



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

*Logol : Modelling evolving sequence families  
through a dedicated constrained string language*

Catherine Belleannée — Jacques Nicolas

N° 6350

November 2007

Thème BIO

*R*apport  
de recherche





## Logol : Modelling evolving sequence families through a dedicated constrained string language

Catherine Belleannée, Jacques Nicolas

Thème BIO — Systèmes biologiques  
Projet Symbiose

Rapport de recherche n° 6350 — November 2007 — 19 pages

**Abstract:** The report reviews the key milestones that have been reached so far in applying formal languages to the analysis of genomic sequences. Then it introduces a new modelling language, Logol, that aims at expressing more easily complex structures on genomic sequences. It is based on a development of String Variable Grammars, a formal framework proposed by D. Searls.

**Key-words:** Bioinformatics, Sequence Modelling, Formal languages, Logical Grammars, Constraints, String variable

## **Logol : modéliser l'évolution de familles de séquences biologiques au moyen d'un langage dédié à base de contraintes**

**Résumé :** Ce document concerne l'utilisation de langages formels au service de la modélisation de séquences génomiques. Dans un premier temps, une étude des travaux fondateurs sur ce sujet est développée. Ensuite, un nouveau langage de modélisation, Logol, est présenté. Logol vise à exprimer plus facilement des structures complexes sur des séquences génomiques. Il est basé sur un développement des grammaires à variable de chaînes (SVG), structure formelle proposée par David Searls pour la modélisation biologique.

**Mots-clés :** Bioinformatique, modélisation de séquences, langages formels, grammaires logiques, contraintes, variables de chaînes

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Modelling with formal languages</b>	<b>4</b>
2.1	From regular to context free grammars . . . . .	4
2.2	String Variables . . . . .	7
2.3	Parsers in practice . . . . .	8
<b>3</b>	<b>Logol: a string based language</b>	<b>9</b>
3.1	Duplication of genomic sequences . . . . .	9
3.2	Multipatterns in genomic sequences . . . . .	10
3.3	Parsing strategies and optimization . . . . .	11
3.4	Basics of Logol . . . . .	12
<b>4</b>	<b>Discussion and Conclusion</b>	<b>15</b>

## 1 Introduction

Comparison of new sequences with known ones is a routine for molecular biologists trying to decipher the function of these new sequences from similarities observed between the chains of acids in the macromolecules. This is achieved via alignment algorithms like Blast [2], that find the best matching between positions in a given set of sequences.

However, with the increase of knowledge on biological mechanisms, active sites and spatial structures, it becomes possible to be less dependent on the existence of relevant alignments with similar molecules, position by position. The idea is to build richer characteristic models of families of sequences, on the basis of their content and structure. This offers both an integrated view of each family and a better predictive accuracy while annotating new sequences.

The theory of formal languages is a natural framework to study models on sets of sequences. In fact, regular expressions are often used to query biological data bases and to look for particular sequences. But a closer inspection of natural structures in genomic sequences shows the need for more expression power, requiring even non algebraic grammars. This places the class of genetic languages at the same level as natural languages, in a class that A. Joshi called “midly context sensitive languages” [16]. Some attempts have been made in the past to propose biologists a high level generic modelling language belonging to this class. It seems however that they bring only a partial answer to this need for modelling and furthermore that practical tools are lacking for this purpose.

We review some of the most important contributions in this context and try to exhibit their strengths and some weaknesses that may have put a break on their spreading. Then we propose to develop a new language, called Logol, taking into account the accumulated experience and based on a careful biological foundation of the string object such as for instance the fact that observed sequences are variants that have evolved from a common ancestor.

## 2 Modelling with formal languages

### 2.1 From regular to context free grammars

To the best of our knowledge, one of the first serious study mentioning the use of formal languages in order to describe structures on genomes dates back from 1984 [7]. The paper describes on an interesting example of RNA phages the possibility to model such elements by a combination of finite state automata and to model the translation of their genes by transducers. The basic idea is that genomes result from a series of various constraints, each constraint imposing a language, that is, a set of possible sequences. A genome may be represented by the intersection of all these languages. Brendel and Busse suggest that regular languages suit very well to this task since most characteristic motifs are simple (regular) and the intersection and the product of regular languages are regular languages.

The interest of regular languages seems obvious in describing families of proteins. Indeed, active sites within proteins have a relatively precise location and the representation of the family can be reduced to a regular expression matching possible sequences of sites. It is even possible to be restricted on a subset of

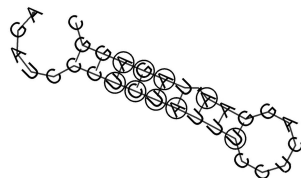


Figure 1: Secondary structure of Levivirus, positions 3349-78 in MS2

regular languages, using expressions of type  $\Sigma^* E \Sigma^*$ , where  $E$  is a finite language using disjunction only on letters (Prositate syntax, [8]). A recent paper in Nature [18] shows the interest of such an elementary level of modelling and the fact that it is still far from being an evidence in biology. The authors worked on an important class of proteins with natural antimicrobial and antifungal properties. They built a finite language by tiling Prositate patterns of fixed length characteristic of this family and obtained larger regular patterns allowing the production of new putative active peptides. Regular languages may also be useful on DNA sequences. Kangaroo [4], a simple tool recognizing regular expressions, has been applied on the colorectal cancer typing issue. This problem is reduced to the search of words made of repeated letters in genes, which more easily produce pathogenic mutations.

