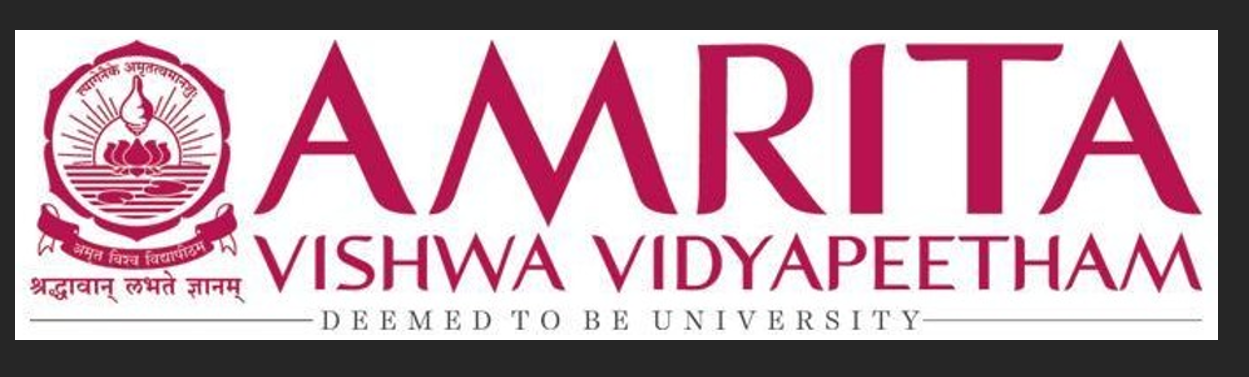
****

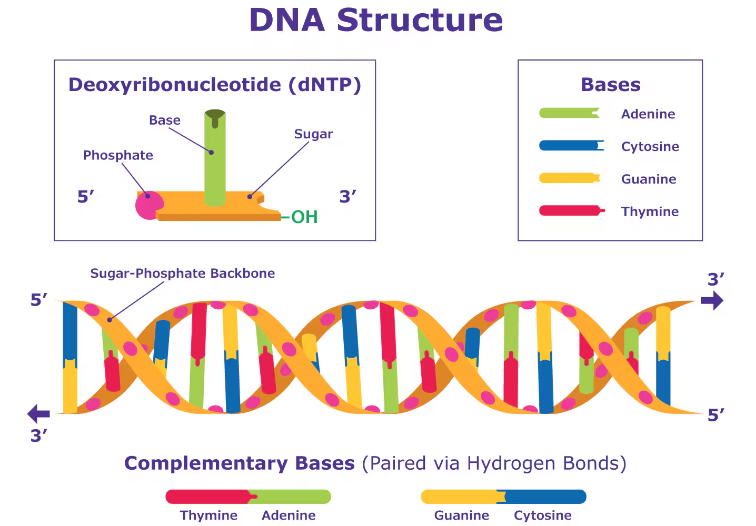
**Department of Computer Science and Engineering**

**19CSE214 – Theory of Computation**

**Case Study Report**

| **S.NO** | **Roll No** | **Name** |
| --- | --- | --- |
| **1** | **CB.EN.U4CSE22536** | **PRANAV PS** |
| **2** | **CB.EN.U4CSE22540** | **S.PRANAY** |
| **3** | **CB.EN.U4CSE22553** | **T.BHUVANA** |
| **4** | **CB.EN.U4CSE22556** | **R.RINITHA** |

**TOPIC : DNA SEQUENCE ANALYSIS**

****

**Problem statement:**

**Objective:**

The goal of this project is to construct a Context-Free Grammar (CFG), a Context-Free Language (CFL), and a Non-deterministic Pushdown Automaton (NPDA) to validate DNA sequences based on specific formal language rules.  
 **Preface and Analysis of the problem statement:**

DNA sequences are composed of four nucleotide bases: adenine (A), thymine (T), cytosine (C), and guanine (G). These bases pair specifically to form complementary strands, with adenine always pairing with thymine (A-T) and cytosine always pairing with guanine (C-G). This pairing rule is fundamental to the structure of DNA and ensures that each strand of a DNA molecule can serve as a template for the formation of a complementary strand.

In the context of formal language theory, we can model DNA sequences using specific patterns that conform to the biological rules of base pairing. By defining a formal language, we can generate and validate sequences based on predefined patterns. This approach can help in understanding and simulating various biological processes and can also be used for practical applications such as DNA sequence analysis and synthesis.

