# (DNA#): Molecular Biology Computation Language Proposal

Aalhad Patankar, Min Fan, Nan Yu, Oriana Fuentes, Stan Peceny
{ap3536, mf3084, ny2263, oif2102, skp2140} @columbia.edu

**Motivation**

Inspired by the parallelism between genetic code and computer code, we would like to provide a platform to "code" genes easily and natively. We are implementing basic models of molecular biology in light of heightened interest in understanding genetics and its potential impact on tailoring medicine, understanding diseases and ultimately improving human life. We are designing DNA# for both the novice user, who is interested in learning the basics of genetic code, and the advanced researcher performing analyses on large data sequences. We would like to rethink genetic code as a form of data representation itself, and provide coders a platform to tinker with genes and clearly see the biological results without hours of laborious manual transcription, complement finding, and referencing external resources. In sum, we would like to create a language for programmers to code in the genetic language and learn synthetic biology, and for synthetic biologists with limited programming experience to open source and optimize their work.

**Summary of goals**

- Provide basic, intermediate and advanced genetic operations that estimate physical properties and mimic real genetic processes, including but not limited to transcription (DNA->RNA) and translation (RNA-> codons)
- Support primitive and complex data structures that can handle simple base sequences to full blown genetic maps
- Provide means to allow scientists to add physical properties (e.g. bond strength, annealing temperature) to existing data structures (nucleotide, codon, amino acid, etc.)
- Allow users to input and output files commonly used to store genetic data (e.g. FASTA) and convert data from such files into mutable native data structures
- Allow users to build higher level algorithms (mentioned below) to model molecular biology and calculate optimal sub-sequences in DNA to perform operations such as designing primers for polymerase chain reaction (PCR), a basic genetic tool

**Language design/description**

This programming language is inspired by Biopython, a Python library providing data structures and methods for dealing directly with DNA processes. However, our language is a more general, stripped down version of Biopython that does offer native methods for file parsing, but allows for such features to be built into the language. The basic data unit of our language is the nucleotide, and the language provides data structures for higher levels of genetic modelling

(e.g. DNA sequence, amino acid) composed of the fundamental nucleotide unit. Our language also provides methods for basic file I/O, and a method to interface with several types of files wherein genetic data is often stored.

**Primitive Operations and Higher Level Algorithms**

Below are some basic operations we envision our language supporting and some useful higher level algorithm examples that can be designed using our language.

Low level (Natively supported)

- Converting between data types, such as DNA to RNA, RNA to DNA
- Equality and comparison
- Finding complementary sequences (e.g. (ATTG -> TAAC))
- Slicing a DNA/RNA/Peptide sequence
- Concatenation and appending sequences
- Transcription
- Associating physical properties with data (e.g. melting point of each sequence)
- File Input/Output

Several high level algorithms can be built using our language and are useful in molecular biology, for example:

- **Primer prediction:** One of the most common molecular biology processes is polymerase chain reaction (PCR) which is used to amplify DNA molecules. In order to do this, a process called primer design is required, which selects a sub-sequence of DNA to "start" the amplification reaction. Selecting this subsequence can be laborious, as it requires calculating factors such as annealing temperature (a physical property depending on the nucleotide sequence), and the GC content (the % of G and C nucleotides in the sub-sequence). Since this information is supported by DNA# data structures and DNA# allows for easy processing of DNA sequences, an algorithm can be written in DNA# to find the optimal primer subsequence.
- **Sequence Alignment Calculator:** A common algorithm studied in the field of bioinformatics is the calculation of alignment between two DNA sequences. This is not a straightforward task as random insertion or deletion of nucleotides can throw off comparison algorithms that compare the sequences base by base. Better sequence alignment algorithms can be written in DNA# using its native data structures.
- **Mutation Simulator**: Random mutations are an inherent part of any DNA operation and can take the form of insertion, deletion or substitution of a nucleotide. These mutations must be accounted for when performing any molecular biology processes. Thus, a library could be built using our language and the supported nucleotide codes that represent variability to encode for mutations, as well as determine whether these mutations are silent (result in the same codon) or have an effect on the final peptide sequence.