There exists numerous free available tools for the recognition of regular expressions on genetic sequences (e.g Fuzznuc in Emboss [27], or tacg [20] also recognising matrices and boolean expressions).

However, regular languages are not fully adequate for more complex structures:

- even for simple viral genomes, describing the observed conserved secondary structures in RNA families needs more expressive power. As an example, consider the 3' side of synthetase gene in Leviviridae. A common stem-loop structure is conserved in all elements of the family, displayed in figure 1, but conservation exists on the *structure* itself, not on the sequence. This is emphasized by circled nucleic acids on the figure, which exhibit at least one mutation in at least one species with respect to consensus. For instance, the corresponding sequence of the virus MS2 belonging to this family writes

**CGGUUCC**CACAUUCCUCAGGAGUGUGGGC

whereas consensus sequence is (mutations in bold) :

**AGA**UCCUCUAUUCCUCAGGAAU**AG**AGGC.

Mutations do not appear isolated : the word "UCU" that is matching its dyadic symmetric word "AGA" with respect to Watson-Crick bonds in the consensus becomes "CAC" matching with "GUG" for MS2. This probably fonctionnal conservation requires an algebraic language, and this class is no more close under intersection. In fact, even non algebraic structures are known in the RNA world such as pseudo-knots (see figure 2).

- Brendel and Busse viewed regular languages only as a mathematical framework to better reason on biological models. If one aims at analysing real sequences, a finer level of description is often necessary and involves more complex constructions. For instance, how to write simply the fact that

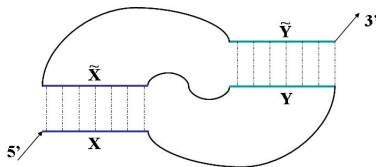


Figure 2: RNA structure of pseudo-knot

sequence "GGAG" observed in front of the capsid gene of virus MS2 is an instance of a more general model which is a "sufficient" subsequence of the so-called Shine-Dalgarno sequence "AGGAGG", or how to describe correctly overlapping sequences of genes in such a virus ? It is not only a question of language but also a question of grammar. That is, the way models are described in term of structures is important because it enlightens spatial structures and biologic mechanisms applying on sequences.

- A last point concerns the fact that every living organism is a dynamical system and that several alternative structures may be produced depending on the current environment via regulation mechanisms. At the level of modelling, this means ambiguous grammars must be allowed. More generally, the set of processes that act on macromolecules and metabolites (copy, transcription, splicing, translation...) may be interpreted as transformations between several chains that have to be addressed in modelling.

According to these more sophisticated needs, a number of specific pattern matching tools have been developed and made available. RNA modelling has been particularly well developed, making use of specific a priori grammars [14, 28, 29]. These methods may achieve a level of accuracy that competes with the accuracy of complex energy minimization methods with more parameters [11].

Although very useful, such studies are not generic enough to take into account the various constraints that add up until the formation of complicated structures. For instance, in order to recognize frameshift regions in mRNA molecules, one must not only recognize a spatial structure (a pseudo-knot) but also a region in front of it, called a sliding window. It is made up of 9 bases (e.g "uuuaaaca") that are responsible of -1 shift of the reading frame during translation from mRNA to protein. Authors in [34] use a specific Tree Adjoining Grammar to represent frame-shifted sequences. However, it is not very productive to design for each possible family a new program and it is not feasible for a non-programmer to check its own models without help. Thus, some people have tried to design true modelling languages, where the description of the model is independant from the parser. Palingol [5] is a small language dedicated to the analysis of RNA. It allows to express constraints on the size and the composition of constitutive parts as well as constraints on the distances and relations between these parts. Sharing characteristics of constrained languages, Palingol remains mainly a dedicated programming language.

## 2.2 String Variables

The major contributions to genomic sequences modelling are due to D. Searls, which has published many papers on this topic and laid the foundations of a serious research in the domain. The article [31] offers a good synthesis on his theoretical and practical work. He was the first to supervise developments allowing users to conceive grammars and parse real genomic sequences with them [32, 10].

One of the key ideas of D. Searls is to try to find a balance between the well-founded framework of algebraic languages that offer a good expressivity/efficiency tradeoff and the necessity to describe easily basic biological mechanisms such as copy (direct or reverse) that is at the core of the evolution of genomes. He has proposed to introduce in context-free grammars a new kind of object for this purpose, the *string variable*. The resulting formalism is called SVG, for String Variable Grammars. An example of SVG rule is  $\text{foo} \rightarrow X, \text{"aca"}, X$ . It expresses that `foo` accepts every string made of two copies of a same string, separated by `"aca"` (`"ctacact"` is a `foo` for instance). This concept of string variable that deals with copies was not new. It had been introduced by D. Angluin in so-called pattern languages [3]. String variables are mathematical variables that must represent (be uniformly instantiated with) a same word in all its occurrences (generally excluding the empty word). The class of pattern languages is included in context sensitive languages and the membership of a word to a pattern is a NP-complete problem.