For this project, the language will consist of sequences formed by repeating specific substrings, which represent segments of DNA. Each substring will correspond to a specific pattern of nucleotides. Additionally, the complementary strand of a given sequence must also fit certain patterns, ensuring that the base-pairing rules are respected.

**Alphabet Set (∑):**

A**:**CGCG - DNA sequence

B:ATAT -DNA sequence

C:CGCGATAT - DNA sequence

**Language (L):**

L = {a^n b ^m c ^n +m : n, m belongs to N}

The language helps in modelling DNA sequences with specific repetitive structures, ensuring that both the original and complementary strands adhere to defined patterns. This allows us to capture the repetitive and structured nature of DNA sequences in a formal language framework, which is useful for theoretical analysis and practical applications in computational biology.  
  
Breaking down how the formal language works:

**L = {a^n b ^m c ^n +m : n, m ∈ N}**

can be used to represent a DNA sequence and its complementary strand.

Formal Language Representation

DNA Sequence Components:

𝑎

a- Represents a specific substring, for example, "CGCG".

𝑏

b- Represents another specific substring, for example, "ATAT".

𝑐

c- Represents a combination of the substrings for 𝑎 and 𝑏, for example, “CGCGATAT".

**Language Definition:**

The language 𝐿 defines sequences of the form:

**L = {a^n b ^m c ^n +m : n, m ∈ N}**

a^n means the substring 𝑎 repeated 𝑛 times.

b^n means the substring b repeated 𝑚 times.

c ^(n +m) means the combined substring 𝑐 repeated (𝑛+𝑚) times.

**DNA Sequence Example**

Suppose we have the following assignments for the substrings:

𝑎="CGCG"

𝑏="ATAT"

𝑐="CGCGATAT"

A valid DNA sequence in this language might be constructed as follows:

𝑎^2="CGCGCGCG" (two repetitions of "CGCG")

𝑏^1="ATAT" (one repetition of "ATAT")

𝑐^(2+1)="CGCGATATCGCGATATCGCGATAT"" (three repetitions of “CGCGATAT")

Putting it together:

Sequence:

𝑎^2^𝑏^1^𝑐^3="CGCGCGCGATATCGCGATATCGCGATATCGCGATAT"

Input tape word:

*aabccc*

**Complementary Strand:**

To ensure the complementary strand also fits a specific pattern, we need to validate it similarly:

Complementary bases:

C↔G

G↔C

A↔T

T↔A

Given the sequence 𝑎^2^𝑏^1^𝑐^3:

**Original sequence:** "CGCGCGCGATATCGCGATATCGCGATATCGCGATAT"

**Complementary strand:** "GCGCGCGCTATAGCGCTATAGCGCTATAGCGCTATA"

The complementary strand should follow the same structural rules. If the original sequence fits the pattern 𝑎^𝑛𝑏^𝑚c^𝑛+𝑚 , its complementary strand should be a valid sequence composed of complementary bases.

**Explaining the Pattern :**

The pattern **a^n b ^m c ^n +m** is chosen to enforce a relationship between the segments of the DNA sequence:

**a^n**: Ensures that a segment (e.g., "CGCG") is repeated

**b ^m**: Ensures that another segment (e.g., "ATAT") is repeated

**c ^n +m**: Ensures that a combined segment (e.g., "CGCGATAT") is repeated (n+m) times,which ties the repetition counts of a and b together.

This structure can be used to represent various biological or computational properties of DNA sequences, such as specific binding sites or regions of interest that must appear in a particular order and frequency.

In the context of the formal language

**L = {a^n b ^m c ^n +m : n, m ∈ N}** where:

a could be mapped to "CGCG" (a 4-nucleotide sequence),

b could be mapped to "ATAT" (a 4-nucleotide sequence),

c could be mapped to "CGCGATAT" (an 8-nucleotide sequence, combining one instance of

a and b)

**PDA – PushDown Automata:**

**A Brief intro of theory:**

Context Free Language is visualised in Push Down Automata (PDA). PDA. It comes with an extra stack memory. This extends the capabilities of the Finite Automata.

