Journal of Universal Computer Science, vol. 12, no. 11 (2006), 1500-1520
submitted: 1/5/06, accepted: 15/10/06, appeared: 28/11/06 © J.UCS

Constraint Based Methods for Biological Sequence Analysis

Maryam Bavarian
(Computing Science Department
Simon Fraser University, USA
mbavaria@sfu.ca)

Veronica Dahl
(Computing Science Department
Simon Fraser University, USA
veronica@cs.sfu.ca)

Abstract: The need for processing biological information is rapidly growing, owing to the masses of new information in digital form being produced at this time. Old methodologies for processing it can no longer keep up with this rate of growth. The methods of Artificial Intelligence (AI) in general and of language processing in particular can offer much towards solving this problem. However, interdisciplinary research between language processing and molecular biology is not yet widespread, partly because of the effort needed for each specialist to understand the other one's jargon. We argue that by looking at the problems of molecular biology from a language processing perspective, and using constraint based logic methodologies we can shorten the gap and make interdisciplinary collaborations more effective. We shall discuss several sequence analysis problems in terms of constraint based formalisms such Concept Formation Rules, Constraint Handling Rules (CHR) and their grammatical counterpart, CHRG. We postulate that genetic structure analysis can also benefit from these methods, for instance to reconstruct from a given RNA secondary structure, a nucleotide sequence that folds into it. Our proposed methodologies lend direct executability to high level descriptions of the problems at hand and thus contribute to rapid while efficient prototyping.

Key Words: protein structure, RNA secondary structure, gene prediction, concept formation, constraint handling rules, constraint handling rule grammars
Category: I.2.1, D.3.2, D.3.3

1 Introduction

The application to molecular biology of Artificial Intelligence (AI) methods such as logic programming and constraint reasoning constitutes a fascinating interdisciplinary field which, despite being relatively new, has already proved quite fertile. For instance, our book Logic Grammars [1] was widely used to help discover the human genome [23]. Our work on plant pathogen identification for Agriculture and Agri-Food Canada [30] yielded spectacular results: whereas with previous tools, the processing time increases exponentially with sequence length or number of sequences, we provided a novel algorithm for which processing time increases linearly with the amount of data to be analysed. Our methods can moreover be viewed as modules to be embedded within higher level while still efficiently executable descriptions of other interesting molecular biology problems.

Over the past decade there has been a dramatic increase of collection rates for biological data, making the need for resorting to AI methods even more acute. Simultaneously, the intersection between logic programming and constraint reasoning has been maturing into extremely interesting methodologies, most notably Constraint Handling Rules, or CHR [18]. Dahl has applied these methodologies first to human language processing, through implementing Property Grammars [7, 9] (a linguistic formalism based on constraints between sentence constituents rather than on the traditional notion of phrase structure) in CHR, and through a parsing system for balanced parenthesis [8] and then to cognitive sciences, through generalizing these results into a general cognitive theory of concept formation [17] with applications to cancer diagnosis [5, 4], to medical report interpretation [28] and to concept extraction [14]. We are presently applying CHR methodologies to the problem of genetic structure analysis, in particular, we have obtained excellent results for reconstructing a nucleotide sequence which can fold into a given RNA secondary structure: whereas previous approaches, while working well for short sequences, are computationally hard for longer ones (more than 500 nucleotides), our methodology by using heuristics obtains approximate but useful results in O(n) time for such long sequences.

In this tutorial paper we share the expertise obtained in the course of this work in a pedagogical fashion. We first provide a short intuitive introduction to the main concepts involved in biological sequence analysis. Next we survey the CHR formalism itself, as well as its grammatical counterpart, CHRG [12], and our own CF (Concept Formation) formalism, exemplifying each with simplified subproblems within those we have addressed in the literature above referenced. We then discuss three major sequence processing problems (gene prediction, RNA secondary structure and protein structure) and compare the methods we advocate in this article with previously used methods.

2 Biological sequence analysis: the problem

We are interested in the general problem of protein generation from DNA. DNA (Deoxyribonucleic Acid) is a nucleic acid that contains the genetic instructions specifying the biological development of all cellular forms of life (and many viruses) working like an information archive in each organism. DNA is often referred to as the molecule of heredity, as it is responsible for the genetic propagation of most inherited traits and it is replicated and transmitted to the offspring during reproduction. DNA can be looked at as a template for building other DNAs and proteins. DNA is encoded with four building blocks, called *nucleotides* or *bases*, which are: A (Adenine), C (Cytosine), G (Guanine) and T

(Thymine). DNA can be read and replicated and it is believed that much of the role that DNA plays in forming proteins depends on this. These characteristics make the linguistics treat a gene as a 'word' while a genome, the total genetic material of species as 'text' in DNA code.

The central dogma of molecular biology considers how a sequence of DNA bases turns into a sequence of amino acids which in turn, forms proteins. The procedure is as follows: the information from a gene recipe is copied from a strand of DNA to a strand of messenger RNA (mRNA). The mRNA molecules then travel out of the nucleus and ribosome molecules read the genetic information inside mRNA and translate them into amino acids based on the genetic coding scheme . This strand of amino acids is then folded to make the three dimensional structure of the protein.

$$DNA \Longrightarrow_{RNApolymerase} pre - mRNA \Longrightarrow_{Splicesome} RNA \Longrightarrow_{Ribosome} Protein$$

There are some important facts regarding these transformations. First of all, in the first transformation, genes are the material that would be processed by RNA polymerase machinery. The second fact is that splicesome can generate different RNA resulting in different proteins. And finally, some generated proteins of this process might work as a barrier in forming other proteins and on the other hand some of them might accelerate the production of others [13].