Although it is seldom used on biological data, a notion of string variable called *backreference* [1] exists in most practical tools for the search of -extended- regular expressions (egrep, awk, sed, perl, python, JavaScript,...). A nice development of the idea is proposed in [9] with the *synchronized regular expressions* (SRE), where two types of variables may be used : *string variables* whose instantiation domain has to be declared by a subexpression and *power variables* that allow to describe finite repetitions and have to be substituted by an integer. The paper mention the possibility to use them on DNA sequences but we are only aware of a recent tool using a very restricted type of SRE, Pattern Locator [21] (backreferences only to given positions in the pattern). SRE contain pattern languages and regular languages but are not comparable with context free languages (curiously, copy languages are in SRE but palindromic languages are not !). The membership of a word to a SRE remains a NP-complete problem, even if only power variables are allowed.

Variables also appear in the framework of logic programming (LP) that Searls used, but variables of LP do not have the same power of expression as string variables. Indeed, in the context of LP, strings are represented by right-branching binary trees that determine their possible segmentations: one can access either to a word of fixed size at the head or to the tail of the list of characters. CLP (Constraint Logic Programming) is an extension of LP replacing unification of terms with specialized constraints solvers. Because of the complexity of the general problem, no practical solver exists for strings, except for restricted constraints. An interesting prototype, S-log, has been proposed by A. Rajasekar [26], where variables are couples  $W:t$ ,  $W$  being a sequence that can unifies with a sequence of fixed size  $t$ . The corresponding reduction algorithm can for instance address the equation  $(X:2).(Y:3) = (Y:3).(X:2)$ , giving the solution  $X=(Z:1).(Z:1)$ ,  $Y=(Z:1).(Z:1).(Z:1)$ . Another aspect

of LP is to work on attributed grammars, where non-terminals may carry some parameters. As these parameters (attributes) may be string variables, this allows non-terminals to transmit string variables to sub-models. From the point of view of expressivity on biological sequences, this allows to take into account hierarchical aspects of life and to introduce contextual aspects.

In his formal work, Searls distinguishes two alphabets, a specification alphabet and a sequence alphabet in order to take into account the various transformations acting on sequences. These transformations have to be expressed via oriented substitutions (direct or reverse) mapping an alphabet to the set of subsets of the other.

SVG include context free grammars and are included in indexed grammars. They differ strictly from Tree Adjoining Grammars and Reduplication Push-Down Automata. Formally, SVG are defined as follows:

**definition 1** A *SVG* is an 8-tuple  $(\Delta, \Gamma, \Sigma, N, V, S, F, P)$  where:

- $\Delta, \Gamma$  et  $\Sigma$  are respectively the specification alphabet and the alphabets of functions and sequences ;
- $N$  and  $V$  are the sets of non-terminals and variables;
- $S$  is the initial axiom of the grammar;
- $F$  is a set of finite substitutions from  $\Delta$  to  $2^\Sigma$  associated to symbols of  $\Gamma$ . Substitutions are extended from letters to words and signed: for every substitution  $f$  and every word  $aW$  of  $\Delta^+$ ,  $f^+(aW) = f(a).f^+(W)$  and  $f^-(aW) = f^-(W).f(a)$  ( $\cdot$  is a set product) ;
- $P$  is a set of production rules  $A \rightarrow \Phi$  or  $B(X) \rightarrow \Phi$ , with  $A \in N$ ,  $B \in N - \{S\}$ ,  $X \in V$  and  $\Phi \in (\Sigma \cup N \cup (V \times \{+, -\} \times \Gamma) \cup (N \times V))^*$

### 2.3 Parsers in practice

Beyond this formalization, a few authors have proposed a generic parser for genetic sequences in the framework of LP. Among these, only Hypasearch is still available, for academic<sup>1</sup>. The others, even not available anymore, convey additional concepts that are worth considering.

Genlang, proposed by Searls [10] is the most complete. It adds three main ideas to SVG:

1. A set of attributes is attached to each non-terminal, related to parsing. The system manages the errors in analysis (cost), positions of the model (span), content and size of the parsed sequence. One can put constraints on these attributes in the right part of rules.
2. There exists also a position variable, that may be handled through operator @. For instance, @ $X$ , where  $X$  is a variable, unifies with the current position in the analysed sequence. Once unified, other occurrences of the variable allow either to put some constraints on the current position or to set a position at a given value.