PDA comes with Seven Tuples (Q, ∑, Γ, δ, q0, z, F):

Q = Finite set of States

∑ = Finite set of Input Symbols

Γ = Finite Stack Alphabets

δ = Transition function

q0 = Initial State

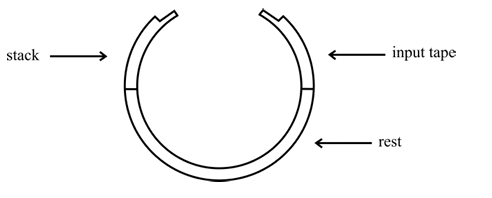
z = Initial Stack Symbol

F = Set of Final States

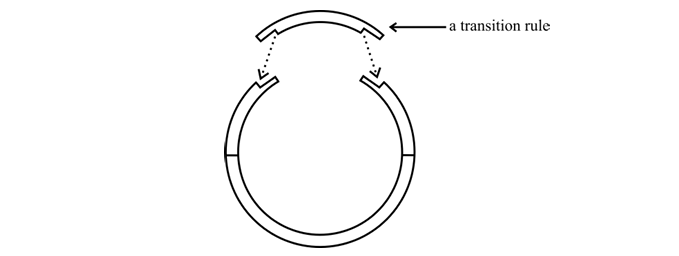
**The implementation of PDA** :

The basic elements of PDA

the input tape and the stack are represented in the same circular dsDNA molecule which one end represents the stack and the second one the input word

The sticky end of the stack represents the top symbol on the stack and 

the sticky end of the input tape represents the first symbol of the input word (to be read) and simultaneously the state of PDA. The transition rules of PDA are suitable DNA molecules which hybridise to both ends of the circular DNA.



In this process, restriction enzymes cut a circular DNA molecule after it has been modified by ligation. Here's a simplified explanation:

1.Initial Setup: A circular DNA molecule is modified by adding a transition molecule.

2.Restriction Enzyme Action: The restriction enzyme (BglI) will only cut the DNA if the transition molecule is attached at both ends of the circle. BglI recognizes two specific sites, which are only present when the transition molecule is attached.

3.Cutting and Modification: After BglI cuts the DNA, other molecules and enzymes modify the DNA sequence, updating the stack and the input word.

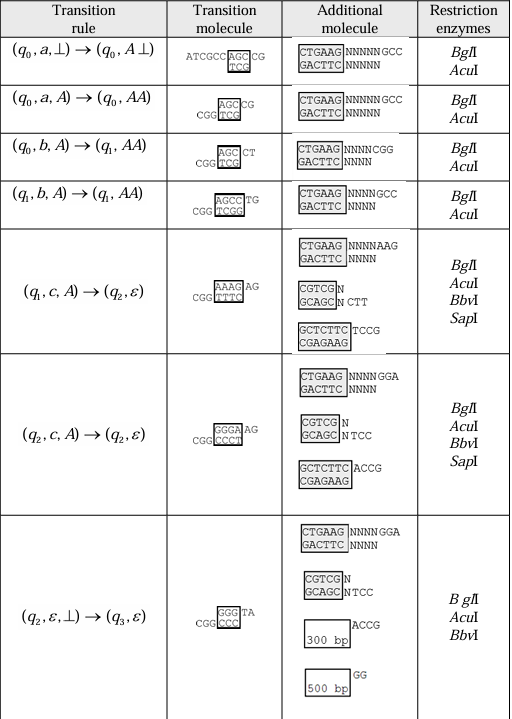
4.Next Transition: This process repeats for each transition rule.

Final State: If the sequence of transitions leads to a final state, a long

DNA molecule attaches to the DNA. This can be detected using gel electrophoresis, indicating the word is accepted.

There are set of transition rules to be applied before finding out the

PDA:



**Basic Idea for NPDA:**

1.**Input DNA Sequence:** Push the DNA sequence onto the stack.

2.**Generate Complementary Strand**: Pop the sequence from the stack, generating the complementary strand.

3.**Validate the Sequence:** Check if the complementary strand matches the expected pattern or rules.

To justify the use of a Non-deterministic Pushdown Automaton (NPDA) in the context of DNA sequence analysis, let us consider how DNA sequences can be modelled and validated using the formal language **L = {a^n b ^m c ^n +m : n, m ∈ N}** and why a DPDA is not suitable for this task.

**Approach:**

1. Push a's:

- For each a encountered in the input tape, push A onto the stack and transition to state q1.

- Transition: (q0, a, z) -> (q1, A, z)

2. Push b's:

- For each b encountered in the input tape, push B onto the stack and remain in state q1.

- Transition: (q1, b, A) -> (q1, B, A)

3. Pop c's and compare with a's and b's:

- For each c encountered in the input tape:

- Pop from the stack and transition to state q2.

- Repeat this process until the stack is empty.

- At the end of the input tape, if the stack is empty, transition to the final state q3; otherwise, reject.

