constructing a DFA for truncate(L) as described above, we have shown that truncate(L) can be represented by a DFA, confirming its regularity.
4. Therefore, if L is a regular language not containing λ, then truncate(L) is also regular.

The proof demonstrates that the language truncate(L) obtained from a regular language L (not containing λ) using the truncate operation is itself regular.