<sup>1</sup><http://onyx.biophys.uni-duesseldorf.de/hypa/>

3. Parsing strategy may be partially controlled through a complex set of primitives. In the right part of rules, one may add specific non-terminals that derive in the empty string with side effects on the strategy. For instance, firing the rule  $S \rightarrow A, @stay, B.$  will start the analysis of  $A$  then  $B$  but will return only the result of the analysis of  $A$  in case of success. *skip* has the original side effect of requiring all solutions to be non-overlapping.

All these added primitives are fundamental in practice to get operational models but need a good level of training to be used.

Among studies close to Genlang, are Hypasearch, BGG (Basic Gene Grammars) and PALM. Hypasearch, proposed by Kurtz [13, 33], is based on the modelling language HyPaL. Among properties of HyPaL are:

1. HyPaL is entirely declarative: contrary to Genlang, it is not possible to specify the parsing strategy.
2. As for Genlang, HyPaL allows for user-defined scoring functions (such as profiles or consensus matrices) and user-defined constraints (properties like energy values or molecular weights of sequences). These constraints constitute an “hybrid part” of the sequence patterns that move HypaL away from a pure language.
3. HyPaL has been particularly developed for the field of RNA structures and proposes a library of predefined models in the domain, HypaLib.

Hypasearch relies on suffix arrays for a first indexing of sequences and an efficient search.

BGG (Basic Gene Grammars) [17] insists on chart parsing strategies and introduces two descriptors for this purpose, one indicating overlapping of analyses and the other left or right priority for the parsing order. PALM [15] is a direct competitor of Genlang, but has only been developed as a prototype. It includes interesting additional characteristics, repeats and a limited form of negation, which should deserve further developments. Another generic parser has been developed for the language proposed by A. Brazma and D. Gilbert [6], slightly less expressive than Genlang, including a CLP(FD) version and a CSP version [12], both for demonstration purposes only.

### 3 Logol: a string based language

The state of the art and the experience gained from user queries on our bioinformatics platform clearly demonstrate the need for high level languages to describe the structures of life. We propose now to elaborate on strengths and weaknesses of current propositions, and to introduce a new language, Logol, which could be a further step towards a fully fledged modelling language. Overall, Logol is designed as a practical constrained string logical language. A characteristic sample of the language syntax is available in table 1.

#### 3.1 Duplication of genomic sequences

Genomic sequences evolve through a fundamental duplication process followed by mutations or errors. The notion of string variable captures the fact that

subwords may appear several times in a sequence. In all languages we have described so far, the possible variations between occurrences of a string variable are taken into account by introducing a notion of cost. More precisely, taking the string of the first occurrence, it is possible to constrain the other to be strings at a fixed maximal distance from it. This process is not at all symmetrical and does not correspond to the needs in biology.

What we observe in a set of biological sequences of a same family generally derives from a common ancestor, but each sequence contains its own set of mutations. The difficulty of modelling and detecting such a repeat is that the reference ancestor sequence may well not appear at all in the sequence. For instance, AATT may be an observed string made of two copies, AA and TT, of a same ancestor TA that is not observed, and it is simply impossible to formalize it with current patterns languages. In contrast, LOGOL language let to mention and refer such an abstract pattern, even if it does not appear in the sequence. In our example, it is sufficient to use a model like  $X:1, X:1$ . This model, using a string variable  $X$ , means that a same word  $X$  has been derived with two independent single mutations. Note that it is different from a model  $Y, Y:2$  in the sense that  $X$  gets the value of the ancestor, which is not the case for  $Y$ .

More generally, Logol considers two levels for chains (i.e. series of bases): a level which designates abstract chains - called *words*-, that is "ideal" ones, not necessarily existing in the observed sequences (TA in our previous example), and a level which designate concrete subsequences -called *strings*-, which are parts of the observed sequences ( AA or TT in our example).

Models in Logol deal with sets of words representing the abstract strings we are looking for, and sets of constraints indicating which strings can be accepted as instances of the words, such as localization of the string in the sequence (they are *string constraints*) or such as admissible distortions (they are *structure constraints*).

From a computational point of view, the issue of finding a common representation of a set of words clearly pertains to machine learning. The system Winnower addresses this point [24]. It is also related to the center and median string problem [22]. It is a NP-complete problem and in practice only bounded cases may be solved. It is the case here since there exists a fixed, generally small number of copies and the maximal distance to the center is fixed.

### 3.2 Multipatterns in genomic sequences

Another strong feature of genomic sequences is that in order to adapt to their environment, they have to maintain multiple alternatives in their code. Due to finite resources, and particularly for small organisms, this results in a high compactness of coded information.

Overlapping of genes is found throughout nature, from bacteria and viruses where genes are often very close to mammals, including humans, within gene-rich regions [19]. The alternative possible translation of these genes sharing a portion of code belongs probably to important regulation mechanisms. A quite different type of alternative interpretation of a same code has been described by D. Searls on RNA with attenuators, which correspond to a fine control mechanism on the expression of bacterial genes. Attenuators contain a sequence that may match with two alternative complementary sequences forming in each case a stem structure active or not for the termination of transcription. Finally,

we can mention competing overlapping transcriptor factor sites as another key factor in the regulation of expression.