The major processes in protein production are:

1. **Transcription:** DNA sequences are transcribed by a biological machine called RNA polymerase into sequences called Pre-mRNA, in which the nucleotide T has changed to U(Uracil).

2. **Splicing:** Pre-mRNA is transformed into mRNA by another biological machine called splicesome.

3. **Translation:** another biological machine, called ribosome, reads the triplets of consecutive nucleotides (called *codons*) in mRNA and generates amino acids in parallel.

4. **Folding:** amino acids twist and fold to form proteins.

Computational methods have been used in each step of the transformation to simulate the work of these machines. Here we only focus on the language of DNA which consists of four words: A,C,T,G. As we mentioned, nucleotide T would be transformed into nucleotide U in transcription step. The words of this language group together to code for different amino acids. For instance, the sequence *UUU* corresponds to the amino acid known as phenylalanine. A first programming problem could be to implement in Prolog the translation of codons into aminoacids- an easy enough task for any of us, and while challenging for biologists, a good introduction to interdisciplinary work. The core predicate contains table-like information such as:

```
translate([u,u,g],tryptophan).
translate([u,u,a],leucine).
translate([u,u,c],leucine).
```

and we can conceive a logic grammar version as well, containing such grammar rules as:

```
s([u,u,g]) --> [tryptophan].
s([u,u,a]) --> [leucine].
s([u,u,c]) --> [leucine].
```

As we can observe from this example, we are dealing with an ambiguous language: different codons can code for a given amino acid.

Further Logic Programming (LP) base methodologies will be examined, which can accommodate more sophisticated problems of biological sequences. In the next section, we introduce them through simple examples for didactical purposes, and later we shall expand on their use for three substantial biological sequence analysis problems: gene prediction, RNA secondary structure and protein secondary structure.

3 Overview of the proposed methodologies

3.1 Assumption Grammars

3.1.1 The methodology

In human communication as well as in AI, assumptions play a central role. Linguists and logicians have uncovered their many facets. For instance, the assumption of a looking glass' existence and unicity in "The looking glass is turning into a mist now" is called a *presupposition*; the assumption that a polite request, rather than a literal question, will be "heard" in "Can you open the door?" is an *implicature* [15].

Assumptions [15, 16] are basically backtrackable assertions which can serve among other things to keep somewhat globally accessible information (an assertion can be used, or consumed, at any point during the continuation of the computation).

Assumption Grammars are logic grammars augmented with a) linear and intuitionistic implications scoped over the current continuation (i.e., the remainder of the computation from the current point in time) and b) hidden multiple accumulators useful in particular to make the input and output strings invisible. Intuitionistic assumptions */1 adds temporarily a clause usable in later proofs.

Such a clause can be used an indefinite number of times like asserted clauses except that it vanishes on backtracking. Linear assumptions +/1 adds temporarily a clause usable at most once in later proofs. For consuming these assumptions, one can use -/1.

In the next section we introduce Assumption Grammars through a natural language processing example (anaphora), and next we shall show its uses for processing biological sequences.

3.1.2 An example

We shall now illustrate how assumption grammars can deal with intersentential dependencies through the example of anaphora, in which a given noun phrase in a discourse is referred to in another sentence, e.g. through a pronoun. We refer to the noun phrase and the pronoun in question as entities which co-specify, since they both refer to the same individual of the universe [15].

As a discourse is processed, the information gleaned from the grammar and the noun phrases as they appear can be temporarily added as hypotheses ranging over the current continuation. Consulting it then reduces to calling the predicate in which this information is stored.

We exemplify the hypothesizing part through the following noun phrase rules:

```
np(X,VP,VP) --> proper_name(X),  *specifier(X).
np(X,VP,R) --> det(X,NP,VP,R), noun(X,F,NP), *specifier(X,F).

pronoun(X,[masc,sing]) --> [he].
pronoun(X,[fem,sing]) --> [her].

anaphora(X) --> pronoun(X).

noun(X,[fem,sing],woman(X)) --> woman.
```

The intuitionistic assumption, `*specifier(X)`, keeps in `X` the noun phrase's relevant information. In the case of a proper name, this is simply the constant representing it plus the agreement features gender and number; in the case of a quantified noun phrase, this is the variable introduced by the quantification, also accompanied by these agreement features.

Potential co-specifiers of an anaphora can then consume the most likely co-specifiers hypothesized (i.e., those agreeing in gender and number), through a third rule for noun phrase:

```
np(X,VP,VP) --> anaphora(X), -specifier(X).
```

Semantic agreement can be similarly enforced through the well-known technique of matching syntactic representations of semantic types.

3.1.3 Applications

Assumptions can be applied to find some patterns in biological sequences such as *tandem repeats*. A tandem repeat is a nucleotide sequence that results from a class of mutation event called tandem duplication which converts a stretch of DNA code (called the "pattern") into two or more copies, each following the preceding one in contiguous fashion [25]. One of the reasons for identifying tandem repeats is that according to biology and genetics sciences, tandem repeats occur frequently in genomic sequences and they can have a potential role in gene regulation, including development of immune system cells. Many genetic diseases such as Huntington's disease and Friedreich's ataxia (FRDA) are also shown to be associated with uncontrolled expansions of tandem repeat patterns.