**Data types/Data structures**

DNAs will be represented in an array of nucleotides depending on what operations will be required to maximize efficiency. Although DNA contains 2 strands, it is sufficient to store the information in 1 array as the second array is complementary to the first. RNA can be represented in a single array of nucleotides. The data structures will accept particularly the following base inputs: A,T,C,G,U. Nucleotides A,C,T,G are used for DNA while A,U,G,C constitute RNA. Additionally, we will support nucleotides K, M, R, Y, S, W, B, V, H, D, X, N which are actually nucleotide codes used in the FASTA format which stand for variable bases (for example, M stands for a variable nucleotide that can be either A or C due to mutation). Like nucleotides, nucleotides also have their own complements. The list of all nucleotides, nucleotide codes and their complements is provided in Figure 1.

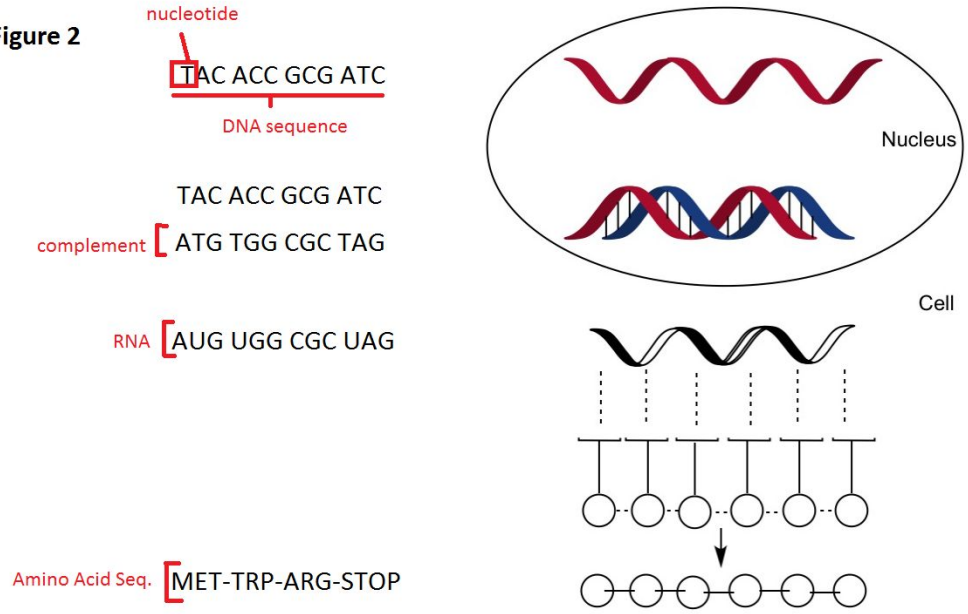
Codons will contain 3 RNA bases and will be used to translate RNA into peptides. Hence, peptides will be represented as linked lists of amino acids. This way, we can use our data structures to represent different stages of the DNA to protein process The data structures will check for the validity of input. The data structures corresponding to nucleotides, DNA, etc. will also carry data about their physical properties, such as bond strength, to relate the data types to the physical DNA molecule, allowing for computational models to be built using our language. A complete representation of the DNA to peptide process using the data structures of DNA# is shown in Figure 2.

**Figure 1**

| Code | Meaning | Etymology | Complement | Opposite |
|---|---|---|---|---|
| A | A | **A**denosine | T | B |
| T/U | T | **T**hymidine/**U**ridine | A | V |
| G | G | **G**uanine | C | H |
| C | C | **C**ytidine | G | D |
| K | G or T | **K**eto | M | M |
| M | A or C | A**m**ino | K | K |
| R | A or G | **P**urine | Y | Y |
| Y | C or T | **P**yrimidine | R | R |
| S | C or G | **S**trong | **S** | **W** |
| W | A or T | **W**eak | **W** | **S** |
| B | C or G or T | not A (**B** comes after A) | V | A |
| V | A or C or G | not T/U (**V** comes after U) | B | T/U |
| H | A or C or T | not G (**H** comes after G) | D | G |
| D | A or G or T | not C (**D** comes after C) | H | C |
| X/N | G or A or T or C | a**n**y | **N** | . |
| . | not G or A or T or C | | . | **N** |
| - | gap of indeterminate length | | | |

source: http://www.boekhoff.info/?pid=data&dat=fasta-codes

**Figure 2**




- **Standard types**

| Type | Definition | Values |
|---|---|---|
| bool | Boolean | true, false |
| int | Integer | integers |
| double | Double Floating Point | real numbers |
| void | Valueless | no values |