It is almost impossible to get a model of such fundamental mechanisms with available languages. With Genlang, it is possible to backtrack at some point in the sequence and to analyse multiple overlapping motifs this way, but to the cost of heavy descriptions and unnatural structures.

From a formal point of view, we propose to handle these types of ambiguity at two levels.

At the lowest level, Logol use two types of commas to describe the succession of entities in a sequence. The first one (usual comma) refers to the succession of the end of an entity with the start of the next that is the standard way of describing consecutive words. The second one (semicolon) refers to contiguous starting positions of each entity. Thus, `E1,E2` means contiguity (e.g. `"AAT","ATGG"` stands for `"AATATGG"`) and `E1;E2` means overlapping (`"AAT";"ATGG"` stands for `"AATGG"`).

At a higher lever, one can express competing views on a same string. To be an instance of the model, a sequence must be an instance of each view of the model. For instance, the 2 views model `("TA",X)&(X,"AT")` will recognize sequences that start with `"TA"` and end with `"AT"`, including the word `"TAT"`.

Note that genome sequences are obtained by the concatenation of small overlapping sequenced segments. A model might also track possible assembly errors in this context.

Tandem repeats or spaced repeats are successive copies of a same entity that are very frequent in genomic sequences. Logol introduces a repeat construct managing a counter of occurrences. Contrary to SRE, we have not introduced this counter as a new type of variable. It would have introduced an unnecessary level of complexity since it is not supported by known structures in biological sequences. In Logol, repeat is denoted as `(Entity Comma Distance)+NbOccur`, where `Entity` is the repeated entity, `Comma` is the succession symbol and `Distance` and `NbOccur` are optional arguments that detail the integer constraints on the series. For instance : `("ACGA",[3,5])+[7]` will match strings containing seven repetitions of `"ACGA"`, without overlapping, with a distance between 3 and 5 separating two contiguous instances while `("ACGA";[3,5])+[7]` is a similar model where successive instances of `"ACGA"` may overlap due to the use of semicolon.

Properties such as the prevalence of overlapping consecutive repeats in tandem repeats has never been checked for in biological sequences due to the lack of such expressivity. Clearly, it could lead to finer understanding of the way they are generated.

The combination of string variables, overlapping successions and views allow to express any equation on strings and is potentially hard to solve. However, we consider only fixed finite sequences and practical models contain a limited number of variables and constraints on the length of strings. We think there exists in this context a large range of possible research developments on the topic of combining parsing and constraint string solving [25] techniques.

### 3.3 Parsing strategies and optimization

A crucial point for the application of rewriting formalisms on real data is the often hidden control part. Parsing a complex model generally requires a careful

tuning of the parsing strategy in order to avoid both useless searches (redundant or locally underconstrained) and production of irrelevant solutions. This point has been partly addressed in Basic Gene Grammars [17] with a general chart parsing strategy and treated in depth in Genlang with the introduction of a number of control primitives. However, introducing such primitives shifts the status of Genlang from a modelling language to a programming language and it is an essential limitation for a large spreading of the tool. A biologist needs a heavy initial commitment to be autonomous with Genlang.

Our approach has been to elucidate the main situations where biological models have to be refined at the level of parsing strategy and express them in constraints associated to entities of the model. In most cases, sequences have evolved under selection pressure until they have achieved a stable, locally optimal, configuration. Most of the time, the biologist needs to express this basic fact. For instance, RNA structures have to exhibit a form of energy minimization in folding.

As a simple example, consider the succession of genes in bacterial sequences. A gene is defined as a series of codons, starting with a start codon ("ATG", "TTG" or "GTG") and finishing with a stop codon ("TAG", "TAA" or "TGA"). A simple grammar rule ( $\text{gene} \rightarrow \text{start}, \text{codons}, \text{stop}$ ) would in fact recognize any combination of start and stop, even if spanning several genes. Logol<sup>2</sup> offers two possibilities in such a case. One can first introduces a negation in the constraints on entity codons (codons is a suite of triplets excluding stop codons). Another possibility is to introduce an optimization constraint on the model (here, for a given start, stop must have a minimum starting position). In fact, in order to get all genes, we have to skip over a gene once it is a solution. This can be stated by another constraint on the position of the start codon, which has to be minimized with respect to the end of other solutions. More generally it is possible in Logol to add an *optimization constraint* to an entity on an identified string  $SX$  ( $SX$  is the ident of the string), using character '!'. Optimization constraints relate to the attributes of Logol entities (see below). They are applied sequentially, following their left to right ordering. Choosing among possible occurrences those with the minimum starting position for  $SX$ , the maximum ending position for  $SX$  or the maximum length is expressed respectively by  $!@SX$ ,  $!\$SX$ ,  $!\#SX$ . In the same way,  $!\$, !\%, !\epsilon, !\%\epsilon$ , are used respectively to choose among possible results the analysis with the minimum cost, minimum relative cost, minimum edit cost, minimum relative edit cost.