Searls in [25] has introduced some grammar rules for finding tandem repeats. These grammar rules can be represented in assumption grammar as follows:

```
tandem_repeat --> [X],{push(X)},tandem_repeat.
tandem_repeat --> repeat.
repeat --> {-stack([])},[].
repeat --> {pop(X)}, repeat, [X].
push(X):- -stack(Y), +stack([X|Y]).
pop(X):- -stack([X|Y]), +stack(Y).
```

Here we make use of a global variable as a stack, add on to it through assumption (noted '+') , and remove elements from it through consumption (noted '-'). Assumptions are available in some modern logic programming environments such as BinProlog and CHRGs.

Another example which is very similar to tandem repeat is *inverted repeat*. Inverted repeats are also common features of nucleic acids, which in the case of DNA result whenever a substring on one strand is also found nearby on the opposite strand [25]:

```
inverted_repeat --> [X],{push(X)}, inverted_repeat.
inverted_repeat --> repeat.
repeat --> {pop(X)},~[X],repeat.
repeat --> {-stack([])},[].
push(X):- -stack(Y),+stack([X|Y]).
pop(X):- -stack([X|Y]),+stack(Y).
```

3.2 Constraint Handling Rule, Constraint Handling Rule Grammars

3.2.1 The methology

Constraint handling rules (CHR) is a concurrent committed-choice constraint logic programming language which has proved useful for algorithms dealing with constraints [19]. By presenting a highly executable framework, CHR has tried to form a bridge between theory and practice in logic programming. It also provides programmers efficiently executable specifications by supporting rapid prototyping. To this day, CHR has been applied in several applications including theorem proving with constraints, combining forward and backward chaining, combining deduction and abduction, bottom-up evaluation with integrity constraint, etc [19].

CHR works on constraint stores with its rules interpreted as rewrite rules over such stores. A string to be analyzed such as *"leucine tryptophan phenylalanine"* is entered as a sequence of constraints

```
token(0,1,leucine),token(1,2,tryptophan),token(2,3,phenylalanine)
```

that comprise an initial constraint store. The integer arguments represent word boundaries, and a grammar for this intended language can be expressed in CHR as follows.

```
token(X0,X1,tryptophan) ==> codon(X0,X1,[u,u,g]).
token(X0,X1,leucine) ==> codon(X0,X1,[u,u,a]).
token(X0,X1,leucine) ==> codon(X0,X1,[u,u,c]).
token(X0,X1,phenylalanine) ==> codon(X0,X1,[u,u,u]).
```

We say that ambiguity is inherently treated because all possibilities will be expressed in the constraint store resulting from an ambiguous input. In the above example, for instance, both a codon [u,u,a] and a codon [u,u,c] will be found between points 0 and 1.

CHR Grammars, or CHRGs for short, are based on Constraint Handling Rules [19] and were introduced in [11, 10] as a bottom-up counterpart to definite clause grammars (DCGs) defined on top of CHR in exactly the same ways as DCGs take their semantics from and are implemented by a direct translation into Prolog. CHRGs are executed as CHR programs that provide robust parsing with an inherent treatment of ambiguity.

The input and output arguments of the above translation example can be made implicit if using CHRG, which uses `::>` for the rewrite symbol. The first rule, for instance, would become:

```
token(tryptophan)::> codon([u,u,g])
```

We use the version of CHR that is an extension of Sicstus Prolog[1], and notation with capital letters for variables, etc., is as in Prolog. Here we restrict ourselves to the subset of CHR consisting of propagation rules only, for which it is easy to specify declarative and procedural semantics.

A CHR *program* is a finite set of *rules* of the form

[1] Sicstus Prolog Manual: http://www.sics.se/isl/sicstuswww/site/documentation.html

```
Head ==> Guard|body
```

where `Head` and `Body` are conjunctions of atoms and `Guard` is a test constructed from built-in predicates; the variables in `Guard` and `Body` occur also in `Head`; in case the `Guard` is the local constant "true", it is omitted together with the vertical bar. Its logical meaning is the formula $\forall(\mathit{Guard} \rightarrow (\mathit{Head} \rightarrow \mathit{Body}))$ and the meaning of a program is given by conjunction.

A *derivation* starting from an initial state called a *query* of ground constraints is defined by applying rules as long as it adds new constraints to the store. A rule as above *applies* if it has an instance H`==>`G`|`B with G satisfied and H in current store, and it does so by adding B to the store.

It is to be noted that if the application of a rule adds a constraint c to the store which already is there, no additional rules are triggered, e.g., p `==>` p does not loop as it is not applied in a state including p.

There are three types of CHR rules:

- **Propagation rules** which add new constraints (body) to the constraint set while maintaining the constraints inside the constraint store for the reason of further simplification.
- **Simplification rules** which also add as new constraints those in the body, but remove as well the ones in the head of the rule.
- **Simpagation rules** which combine propagation and simplification traits, and allow us to select which of the constraints mentioned in the head of the rule should remain and which should be removed from the constraint set.

The factors strengthening CHR include the combination of propagation and multi-set transformation of logical formulae in a concurrent, guarded rule-based language [19]. The rewrite symbols for the first two rules are respectively: `==>`, `<=>` and for simpagation rules, the notation is `Head1\Head2<=>body`. Anything in `Head1` remains in the constraint set and anything in `Head2` is removed from the constraint set and `body` is added to the constraint store.

The CHRG notation makes the word boundary arguments implicit and, analogously to DCGs, includes a syntax for using non-grammatical constraints. The corresponding rewrite symbols for CHRG are `<:>` for simplification and simpagation rules and `::>` for propagation rules.