- **Primitive**

| Type | Definition | Values |
|---|---|---|
| Bases | Individual Nucleotides | A, T, G, C, U |
| Variable Bases | Variable Nucleotides | K, M, R, Y, S, W, B, V, H, D, X, N |

- **Complex**

| Type | Definition/Value | Sample Values |
|---|---|---|
| DNA | A Sequence of A,T, G, C Deoxynucleotides | AGTXWRCC |
| RNA | A Sequence of A, U, G, C Nucleotides | AGUCC |
| Codon | A three-nucleotide RNA sequence specifying a single amino acid | UGU, CGA, ACC, e.t.c |
| Amino Acid | Basic chemical structures composing a protein | Ala, Trp, Cys, e.t.c |

**Syntax**

- **Lexical Conventions**
  Similar to C++, identifiers in DNA# can be any string of letters, digits, and underscores, however, not beginning with a digit.

  Below is a list of reserved words:

| | | | |
|---|---|---|---|
| true | false | if | else |
| elseif | for | while | continue |
| break | include | end | local |
| then | return | | |
| | | | |

- **Expressions**
  Expression list

  Arithmetic

| | | | |
|---|---|---|---|
| + | - | * | / |
| % | ^ | | |

  Relational

| | | | |
|---|---|---|---|
| < | <= | == | >= |
| != | | | |

Logical

| | | | |
|---|---|---|---|
| or | \| | not | ! |
| and | & | | |

String Operation

| | | | |
|---|---|---|---|
| concatenate | complement | transcribe | translate |
| | | | |

- **Expression Precedence**

^
not
* / %
+ -
..
< <= == >= > ==
and
or

- **Data and Data Structures**
  As a strongly-typed language, all expressions have to be defined and initialized before use.

- **Statements**
  - **Basic Traits**
    Each line is considered a single statement, no need for ';' as in C or C++
    No parentheses in control structure such as 'if' ,'for', etc. Language would look more natural and concise.

  - **Assignment**
    a=a+1
    b="DNA.."Sequence"

  Allow multiple assignment:
    a,b=b,a –swap 'a' and 'b'

  - **Control Structure**

**if statement**

```
if cond==true then
  …
end
```

when there is more than one condition to be decided, we have elseif:

```
if cond1==true then
  …
elseif cond2==true then
  …
else
  …
end
```

**for statement**

```
for i=start_num,end_num,step_length do
  …
end
```

**while statement**

```
while cond==true do
  …
end
```

- **Functions**
  Same as functions in C or C++, all functions have explicit return type and input table, which may look like

```
return_type function_name(input_a_type A,input_b_type B)
        …
end
```

Local function

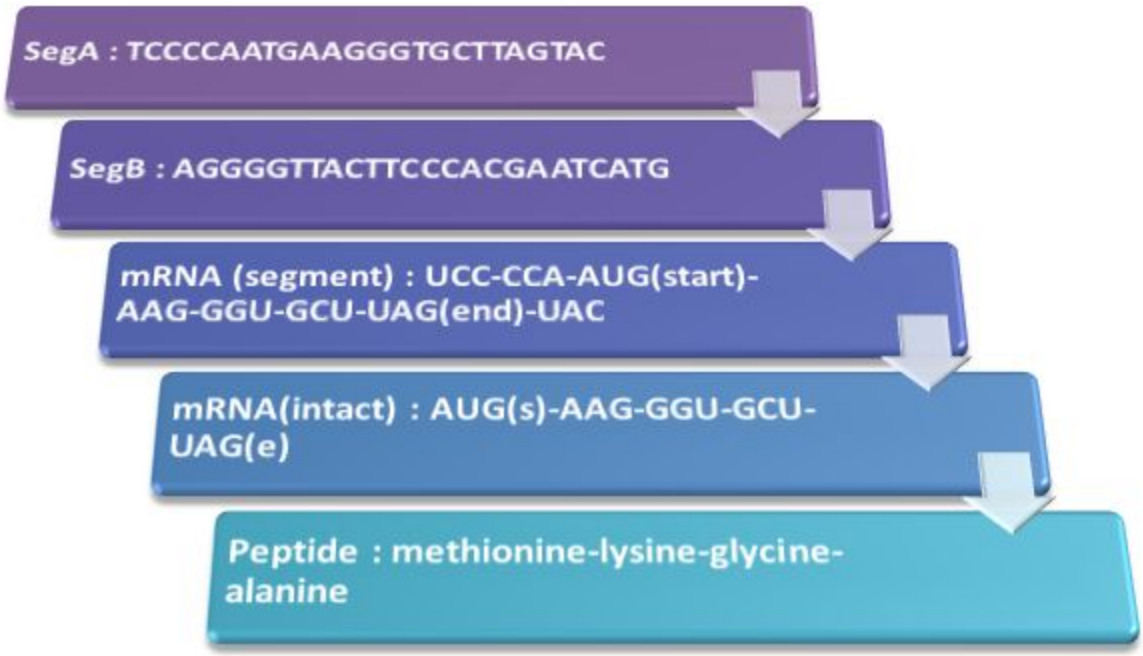
```
local return_type function_name(input_a_type A,input_b_type B)
        …
end
```