### 3.4 Basics of Logol

Logol allows to describe *models*, a representation of a language that is an abstract representation of a set of sequences. A model is composed of *constrained entities* (with Logol syntax: *Entity: String Constraints : Structure Constraints*). An *entity* is an expression that is either a string constant, or a string variable, or a non terminal, or a repeat, or a choice, or recursively of the form (*Model*). String constants and variables may be preceded by a morphism (e.g.  $+ \text{comp}(\text{"ATCC"})$  or  $- \text{func}(X)$ ). A morphism will transform a string by parsing its lexical units, following the given direction, and producing the concatenation of associated strings. For instance if  $\text{comp}$  is defined with rules  $\text{comp}(\text{"T"}) \rightarrow \text{"A"}$ ,

<sup>2</sup>See Logol expression in example 11 of table 1

$\text{comp}(\text{"G"}) \rightarrow \text{"T"}, \text{comp}(\text{"C"}) \rightarrow \text{"G"}, \text{comp}(\text{"A "}) \rightarrow \text{"C"}$ , the previous expression  $+\text{comp}(\text{"ATCC"})$  recognizes the string "CAGG" and  $-\text{comp}(\text{"ATCC"})$  recognizes "GGAC".

An instance of an entity in a sequence is a couple  $\langle \text{string}, \text{structure} \rangle$ , where both *string* and *structure* are objects characterized by attributes. *string* indicates which part of the sequence is an instance of the entity, *structure* indicates how the parsing/matching was done.

o A *string* is an object that represents an occurrence of a word in a sequence. Each string is characterized by five attributes: its name, content, start and end positions, and length. Accordingly, five accessors are defined:

- $\_$  or **ident** : access to the name (identity) of the string;
- $?$  or **content** : word of the occurrence;
- $@$  or **begin** : starting position of the occurrence;
- $\S$  or **end** : ending position of the occurrence;
- $\#$  or **length** : length of the occurrence.

o A *structure* is a term that reflects the matching of a string on the structure of a model. Each structure is characterized by three attributes: its name, content and cost. These attributes can be accessed by six accessors:

- $\_$  or **ident** : access to the name (identity) of the structure;
- $?$  or **content** : term of the structure;
- $\$$  or **cost** : substitution distance (only mutations) between the model and its occurrence (cost);
- $\%\$$  or **%cost** : ratio of the substitution distance by the length of the occurrence (relative cost);
- $\in$  or **distance** : edit distance (includes substitution, insertion and deletion) between the model and its occurrence;
- $\%\in$  or **%distance**: relative edit cost.

Constrained entities are organized in models via three connectors. Models may contain several views that are separated by  $\&$ . A view is a succession of sequences separated by  $'$ ,  $'$  in the non-overlapping case and  $'$ ;  $'$  in the overlapping case.

Formally, Logol grammars are based on an extension of SVG that we call Constrained String Variable Grammars (CSVG).

**definition 2** A *CSVG* is an 8-tuple  $(\Delta, \Gamma, \Sigma, N, V, S, F, P)$  where:

- $\Delta, \Gamma$  and  $\Sigma$  are respectively the alphabets of string ident, morphisms and sequences;
- $N = \bigcup N_k$  is the set of non-terminals of fixed arity  $k$ ;
- $V$  is the set of string variables.  $SV = \Sigma^* \cup V$ .
- $S \in N_0$  is the initial axiom of the grammar;
- $F$  is a set of finite substitutions from  $\Sigma$  to  $2^\Sigma$  associated to symbols of  $\Gamma$ . Substitutions are extended from letters to words and signed: for every substitution  $f$  and every word  $aW$  of  $\Sigma^+$ ,  $f^+(aW) = f(a).f^+(W)$  and  $f^-(aW) = f^-(W).f(a)$  ( $\cdot$  is a set product);
- $P$  is a set of production rules  $A(X_1, \dots, X_k) \rightarrow \Phi$ , with  $A \in N_k$ ,  $X_i \in V$  for  $1 \leq i \leq k$  and  $\Phi$  is a set of elements of  $R = (\mathcal{E} \times (\{, \} \cup \{ ; \})) \times \mathcal{C}^*$ ,