3.2.2 Applications

In Section 3.1, we introduced tandem repeats and their importance and showed an implementation of them using Assumption Grammar. Using the same technique we now present the same program in CHRG:

```
![X],tandem([X|Y]) <:> tandem(Y).
repeat(L) ::> tandem(L).
[X] ::> repeat([X]).
repeat(Y),![X] <:> repeat([X|Y]).
tandem(P1,P2,[]) ==> tandem_repeat(P1,P2).
```

In this implementation instead of using stacks, we make use of a list argument inside the constraints `tandem/1` and `repeat/1`. A tandem repeat is formed when we have a constraint `tandem/1` for which the list argument is empty. This implementation is also capable of identifying tandem repeats inside the input string. The `!` sign is used before the predicate inside simpagation rules that are needed to stay inside the constraint store.

Using string grammars [25], it is possible to solve this problem differently. The solution given below, finds every possible string inside the input string and by identifying similar strings discovers the tandem repeats.

```
[X], string(Y) ::> string([X|Y]).
[X] ::> string([X]).
string(X),string(X) ::> tandem_repeat(X).
```

This implementation can also be extended to identify any number of tandem repeats:

```
[X], string(Y) ::> string([X|Y]).
[X] ::> string([X]).
tandem_repeat(X,C),string(X) <:> C1 is C+1 | tandem_repeat(X,C1).
string(X),string(X) ::> tandem_repeat(X,2).
```

The inverted repeat problem can also be solved in CHRG and CHR using the same techniques. Comparing the three suggested implementations (one with assumption grammar and two using CHRG), we can suggest that depending on the goal of the user any of them can be utilized. The first implementation is very straightforward and is applicable whenever we are only interested in understanding whether or not an input string is a tandem repeat. The second implementation can also give us the opportunity to identify tandem repeats which are hidden inside an input string. The third implementation does not seem as efficient as the other two as it tries to find all the possible strings inside the input string, however it is able to recognize more than two consecutive tandem repeats inside the input string.

3.3 Concept Formation Grammars

3.3.1 The methodology

In [17], we introduced a cognitive model of Concept Formation, which has been used for oral cancer diagnosis [5] and specialized into grammatical concept formation, with applications to property grammar parsing [14].

The grammatical counterpart of Concept Formation is an extension of CHRGs which dynamically handles properties between grammar constituents and their relaxation, as statically defined by the user.

Let's first exemplify within the traditional framework of rewrite rules which implicitly define a parse tree. Whereas in CHRG we would write the following toy grammar[2]:

```
[a] ::> determiner(singular).
[boy] ::> noun(singular).
[boys] ::> noun(plural).
[laughs] ::> verb(singular).

determiner(Number),noun(Number),v(Number) ::> sentence(Number).
```

To parse correct sentences such as "a boy laughs", in CFGs we can also allow incorrect sentences to be generated, but ask the system to point out to us which properties are violated by such incorrect sentences. This requires the following:

- a definition of all properties in terms of a system predicate prop/2.
- a declaration of what properties can be relaxed, done through the system's (Prolog) predicate relax/1, and for properties to be called through the system predicate acceptable/1.

For instance, an agreement property to check that the number of a subject's determiner (NDet) coincides with that of the noun (Nn) and with that of the main verb (Nv), and which can be defined in Prolog as:

```
agreement(Ndet,Nn,Nv):- Ndet=Nn, Nn=Nv.
```

must be expressed in terms of the system predicate prop/2 as follows, with all concerned arguments grouped into a list:

```
prop(agreement,[Ndet,Nn,Nv]):- Ndet=Nn, Nn=Nv.
```

If we now want to relax this property, so that number mismatches are detected but do not block the parse, we can use the system predicate relax/1, as follows:

```
relax(agreement).
```

[2] Terminal symbols are denoted between brackets as in DCGs; the double colon arrow indicates CHR grammar rules, as opposed to CHR proper.

A property is then considered “acceptable” if it either succeeds, or if it fails but has been relaxed:

```
acceptable(prop(P,A)):-call(prop(P,A)); relax(P).
```

Therefore, properties must be tested through the system predicate acceptable/1. For our example, we would write:

```
determiner(Ndet), noun(Nn), v(Nv) ::>
acceptable(prop(agreement,[Ndet,Nn,Nv])) | sentence(Nv).
```

The system will now accept also sentences which do not agree in number, while pointing out the error in the list of violated properties, as a side effect of the parse. In the case of “a boy laughs”, the agreement property will appear in the list of violated properties automatically constructed as a result of the parse.

Degrees of acceptability can also be defined using binary versions of “relax” and “acceptable”, whose second argument evaluates to either true, false, or a degree of acceptability, according to whether (or how much of) the property is satisfied. This allows the user as well to relax specific constraints rather than types of constraints, by specifying right-hand side conditions on these binary counterparts. It would be interesting to allow the user to rank constraints, but this facility is not included in our current model. For full details, see [17].

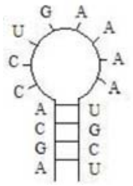

Figure 1: A hairpin loop

3.3.2 Application

It is believed that in biology there are no rules without exceptions [26]. For example, previously scientists believed that a gene can only be responsible for one protein, however the recent works in this area has proved this to be wrong. A gene may generate several proteins under different circumstances. Concept formation features gives us the ability to write rules for biological transformations that are closer to what happens in real life. As an example we show how to use concept formation rules in finding *hairpin loops*, one of the common patterns seen in RNA structure. As shown in Figure 1, a hairpin loop consists of a stem which leads to a loop at the end. According to the biochemical laws the loop part should at least contain three nucleotides, but in some rare RNA structures, it might happen that a loop contains only two nucleotides. Concept Formation Rules is ideally suited to model molecular biology situations such as these because it allows the user to state exceptions declaratively and handles them invisibly and efficiently.