- Transition: (q1, c, B) -> (q2, ε, ε)

4. Acceptance:

- If the NPDA reaches the final state q3 and the stack is empty, accept the input string; otherwise, reject.

**L = {a^n b ^m c ^n +m : n, m ∈ N}.**

**DNA Sequence Analysis and Language** **L = {a^n b ^m c ^n +m : n, m ∈ N}**

**DNA Structure and Sequence Patterns:**

DNA sequences are composed of nucleotide bases:

- Adenine (A)

- Thymine (T)

- Cytosine (C)

- Guanine (G)

In DNA analysis, specific patterns within sequences are crucial. For example, certain regions may consist of repetitive sequences, and their arrangement can be significant for biological functions.

**Mapping DNA Sequences to Formal Language:**

Consider the formal language ( L ) where:

- (a = "CGCG") (a specific pattern in the DNA sequence)

- (b = "ATAT") (another specific pattern)

- (c = "CGCGATAT") (a combination of the above patterns)

A sequence in ( L) might look like:

- ( a^n = {"CGCG"}^n )

- ( b^m ={"ATAT"}^m )

- ( c^{n+m} = {"CGCGATAT"}^{n+m} )

For example, for ( n = 2 ) and ( m = 1 ), a sequence would be:

- ( a^2 ={"CGCGCGCG"})

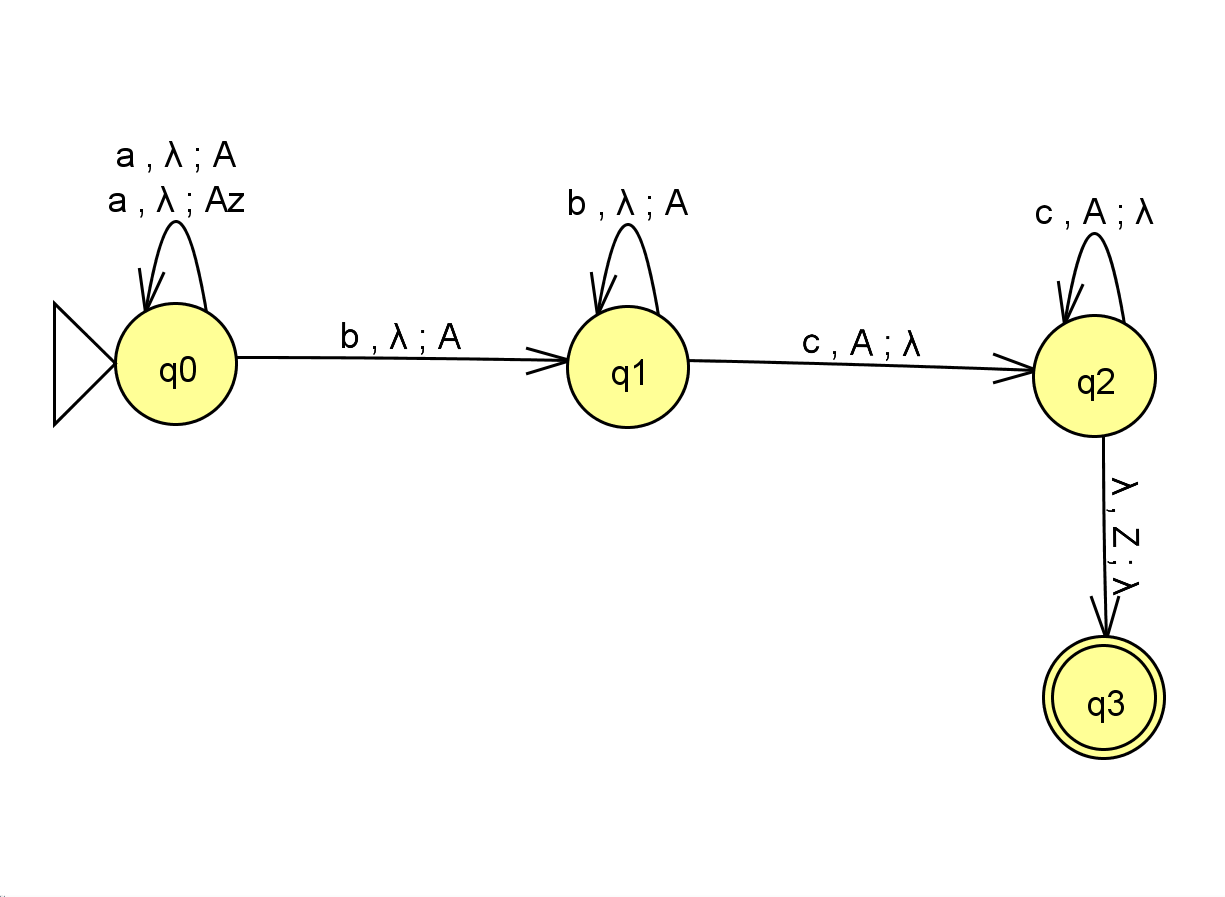
- ( b^1 = {"ATAT"} )

