Project 2019/20

Compilers

MSc in Computer Science

University of Camerino

Prof. Luca Tesei

Assignment

Use ANTLR4 with Java as target language to implement a translator between different representations of RNA secondary structures, as illustrated in the following.

Description

Biological background

Ribonucleic acid (RNA) is a linear polymer made of four different types of nucleotides linked together by *phosphodiester bonds*, referred to as *strong interactions*, in such a way that an orientation can be established according to the polarity $5'$ to $3'$ of the molecule. Usually, this is the left-to-right order in which the sequence of nucleotides, the so-called *primary structure*, is given. The four types of nucleotides are: Adenine (A), Guanine (G), Cytosine (C) and Uracil (U). Some RNA strands fold back on themselves determining a *secondary structure*. During this process, each nucleotide can interact at most with one other nucleotide establishing a *hydrogen bond*, referred to as *weak interaction* or *basic loop*. Such hydrogen bonds, weaker chemical interactions than the phosphodiester ones, are subject to restrictions: only Watson-Crick base pairs (G-C or C-G and A-U or U-A) or wobble base pairs (G-U or U-G) can be established. In 2-dimensions, the RNA folding process can determine many RNA secondary structures depending on the free energy of the configurations. The process will reach the configuration with minimal free energy, the so called *optimal secondary structure*.

A *pseudoknot-free* secondary structure is defined as one in which the composing basic loops do not interact with each other. If there is at least an interaction the secondary structure is called *pseudoknotted*. Figure 1 shows a pseudoknotted RNA secondary structure in which the first part (A) is composed of basic loops that do not interact, while in part (B) there are interactions, i.e., some loops cross with other loops. To better visualise these interactions creating pseudoknots it is common to represent the structure as a particular kind of graph called *diagram*. In a diagram graph, the primary structure is depicted as a horizontally connected sequence of vertices and arcs that connect two non-consecutive vertices correspond to weak interactions. Figure 2 shows the diagram representation of the RNA secondary structure in Figure 1. It is immediate to recognise interactions among loops as crossing of arcs.

Any pseudoknot-free RNA secondary structure can be represented as an *algebraic expression* that combines basic loops using a *nesting* (|) and a *concatenation* (+) operator. We introduce them in the following. First, observe that, starting from a primary structure, the introduction of a basic loop creates a secondary structure, which is composed of a *head* sequence, followed by a basic loop, followed by another sequence of unpaired nucleotides, which is a *tail*. Figure 3(A) shows an example of this case, where the head is formed only by the nucleotide A and the tail is the sequence $AGUU$. Secondly, we observe that the introduction of a new basic loop between two unpaired nucleotides that are "under" the arc of the existing loop

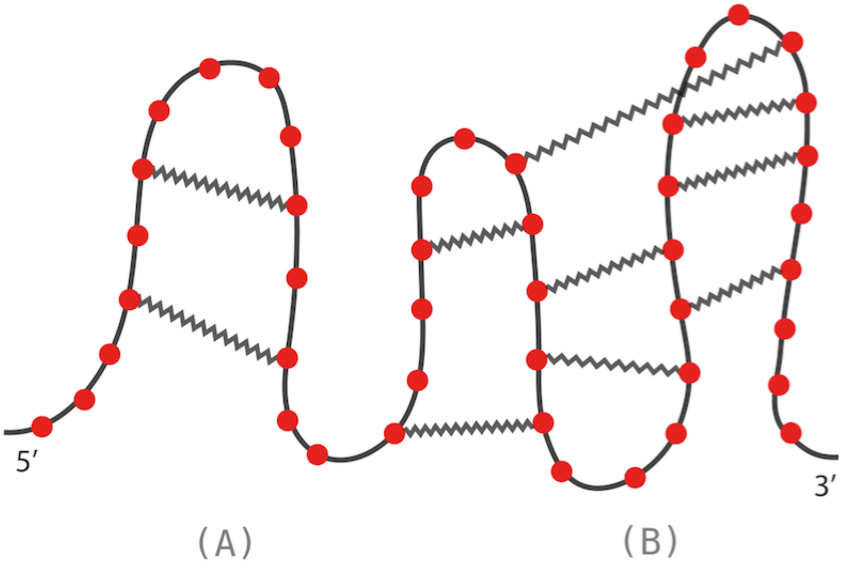

Figure 1: An RNA secondary structure. Each nucleotides is represented by a ball, a strong interaction is depicted by a line and a weak interaction by a zigzag line. Part (A) is pseudoknot-free, while part (B) presents a pseudoknotted motif, which makes the whole structure pseudoknotted.