The case of loops with only 2 nucleotides, for instance, can be directly implemented by the CF rule:

```
stem(X1,Y1,X2,Y2),loop(X,Y)==> X is X2+1,Y is Y2-1,
acceptable(prop(length,[X,Y])) |  hairpin(X,Y).
```

4 More sophisticated applications of the proposed methodologies

4.1 Gene prediction

The fundamental physical and functional unit of heredity is called a gene. It is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e., a protein or RNA molecule). Genes determine many aspects of anatomy and physiology by controlling the production of proteins. Each individual has a unique sequence of genes, or genetic code. If you take a look at scientific sections of the newspapers, magazines or internet, you will probably encounter such articles as: “Researchers find genes for depression”, “Researchers find gene that causes liver cancer in mice”, “Two new lung cancer genes” etc. Discovery of genes that influence disease risk in human populations is of key importance to the pharmaceutical industry. This is because identifying the gene automatically identifies the protein in which alteration of function causes disease. For the pharmaceutical industry, this protein, together with other proteins that are part of the same physiological pathway, becomes a potential drug target. Once a drug target has been identified, modern techniques allow the pharmaceutical industry to rapidly screen large numbers of chemical compounds for action on this target. It is expected that drugs acting directly on the proteins, in which genetically determined alteration of function causes disease, will be highly effective in preventing or treating these diseases. This is one of the reasons why researchers in various branches of science are trying to identify genes. The problem of identifying genes responsible for certain diseases or characteristics has many difficulties. One of them is the fact that there are often several genes that are responsible for certain diseases. By studying the DNA of the individuals and their families having rare diseases one may be more successful in pinning down the main responsible genes[13].

The order in which the bases of DNA are linked in a gene is called the sequence of a gene. There exist two types of genes: RNA genes and protein coding which is our focus here. Stretches of DNA that code for proteins are called *exons*. In eukaryotes, cells with membrane-bound nucleus, exons in a given gene are generally separated from each other by stretches of DNA that do not contain instructions for constructing proteins, they are called *introns*.

There are various approaches for the problem of gene finding but here our focus is on the computational methods. Computational gene finding is the process of identifying potential coding regions in an uncharacterized region of the genome. This area is still a subject of active research. There actually exist many different gene-finding software packages and no program is capable of finding everything. Common techniques include homology, combinatorial dynamic programming, probabilistic modeling especially Hidden Markov Models and neural nets but here another approach in logic programming is used. Cohen has proposed a set of grammar rules for finding genes [13] and here we have translated the same grammar into CHRG.

This approach is based on the fact that beginning and ending of the genes are marked by some special codons (start and stop). The start codon is the sequence ATG while the stop codon might take any of the three possible forms: TAA, TAG, or TGA. The main part of the gene is placed between the start and stop codon and is composed of exons interrupted by introns, distinguished from each other through a set of relatively complicated rules:

```
start_codon,gene_body, stop_codon ::> gene.
start_codon,stop_codon ::> gene.
[a,t,g] ::> start_codon.
[t,a,a] ::> stop_codon.
[t,a,g] ::> stop_codon.
[t,g,a] ::> stop_codon.
exon_body,left ::> gene_body.
exon_body ::> gene_body.
left ::> gene_body.
[a,g],intron_body,right ::> right.
[a,g],exon_body,left ::> right.
[a,g],left ::> right.
[a,g],right ::> right.
[a,g],exon_body ::> right.
[a,g] ::> right.
[g,t],intron_body,right ::> left.
[g,t],exon_body,left ::> left.
[g,t],exon_body ::> left.
[g,t],left ::> left.
```

```
[g,t],right ::> left.
[g,t] ::> left.
base ::> intron_body.
base ::> exon_body.
base,[a] ::> base.
base,[c] ::> base.
base,[g] ::> base.
base,[t] ::> base.
[a] ::> base.
[c] ::> base.
[g] ::> base.
[t] ::> base.
```

4.2 RNA secondary structure

In biology there is an important relationship between structure and function. The shape of the molecules usually determine whether or not two compounds can interact with each other [29]. Here we examine the structure of RNA and how it can be represented and solved by means of constraint handling rules. RNA (ribonucleic acid) is a chemical found in cells which codes for amino acid sequences. Each RNA molecule is made up of four different compounds called nucleotides or bases, each noted with one of the letters: A, C, G, and *U* (Uracil). During the transformation process of proteins, synthesis of RNA molecules are directed by DNA sequences.

RNA molecules have two sets of structural information: the primary structure and the secondary structure. The linear arrangement of the four compounds (nucleotides A,C,G and U) is known as the *primary structure* of RNA. As we already mentioned the primary structure of RNA and DNA are very similar to each other and the only difference is that instead of nucleotide U in RNA, DNA contains nucleotide T. However, in the actual structure, DNA is formed by a double helix while RNA is typically single stranded. This strand of nucleotides is folded onto itself by pairings of the nucleotide A with U and C with G which are called *Watson-Crick base pairs* or *canonical base pairs.* Pairing also might happen between nucleotide G and nucleotide U but this is not very frequent. The structure made by these pairings is called *RNA secondary structure* . Each base in the secondary structure can be paired with at most one base. The *tertiary structure* of RNA is the actual 3D structure of RNA in nature. The secondary and tertiary structure of RNA determine the RNA interaction with other cell components. A number of structural motifs are found inside RNA secondary structure such as helix, hairpin loop, bulge loop, internal loop, etc (Figure 2).