- ( c^3 = {"CGCGATATCGCGATATCGCGATAT"} )

**Complementary Strands:**

The complementary strand of a DNA sequence must also fit specific patterns, ensuring the sequences respect base-pairing rules (A pairs with T, C pairs with G).

Using JFLAP - Tool(NPDA)



STRINGS:

i)The accepting strings are

ac,bc,abcc,abbccc,aabbbccccc,abbbbbbccccccc ……………………. a^n b^m c^m+n

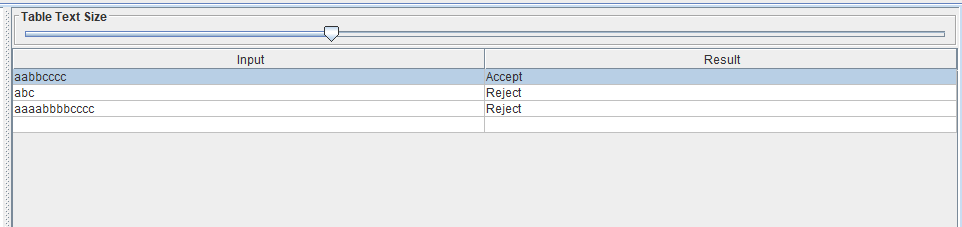
We use 'c' to balance 'a', ensuring that when 'a' or 'b' is pushed onto the stack, a complementary strand results. If 'a' appears alone, 'c' is utilized to find the correct match. Similarly, when 'b' appears alone, 'c' serves the same purpose.

If both 'a' and 'b' are present to join the DNA sequence, we first complement them and find their match using 'c'. This ensures that the sequences of 'a' and 'b' can be correctly aligned and joined.

ii)Non accepting strings :

A,b,c,aab,aabc,abc,bbc,aac,aaaaaaaaaaac,abcccc, …… != a^nb^mc^m+n

If we push 'a' alone onto the stack without a corresponding match, we can't join the sequence, leading to a non-accepting state. Additionally, if there are unequal numbers of c’s with respect to 'a's and 'b's we won't be able to find the correct pairs for the complementary strands. Consequently, the DNA sequence cannot be completed, resulting in a non-accepting state.



**Why NPDA is Suitable:**

**Handling Complex Patterns**.

The language ( L) requires the automaton to:

1. Count the number of ( a)'s.

2. Count the number of ( b)'s.

3. Ensure the number of ( c )'s is the sum of ( a )'s and ( b)'s.

This involves:

- Pushing ( a )'s onto the stack while reading them.

- Pushing ( b )'s onto the stack while reading them.

- Popping elements for each ( c ) read and ensuring the stack is empty at the end.