<p>1. <i>Model</i> ("CA "A"), "AATC", foo with foo → "TA "CT"  Matches for instance the sequence "AAATCTA". foo is a non-terminal.  A succession of constrained entities is at the basis of model description. Here, contiguous succession (because of ',') of entities without constraints.</p>
<p>2. <i>String variables</i> X, "GAG", -, X, -  Matches for instance the sequence "GTTGAGAGTTGA"  This model includes the two types of string variables (-, called mute variable, and X). It matches the sequence with X= "GTTGA" (both mute variable values empty) and also with X= "GTT". As in Prolog two mute variables correspond to different variables and may have different values. It is useful for describing gaps.</p>
<p>3. <i>String variable with morphism</i> X, -wc X  with wc("T")→"A", wc("G")→"C", wc("C")→"G", wc("A")→"T".  Matches for instance "TCGCGA" with X="TCG". This model describes a biological palindrome. It includes an occurrence of X, followed by a transformation of X by a reverse morphism (direction -) described by non-terminal wc.</p>
<p>4. <i>Overlapping succession</i> X, "GAG"; (-, X)  Matches for instance "GATTGAGATT", with X= "GATT" and -= "A". Here, (-, X) entity overlaps "GAG" entity.</p>
<p>5. <i>Views</i> -, "ATCCGA", - &amp; -, "TAGTAT", -  Matches for instance the sequence "CCTAGTATCCGATAC".  This model states that to be admissible, a sequence must at the same time contains "ATCCGA" and "TAGTAG", no matter of their respective locations.</p>
<p>6. <i>Strings constraints: constraints on length</i> X : {#[2,4], _SX}, Y: #(#SX +1)  Matches for instance the sequence "AATCCAT", with X= "AAT" and Y= "CCAT".  _SX denotes the ident of the occurrence of X and allows to refer to this occurrence. So #SX points to the length of the occurrence of X, and the constraint #(#SX +1) asks the length of the occurrence of Y to be 1 character longer than X. It illustrates the fact that one can  - constrain the domain of an attribute (here [2,4] for the size of the occurrence);  - access its value (here, the actual size #SX of the occurrence of X), and use it to put some constraint on another entity (here, the size of the occurrence of Y).</p>
<p>7. <i>Structure constraints : substitution distance</i> X :: \$[0,1], X :: \$[0,1]  Matches for instance the sequence "AATT".  Description of a simple tandem repeat, where copies of an unknown ancestor word may have diverged by at most one mutation. The ancestor X, "AT" in the instance, is not present in the sequence. Such finding would be impossible with a Genlang-like expression such as X, X :: \$1, since the two copies of X have no letter in common and are at distance 2.</p>

8. <i>Pseudoknot</i> $-, X:#[5,8], -, Y:#[5,8], -, -wc X, -, -wc Y, -$ Can recognize pseudoknot of figure 2. In case of more specific search, the model could be more restrictive, with additives indications on sizes of loops (such as $_:#[1,3]$ ).
9. <i>Structure constraints: edit distance</i> $X:#[2,4], _SX, X::\in[0,1], Y:##(SX+1)$ Matches "ACGAGTTAT", with $X="ACG"$ and $Y="TTAT"$ . It includes a tandem copy with at most one error. It illustrates the fact that $SX$ refers only to the first copy of $X$ that could have size differing from the second copy.
10. <i>Optimization constraint : maximal length</i> $_, X:#!#[2,4], X, -$ Introduces an optimization constraint on the length of solutions. On sequence "ATATATATACGCG", it returns only three solutions at positions [1,8], [2,9] and [10,13] with $X="ATAT"$ , $X="TATA"$ and $X="CG"$ . $X="AT"$ or $X="TA"$ are not proposed as solutions because they are not optimal: their occurrences are included in larger occurrences.
11. <i>Optimization constraint : minimal starting position</i> $(_, "ATG":_SATG,@[3,25], (:#3,?~stop)+, stop, _):!@SATG$ with $stop \rightarrow "TAG"$ Introduces an optimization constraint on the starting position of solutions. It returns only two solutions on sequence "ATGATGTATGGAATGTAGCATGCGGTAGGG", at positions [4,18] and [20,28]: "ATG" at position 1 is not in the allowed range of starting positions, occurrence [4,28] contains a stop inside it and occurrences starting at 8 and 13 are not starting after position 18 that corresponds to the end of the leftmost solution.

Table 1: Examples of Logol models

where  $\mathcal{E} = (SV \cup (\{+, -\} \times \Gamma \times SV) \cup (\bigcup N_k \times V^k))$  and  $\mathcal{C}$  stands for a constraint.

Production rules allow to derive words in the usual way, except that the right part is a set whose all elements must be derived (views). Constraints allow to establish the correspondance between words and strings. A constraint  $\mathcal{C}$  may be of two types; either it restricts the set of strings by bounding their size, position and content to a given set of values (string constraint) or it restricts the matching of a string to a word by bounding the distance between them (structure constraint). Furthermore, it is possible to access the complementary of values of strings and use them in constraints.

## 4 Discussion and Conclusion

We have proposed a language for the analysis of genomic structures that builds on SVG background for an enhanced expressivity. The key components of the language are string variables with errors, overlapping and concurrent analysis, and optimization primitives. This strictly extends the possibilities of SVG while preserving the foundation on naturally occurring biological structures. Logol still remains an "open workshop" that might include other features relevant for sequence modelling with the challenge of staying practical and tractable. We conclude by pointing at several important points that have not been adressed in this work.