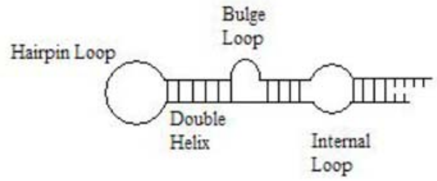

Figure 2: Common motifs in RNA secondary structure

There are many problems in biology that involve RNA, but here we only focus on two of these problems which are strongly related to each other. The first problem is the problem of RNA secondary structure prediction where given the primary structure of an RNA, the secondary structure is to be determined. The second problem is in fact the inverse of the first problem: given a secondary structure, a sequence of nucleotide is asked for that folds onto that structure. Below, we will show how we can use our methods to solve these two interesting and at the same time challenging problems.

4.2.1 RNA secondary structure prediction

The problem of RNA secondary structure prediction consists of determining which secondary structure will be adopted by a given sequence of nucleotides (Figure 3). One of the significant benefits of understanding the secondary structure of RNA is to determine its chemical and biological properties. Although it is not yet possible to reliably predict RNA tertiary structure, there are fairly good RNA secondary structure prediction algorithms available. Clearly, the tertiary structure of RNA is much more useful and gives us a better insight of RNA functions, but while it is still quite difficult to predict, researchers use the secondary structure to explain most of the functionalities of RNA.

Several methods have tackled this problem so far, yet there are two main approaches. The most common method is finding *minimum free energy* [31]. In this method we look for the structure out of all possible folds with the least free energy. There are several factors which determine the free energy of a fold, some of them are: the number of GC pairs in comparison to AU and GU (the reason is that the GC pairs form a more stable structure), number of base pairs in a double helix, the number of bases which are not paired.

The second method of RNA secondary structure prediction deals with sequence comparison. It uses multiple sequence alignment, so for finding the secondary structure of a given RNA we have to first align it with a couple of similar sequences and from the homology, derive the secondary structure.

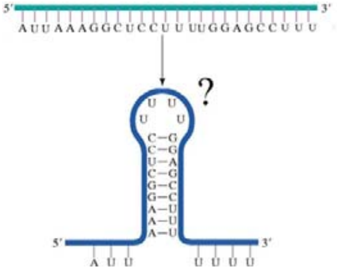

Figure 3: RNA secondary structure prediction

The formation of secondary structure of RNA can be shown by a few grammar rules [3]:

$$S \rightarrow cSg|gSc|aSu|uSa|gSu|uSg$$

$$S \rightarrow aS|gS|uS|cS$$

$$S \rightarrow a|g|u|c$$

$$S \rightarrow SS$$

The first rule pairs either a Watson-Crick base pair or a GU pair. The second rule is used to form the regions in which the bases are unpaired (such as loops). The third rule accepts a sequence of the length of one nucleotide as a valid RNA sequence and the last one can be used to join two structures together. These rules are relatively simple, but at the same time ambiguous. The reason for ambiguity is that there might be more than one derivation for the same sequence, e.g. for the short sequence *cccg*, we have:

$$S \Rightarrow cS \Rightarrow ccSg \Rightarrow cccg$$

$$S \Rightarrow cSg \Rightarrow ccSg \Rightarrow cccg$$

The translation of the RNA grammar rules in CHRG format would be :

```
(1) [c],s,[g] ::> s.
(2) [a],s,[u] ::> s.
(3) [g],s,[u] ::> s.
(4) s,s ::> s.
(5) [a] ::> s.
(6) [g] ::> s.
(7) [c] ::> s.
(8) [u] ::> s.
```

The first three rules should also be written for GC, UA and UG base pairs as well. This piece of code is very simple and it can be directly executed on a Sicstus Prolog engine (the code needed for CHRG should be compiled first).

This program will try to find all the possible secondary structures for the sequence, but it would probably take a long time to finish. Other techniques of RNA secondary structure prediction have some criteria for choosing the best solution, e.g. the one with the minimum free energy. According to these, we have to add some features to the existing rules to somehow prioritize in selecting rules. For instance while parsing a sequence when it comes up with a base C, it has to decide between choosing rule (1) and rule (7) (the current code accepts both of them and follows both branches according to that). To give priority to the rules, we have to find the probabilities that govern the known RNA secondary structures (which have been found by biological or computational methods). Sakakibara et al. have tried to solve this problem by using a stochastic context free model [24]. We suggest another method for finding these probabilities by using Constraint Handling Rules. For the input, we use a number of RNA sequences for which the secondary structure is known and we give this input to a program similar to the one written in CHRG. This new program keeps track of the number of times each of the rules has been used (to make the input structure) and at the end by running the program for each structure, we would be able calculate the probability of each rule according to those numbers and find the average between all the input structures. The reason why we used CHR here is that this language gives us a bottom-up framework that can be exploited to find the possible parse for each structure.

4.2.2 RNA secondary structure design

As mentioned earlier, the problem of RNA secondary structure design is defined as follows: given an RNA secondary structure, find an RNA sequence which folds onto the given structure (Figure 4). Major motivations to solve this problem include design of artificial RNA nanostructures, drug design, and ribozyme therapy [2]. Moreover, finding a solution to the problem of RNA secondary structure design might assist in solving the same problem for DNA (due to their analogous structures) which consequently can be used in DNA self-assembly computation. Complexity-wise this problem is believed to be computationally hard [2] but it has not been proven yet.