- **File Modules**
  Use ‘include’ command to include other files,eg**:**

```
Include 'Data.lang'
```

**Sample programs**

- **Program 1:**
  Think of the following scenario: A biologist gets a random DNA segment from his DNA sequencing machine and he is really interested in knowing whether this random segment contains a sequence which could be transcribed into an intact mRNA (in this scenario an intact mRNA means an RNA sequence containing start codon and end codon). Furthermore, it is important to know what kind of peptide this mRNA will produce.
  As a language specifically designed for molecular biology computations, DNA# is implemented to handle operations as shown in the chart below. The chart shows the whole genetic transcription to peptides. SegA is the random DNA segment (in the real world the segment could be very long and complicated, this is just a simple example of it).



**File BioExpData.dat**

```
<sample1>
TCCCCAATGAAGGGTGCTTAGTAC
<\sample1>
```

**File DNA2Protein.dnas**

```
include “io.dnas”
include “basicBio.dnas”

DNA SampleA;
DNA SampleB';

SampleA = import_dna("BioExpData.dat",sample1)
(*SampleA = TCCCCAATGAAGGGTGCTTAGTAC*)

SampleB = complement(sampleA)
(*SampleB = AGGGGTTACTTCCCACGAATCATG*)

list<codon> mRNA
transcribe(SampleA, mRNA)
 (*mRNA = UCC-CCA-AUG(s)-AAG-GGU-GCU-UAG(e)-UAC*)

i = findStartCodon(mRNA)
j = findEndCodon(mRNA)
mRNA=rnaCutOut(mRNA,i,j)
(* i=3
 * j=7
 * mRna=AUG(s)-AAG-GGU-GCU-UAG(e)  *)

list<amino_acid> PeptideA
translate(list_mRNA, PeptideA)

print("Below is the result:\n")
print_list(PeptideA)
```

**Running Result**

Below is the result:

```
(origin)
->methionine
->lysine
->glycine
->alanine
->(terminal)
```

- **Program 2:**
  Now we perform the following scenario. A geneticist downloads a DNA sequence with some variable regions (such as N, B, etc.) and desires to know what all the possible peptide sequences could result from the sequence

  DNA: TAC-AAK-GGN-CTB-CAT-ATT
  RNA: AUG(s)-UUM-CCN-GAV-GUA-UAA(t)

**File DNA2Protein.dnas**

```
include “io.dnas”
include “basicBio.dnas”

DNA DnaSeg
DnaSeg=DNA(“TAC-AAK-GGN-CTB-CAT-ATT”)

list<codon> mRNA
transcribe(DnaSeg, mRNA)
 (*mRNA =AUG(s)-UUM-CCN-GAV-GUA-UAA(t)*)

list<list<amino_acid>> AllPossiblePeptide
translateFasta(list_mRNA, PeptideA)

print("All possiblities:\n")


from AllPossiblePeptide.begin to AllPossiblePeptide.end
      print(AllPossiblePeptide.current)
end
```

**Running Result**

All possibilities:
#1: methionine-phenylalanine-proline-aspartic-valine-(terminal)
#2: methionine-phenylalanine-proline-glutamic-valine-(terminal)
#3: methionine-leucine -proline-aspartic-valine-(terminal)
#4: methionine-leucine -proline-glutamic-valine-(terminal)