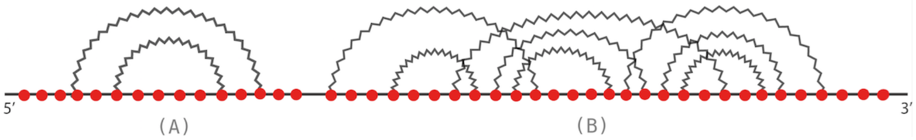

Figure 2: The secondary structure of Figure 1 represented as a diagram graph. Pseudoknots in part (B) are clearly visible as crossings of zigzagged arcs.

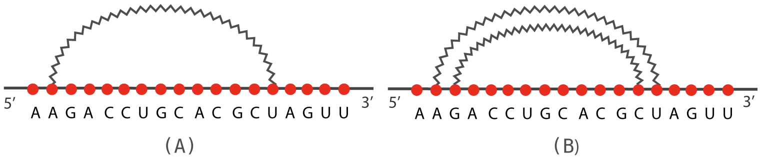
5′
A A G A C C U G C A C G C U A G U U
3′
(A)
5′
A A G A C C U G C A C G C U A G U U
3′
(B)

Figure 3: A basic loop (A) and a nesting of two basic loops (B).

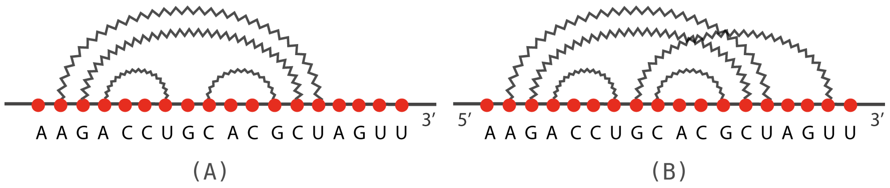
A A G A C C U G C A C G C U A G U U
3′
(A)
5′
A A G A C C U G C A C G C U A G U U
3′
(B)

Figure 4: A concatenation of two basic loops nested into two nested basic loops (A). A pseudoknot is created in (B) by adding a new basic loop that involves an unpaired nucleotide of the structure and another one that belongs to the tail.

in the diagram generates another loop. More specifically, the result is that there are two loops such that one is *nested* into the other, as illustrated in Figure 3(B). *Nesting* is the first of our operators, used to represent these situations. Thirdly, we observe that adding simultaneously two or more new basic loops one after the other without crossing results in a sequence of loops that are *concatenated* with possibly empty sequences of unpaired nucleotides in between. Figure 4(A) shows this case by introducing into the structure of Figure 3(B) two new weak interactions. The two new basic loops that are created are linked by the nucleotide *G*. *Concatenation* is the other operator, introduced to model these situations. Finally, Figure 4(B) shows an example of the introduction of a crossing by making the structure pseudoknotted. We will not permit this situation in our representation language.

Representation with expressions

A pseudoknot-free secondary structure can be represented by a text file specifying the primary structure, the basic loops and their combination with the operators. For instance, the secondary structure of Figure 4(A), in which an additional (G-U) basic loop has been added at the end, can be represented as follows:

```
# example structure for illustrating the language
primary
# the primary structure is indexed starting from 1
AAGACCUGCACGCUAGUU
declare
# these are the basic loops, identified by the starting index and
# the ending index
h1 = (2,14),
h2 = (3,13),
h3 = (4,7),
h4 = (9,12),
h5 = (16,18)
structure
# this the expression using the nesting and concatenation operators
(h3 + h4) | h2 | h1 + h5
end
```

This file is given with the project as `sample1.txt`.

The words `primary`, `declare`, `structure` and `end` are reserved. Lines starting with `#` are comments and must be ignored. The primary sequence is a string of uppercase or lowercase letters belonging to the IUPAC nucleotide codes (see `https://www.bioinformatics.org/sms/iupac.html`). The string must not contain spaces and its elements are indexed (for the purposes of the next sections) starting from 1. The declarations are a sequence of comma-separated definitions of the form **id** $= (\mathbf{num}_1, \mathbf{num}_2)$, where each **id** can be an sequence of letters or digits starting with a letter. The two numbers $\mathbf{num}_1$ and $\mathbf{num}_2$ are, respectively, the starting index and the ending index of a basic loop. They must respect the following constraints:

- $\mathbf{num}_1 + 1 < \mathbf{num}_2$, i.e. a basic loop must contain at least an unpaired nucleotide;
- the primary sequence at indexes $\mathbf{num}_1$ and $\mathbf{num}_2$ must form a valid pair (Watson-Crick or wobble). Note that even if any IUPAC nucleotide code is admitted in the primary sequence, basic loops can be formed only with the nucleotides A, G, C and U;
- any nucleotide can belong to *at most* one basic loop, i.e. a structure in which an index i belongs to two different declarations is not admissible.