Previously there were only two major algorithms for solving this problem: RNAinverse [20] and RNA-SSD [2]. These methods generate outstanding results while designing shorter RNA sequences (less than 500 bases), however because of a bottle-neck in their algorithm which involves repeated calls to RNA secondary structure algorithms, they happen to be very slow while designing longer RNA sequences. In [6], we have introduced a new algorithm for solving this problem which uses the RNA grammar discussed above, and a nucleotide composition probabilistic model. Moreover, we have used CHR rules to implement the RNA grammar.

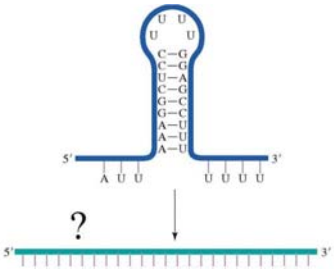

Figure 4: RNA secondary structure design

We have calculated these nucleotide composition probabilities by comparing a number of RNA sequences from several RNA database. After comparing 100 test cases with various length from 100 to 1500 bases, we found the following probabilities for each base pair:

$$P_{CG} = 0.53, P_{AU} = 0.35, P_{GU} = 0.12$$

The other probabilities which are of interest are the probabilities for an unpaired base to be one of A, C, G, or U. The results are as follows:

$$P_G = 0.18, P_A = 0.34, P_C = 0.27, P_U = 0.20$$

Inserting the probabilities into the rules is the most challenging part of the implementation. In our implementation, it is done by generating a random variable in the guard section of the rules, which is the only part that accepts Prolog predicates. This random variable is then tested according to the probabilities: for instance for the following rule if the random variable `I` is less than 0.53, it will assign a GC pair to `pair(X1,Y1)`. The `L` parameter in `s` contains the list of bases already added to the sequence and `find(M,N,I)` assigns a base pair to `M` and `N` based on the random variable `I`.

```
pair(X1,Y1)\s(X,Y,L) <=> X1 is X-1,Y1 is Y+1,random(I),
find(M,N,I), append([X1:M,Y1:N],L,L1) | s(X1,Y1,L1).
```

All the other rules inside the RNA grammar are translated to CHR rules combined with probabilities similar to the rule above.

4.3 Protein structure

Proteins are the molecules responsible for much of the structure and activities of organisms. From a chemical point of view, proteins are long polymers containing thousands of atoms. The proteins are typically 200–400 amino acids long which means that they require at least 600–1200 letters for the DNA message to specify them (not including the introns).

Protein architecture could be analyzed in three levels like RNA and DNA. Primary structure would be the order of the amino acids in the sequence which has been formed by three codons translation. Secondary structure however represents how some of the amino acids in the sequence form common structures such as *alpha helices*, *beta sheets*, etc. Alpha helices are common structures of proteins, characterized by a single, spiral chain of amino acids stabilized by hydrogen bonds, while beta sheets consist of two or more amino acid sequences within the same protein that are arranged adjacently and in parallel, but with alternating orientation such that hydrogen bonds can form between the two strands. Finally the tertiary structure would be the exact conformation of a whole protein.

Here we only discuss the problem of *protein secondary structure prediction*. What makes this problem rather difficult is that identical short sequences of amino acids can adopt different secondary structure in different contexts [27]. Muggleton et al. have tried to solve this problem by using Inductive Logic Programming (ILP) [21, 22]. They have applied an ILP program to learn secondary structure prediction rules. The output of their program is a small set of rules which can predict which of the residues in the sequence are part of the alpha helices. This set of rules can also be translated into CHRG. A sample rule in CHRG format is:

```
res(X1),res(_),res(X2),res(X3), res(Y),res(X4), res(X5),_,X6) ::>
not_aromatic(X1),not_k(X1), hydrophobic(X2),
not_aromatic(X3),not_p(X3),
not_aromatic(Y),not_p(Y), not_p(X4), not_k(X4),
hydrophobic(X5),less_hydro(X1,X5),
less_volume(X5,X3),not_aromatic(X6),
less_hydro(X6,X2) | alpha(Y) .
```

In this example, `res/1` represents a residue and the predicates in the guard are the chemical characteristics that each residue should have.

5 Conclusion

We have covered several high level methods for pattern description of biological sequences, and exemplified the advantages of these methods for several concrete such problems. We have shown how to translate codons to aminoacids using DCGs, how Assumption Grammars allow us to modularize stack manipulation in global terms, which we used for implementing tandem and inverted repeats, how Constraint Handling Rules promote direct bottom-up execution of the same problem, as well as having a grammatical version which we used for more direct aminoacid string translations, and how Concept Formation Grammars can in particular accommodate rules with exceptions.

We consider the main advantage of these methods to be the coupling of direct executability with economy of expression within high level specifications that are meaningful for humans and thus can promote quick prototyping of specialized, interdisciplinary knowledge. On the other hand, the discussed methodologies do exhibit some run-time inefficiencies. For future work, some intermediate systems can be designed to automatically translate these high-level languages into low-level ones. This way we would have methods that are not only understandable and more user friendly but at the same time efficient and practical.

With this work we hope to stimulate further interactions between molecular biology, logic and AI.