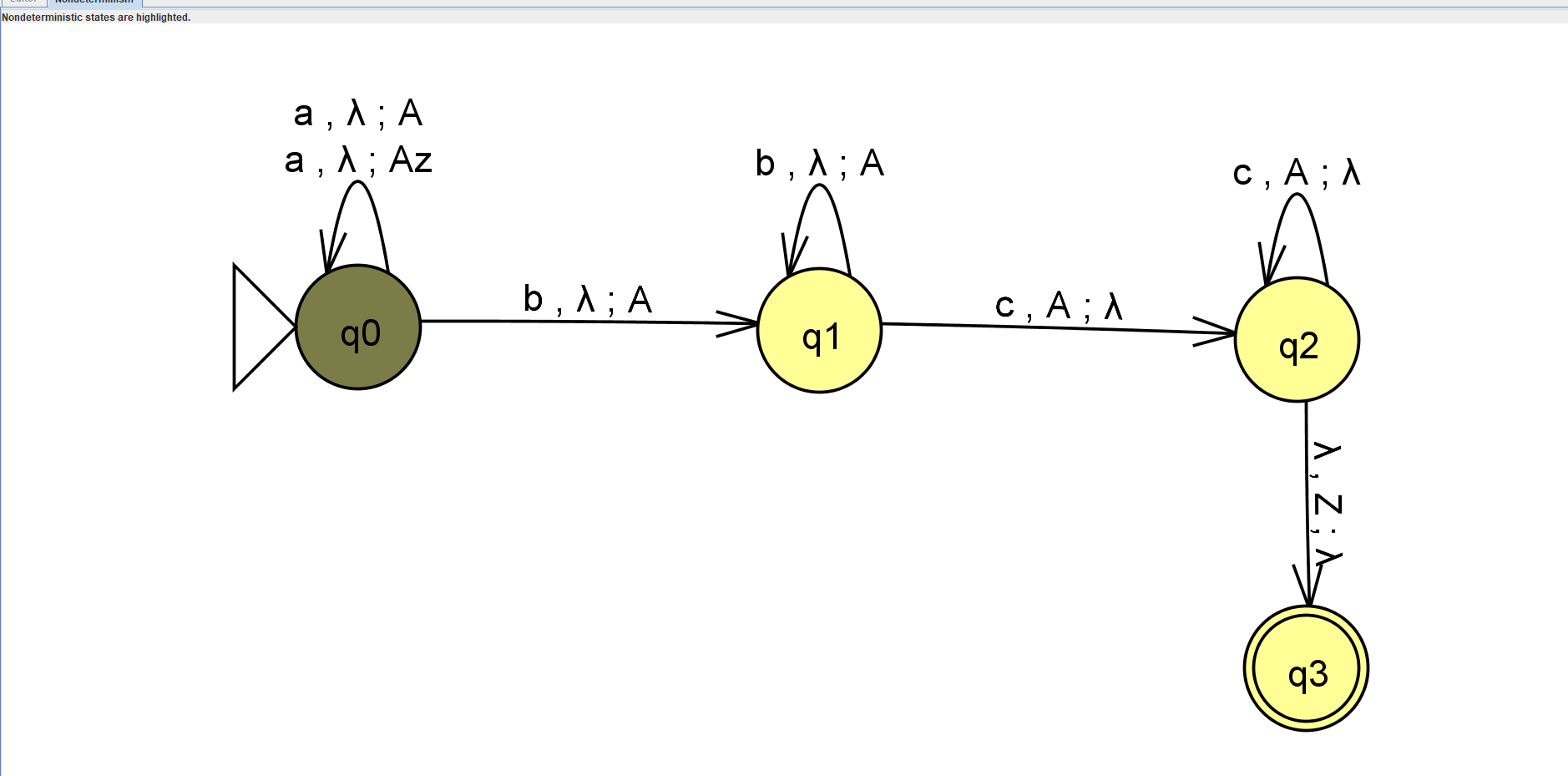
An NPDA can handle this by nondeterministically guessing when the transition from reading ( a )'s to ( b)'s and then to (c)'s occurs.

**Practical Justification in DNA Analysis:**

1. **Recognition of Repetitive Patterns:** In DNA analysis, recognizing repetitive patterns and their correct order is crucial for identifying regions like microsatellites or other repetitive elements. The language L helps model these patterns.

2. **Complex Validation:** The NPDA can validate sequences where the number of repetitions of one pattern(e.g.,a)must be matched with another pattern (e.g.,c). This is analogous to ensuring that certain genetic markers or regions are present in the correct quantities relative to each other.

3. **Nondeterminism for Flexibility**: Biological sequences can vary, and the nondeterministic nature of NPDA allows it to handle variations and uncertainties in sequence patterns more effectively than a DPDA.



We highlighted the non deteministic state in the tool

**Why DPDA is Not Suitable:**

1. **Determinism Limitation**: A DPDA must make decisions based solely on the current state and input symbol, without the ability to backtrack or guess. This is insufficient for languages where validation requires ensuring a balance between different parts of the sequence(e.g., **a^n and c ^n +m)**

2. **Stack Management:** A DPDA would struggle to manage the stack deterministically while transitioning between counting a’s and b's and c’s, particularly because it cannot "remember" the counts in a deterministic manner.

**Conclusion:**

In DNA sequence analysis, the ability to model and validate sequences with complex interdependencies (as described by the language **L = {a^n b ^m c ^n +m : n, m ∈ N}** is crucial. An NPDA is well-suited for this task due to its nondeterministic capabilities, allowing it to effectively handle the necessary pattern recognition and validation. A DPDA, constrained by its deterministic nature, lacks the flexibility to manage such complexity, making NPDA the appropriate choice for modelling and analyzing DNA sequences in this context.