In the structure expression the concatenation operator + associates on the left and has less precedence than the nesting operator `|`. The nesting operator associates on the left as well and *admits as right operand only an **id**, not a complex expression*. For instance, the expression `(a + b) | c` is admissible, while the expression `a | (b + c)` is not. The concatenation operator must respect the following constraint:

- an expression `e1 + e2` must be such that the rightmost paired nucleotide of `e1` must have an index strictly less than the index of the leftmost paired nucleotide of `e2`. This ensures that the structures represented by `e1` and `e2` do not cross and are concatenated in a left-to-right way.

The nesting operator must respect the following constraints:

- as mentioned above, the right operand must be an **id**, say $\mathtt{a}$, and the left operand, say $\mathtt{e}$ can be any subexpression with nesting, **id**s and parenthesised expressions;
- given the expression $\mathtt{e}|\mathtt{a}$, let $\mathtt{e}_{\mathtt{left}}$ be the index of the leftmost paired nucleotide of $\mathtt{e}$ and let $\mathtt{e}_{\mathtt{right}}$ be the index of the rightmost paired nucleotide of $\mathtt{e}$. Supposing that $\mathtt{a}$ is defined as $(\mathtt{a}_1, \mathtt{a}_2)$, it must hold that $\mathtt{a}_1 < \mathtt{e}_{\mathtt{left}} < \mathtt{e}_{\mathtt{right}} < \mathtt{a}_2$, i.e. the structure $\mathtt{e}$ must be completely “under” the basic loop $\mathtt{a}$.

Translation

The parser should parse text files like the sample one above and check all the constraints that have been specified. If all the constraint are met then the so-called Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation) must be used to generate as output the corresponding representation of the secondary structure. The Dot-Bracket notation denotes paired nucleotides by matching pairs of parenthesis () and unpaired nucleotides by dots. As an example, the output text file that should be produced as a translation for the structure of the sample input above is the following:

```
# This is the dot-bracket notation representation of the structure above
# First the primary structure, as in the other file
AAGACCUGCACGCUAGUU
.(((..).(..))).(.)
# After the corresponding dot-bracket notation
```

This file is given with the project as `sample1.translation.txt`.

Also in case of dot-bracket files, lines starting with an `#` are comments (translation is not required to generate any comment). For longer sequences, both the primary structure and the dot-bracket string should be divided into sequences of 50 elements in subsequent lines. For instance, the following is a dot-bracket notation file for a real RNA molecule downloaded from the RNA STRAND Database (see `http://www.rnasoft.ca/strand/`):

```
# File PDB_00943.dp
# RNA SSTRAND database
# External source: RCSB Protein Data Bank 2B3J, number of molecules: 4
# The secondary structure annotation was obtained with RNAview

UUGACUaCGGAUCAAUUGACUaCGGAUCAAGACUaCGGUUUGACUaCGGA
UCAA
((((( .....)))))(((((.....))))).........(((((.....)
))))
```

This file is given with the project as `PDB_00943.dp.dbn.txt`.

As another test case, the corresponding representation using the algebraic expressions is the following:

```
primary
UUGACUaCGGAUCAAUUGACUaCGGAUCAAGACUaCGGUUUGACUaCGGAUCAA
declare
h1 = (5,11),
h2 = (4,12),
h3 = (3,13),
h4 = (2,14),
h5 = (1,15),
h6 = (20,26),
h7 = (19,27),
h8 = (18,28),
h9 = (17,29),
h10 = (16,30),
h11 = (44,50),
h12 = (43,51),
h13 = (42,52),
h14 = (41,53),
h15 = (40,54)
structure
h1 | h2 | h3 | h4 | h5 + h6 | h7 | h8 | h9 | h10 + h11 | h12 | h13 | h14 | h15
end
```

This file is given with the project as `PDB_00943.dp.expr.txt`.

Other test files checking that errors are reported are given with the project:

- `errorBasicLoopWithoutUnpairedNucleotide.txt`
- `errorMoreThanOnePairing.txt`
- `errorPairing.txt`
- `errorConcatenation.txt`
- `errorNesting.txt`

The given implementation should report the right errors in all the cases.

Submission

Prepare a written report describing your grammar, your code and some test results. You can use screenshots to show some of the results.

The class file with the `main` method may get the input as standard input and produce the output on the standard output. In this case operating system redirecting features can be used to pass a text input file to the translator and to redirect the output on a text file. As an alternative, the `main` method can get the input and/or output files as command line parameters with the proper command line options.

Send by email to the teacher all the files of the implementation (as a jar) plus the report in pdf. The sending must occur before the starting of a written test (fixed for each exam session): a student that has not sent the project before the written test can not participate to the written test. The exam is passed when both the following conditions are satisfied:

1. the project has been sent and has been approved;
2. the written test has been passed;

In case a student does not pass the written test, he/she does not have to resend the (approved) project before the next attempt(s) to the written test. The final grade is a combination of the grades of the two tasks.