References

1. Abramson, H. , Dahl, V.,*Logic Grammars*, Springer-Verlag, 1989.
2. Andronescu, M., Fejes, A.P., Hutter, F., Hoos, H.H.,Condon, A., *A new algorithm for RNA secondary structure design,* Journal of Mol. Bio. 336(3), 607–624, 2004.
3. Baldi, P., *Bioinformatics: the machine learning approach*, Cambridge, Mass. :MIT Press, 1998.
4. Barranco-Mendoza, A., *Stochastic and Heuristic Modelling for Analysis of the Growth of Pre-Invasive Lesions and for a Multidisciplinary Approach to Early Cancer Diagnosis*, Ph.D. Thesis, Simon Fraser University, Burnaby, BC, 2005.
5. Barranco-Mendoza, A., Persaoud, D.R. and Dahl, V. *A property-based model for lung cancer diagnosis*, 8th Annual Int. Conf. on Computational Molecular Biology, RECOMB 2004, San Diego, California, March 27-31 (accepted poster), 2004.
6. Bavarian, M., Dahl, V., *RNA secondary structure design using Constraint Handling Rules* , WCB 2005.
7. Bes, G. and Blache, P., *Proprieties et analyse d'un langage*, In Proc. TALN99, 1999.
8. Bes, G., Dahl, V., Guillot, D., Lamadon, L., Milutinovici. I. and Paulo, J.,*A parsing system for balanced parenthesis in NL texts.* Poster and Demo at the Lorraine-Saarland Workshop on Prospects and Advances in the Syntax/Semantics Interface, Loria-Nancy ,2003.
9. Blache, P. and Balfourier, J. M., *Property Grammars: a Flexible Constraint-based Approach to Parsing*, In Proc. IWPT-2001, 2001.
10. Christiansen, H., *CHR as Grammar Formalism, a First Report*, In *Sixth Annual Workshop of the ERCIM Working Group on Constraints*, Apt, K.R., Bartak, R., Monfroy, E., Rossi, F., (eds.), Prague, 2001.
11. Christiansen, H.,*Logical Grammars Based on Constraint Handling Rules*,(Poster abstract). In Proc. *18th International Conference on Logic Programming*, Stuckey, P. (ed.) Lecture Notes in Computer Science, 2401, Springer-Verlang, p. 481, 2002.
12. Christiansen, H. *CHR Grammars*, International Journal in Theory and Practice of Logic Programming, special issue on Constraint Handling Rules, 2005.

13. Cohen, J., *Computational Molecular Biology: A Promising Application Using Logic Programming and Constraint Logic Programming*, Lecture Notes in Artificial Intelligence, 1999.
14. Dahl, V. and Blache, P., *Directly Executable Constraint Based Grammars*,In Proc. *Journees Francophones de Programmation en Logique avec Contraintes*, Angers, France, 149–166, 2004.
15. Dahl, V. and Tarau, P.,*Assumptive Logic Programming*, In Proc. *ASAI 2004*, Cordoba, Argentina, 2004.
16. Dahl, V., Tarau, P., and Li, R.,*Assumption Grammars for Processing Natural Language*. In Proc. *Fourteenth International Conference on Logic Programming*, MIT Press, 256–270, 1997.
17. Dahl, V. and Voll, K., *Concept Formation Rules: An Executable Cognitive Model of Knowledge Construction*, In Proc. First International Workshop on Natural Language Understanding and Cognitive Sciences (NLUCS'04), Porto, Portugal, April 2004.
18. Frühwirth, T.,*User-Defined Constraint Handling*, Poster, International Conference on Logic Programming (ICLP 93), Budapest, Hungary, MIT Press, June 1993.
19. Frühwirth, T. W.,*Theory and Practice of Constraint Handling Rules*,In *Journal of Logic Programming*, 37:95–138, 1998.
20. Hofacker, I.L., Fontana, W., Stadler, P.F., Bonhoeffer, S., Tacker, M., Schuster, P., *Fast Folding and Comparison of RNA Secondary Structures* , Monatshefte f. Chemie 125:167–188, 1994.
21. Muggleton, S., *Inductive logic programming*, Proceedings of the ILP-91 international workshop, Vianna de Castelo, Portugal, 2-4 March 1991.
22. Muggleton, S.,King,R.D., Sternberg.,M.J.E.,*Protein secondary structure prediction using logic-based machine learning*, Protein Engineering, 5:647-657, 1992.
23. Overbeek, R.A., *Invited Tutorial: Logic Programming and Genetic Sequence Analysis*, JICSLP, Washington, DC,1992.
24. Sakakibara, Y., Brown, M., Hughey, R., Mian, S., Sjolander, K., Underwood, R., Haussler, D. *Stochastic Context-Free Grammars for tRNA Modeling*, Nucleic Acids Research. 1994 Nov 25;22(23):5112-20.
25. Searls, D.B., *The computational linguistics of biological sequences*, Artificial intelligence and molecular biology, American Association for Artificial Intelligence, p 47-120, 1993.
26. Searls, D.B., *Grand challenges in computational biology*, In *Computational methods in molecular biology*, S. L. Salzberg, D. B. Searls and s. Kasif, Eds. Elsevier Amsterdam, The Netherlands, 1998.
27. Schulze-Kremer, S., *Molecular bioinformatics: algorithms and applications*, Water de Gruyter, New York 1996, p12.
28. Voll, K.,*A Methodology of Error Detection: Improving Speech Recognition in Radiology* , PhD thesis, Simon Fraser University, 2006.
29. Waterman, M. S., *Introduction to computational molecular biology: Maps , sequences and genome*,Chapman&Hall, p327-340, 1995.
30. Zahariev, M., Dahl, V. and Levesque, A. (Technical Report), *Efficient Algorithms for the Discovery of Oligonucleotide signatures for DNA Sequences and Groups of Sequences.*
31. Zuker, M., *Sciences* 244, 48-52, 1989.