The world of macromolecules is characterized by the possibility of numerous interactions between them. Particularly, DNA and RNA molecules may hybridize in various ways. From the point of view of modelling, this means that

we have to consider several sequences instead of just one. Searls suggests in [30] to introduce a new cut symbol in grammars that expresses a delimitation between interacting parts in a chain. It virtually cuts the string at this point. Our approach consists in giving both a model with multiple views and multiple sequences and expressing interactions via string variables and string idents. For instance,  $(View_1 \& View_2) \Leftarrow \{Seq_1, Seq_2\} \& \{Seq_3\}$  applies  $View_1$  and  $View_2$  to pairs of sequences  $(Seq_1, Seq_3)$  and  $(Seq_2, Seq_3)$ . It remains to study the semantics to be associated to such multiple analyses with respect to biological needs.

Views and string variables allow to set systems of equations on patterns, in the sense of pattern languages (series of letters and variables). A powerful solver on string constraints would need a relevant equational theory on these strings, just sufficient to take into account practical queries on genomic sequences. Unifying patterns is a much more difficult issue than standard unification on terms because no unique most general unifier exists in the general case. Algorithms exist solving equations on patterns [25] that have a complexity at least DEXP-TIME but provides a sound framework for more specialized algorithms. Parsing of models on genomic sequences offers an ideal context for the development of practical constraint solvers acting in coordination with standard parsing strategies.

In conclusion, the availability of a parser for Logol that works on non-toy sequences remains a priority and a strong necessity for the diffusion of advanced aspects of formal language in the biological community. We have developed a first implementation, STAN, for a restricted subset of Logol expressions [23]. Our approach to limit the complexity of parsing makes profit of the fact that genomes to be analysed are stable sequences that can be pre-processed. We produce for each genome a generalized suffix tree and this index allows then to look at string variables efficiently. We are currently developping a parser for the full syntax of Logol. Models are described by means of logical grammar rules, as it is the case in GENLANG with SVG rules [10]. The standard chart parsing strategy could be enhanced using a suffix array data structure in this context.

## References

- [1] A. V. Aho. *Algorithms for finding patterns in strings*, pages 255–300. MIT Press, Cambridge, MA, USA, 1990.
- [2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.
- [3] D. Angluin. Finding patterns common to a set of strings (extended abstract). In *STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 130–141, New York, NY, USA, 1979. ACM Press.
- [4] D. Betel and C. W. V. Hogue. Kangaroo - a pattern-matching program for biological sequences. *BMC Bioinformatics*, 3:20, 2002.
- [5] B. Billoud, M. Kontic, and A. Viari. Palingol: a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. *Nucleic Acids Res*, 24(8), Apr. 15 1996.

- 
- [6] A. Brazma and D. Gilbert. A pattern language for molecular biology. Technical Report 11, City University, London, 1995.
- [7] V. Brendel and H. Busse. Genome structure described by formal languages. *Nucleic Acids Research*, 12(5):2561–2568, February 1984.
- [8] P. Bucher and A. Bairoch. A generalized profile syntax for biomolecular sequences motifs and its function in automatic sequence interpretation. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. B. Searls, editors, *Second International Conference on Intelligent Systems for Molecular Biology (ISMB-94)*, pages 53–61, Menlo Park, CA, 1994. AAAI Press.
- [9] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Inf.*, 39(1):31–70, 2003.
- [10] S. Dong and D. B. Searls. Gene structure prediction by linguistic methods. *Genomics*, 23(3):540–551, 1994.
- [11] R. Dowell and S. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5, July 03 2004.
- [12] I. Eidhammer, I. Jonassen, S. H. Grindhaug, D. Gilbert, and M. Ratnayake. A constraint based structure description language for biosequences. *Constraints*, 6(2/3):173–200, 2001.
- [13] S. Gräf, D. Strothmann, S. Kurtz, and G. Steger. HyPaLib: a Database of RNAs and RNA Structural Elements defined by Hybrid Patterns. *Nucleic Acids Res.*, 29(1):196–198, 2001.
- [14] L. Grate, M. Hebster, R. Hughey, D. Haussler, I. S. Mian, and H. Noller. RNA modeling using gibbs sampling and stochastic context free grammars. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, volume 235, pages 1501–1531, Menlo Park, CA, August 1994. AAAI/MIT Press.
- [15] C. Helgesen and P. R. Sibbald. PALM - A pattern language for molecular biology. In L. Hunter, D. B. Searls, and J. Shavlik, editors, *In Proceedings of the First International Conference on Intelligent Systems for Molecular Biology (ISMB-93)*, pages 172–180, Menlo Park, CA, 1993. AAAI Press.
- [16] A. K. Joshi, K. Vijay-Shanker, and D. Weir. The convergence of mildly context-sensitive grammars. In S. M. Shieber and T. Wasow, editors, *The Processing of Natural Language Structure*, pages 31–81. MIT Press, Boston, MA, 1991.
- [17] S.-w. Leung, C. Mellish, and D. Robertson. Basic gene grammars and GNA-chartparser for language processing of escherchia coli promoter DNA sequences. *Bioinformatics*, 17(3):226–236, 2001.
- [18] C. Loose, K. Jensen, I. Rigoutsos, and G. Stephanopoulos. A linguistic model for the rational design of antimicrobial peptides. *Nature*, 443:867–869, Oct. 2006.

- [19] I. Makalowska, C.-F. Lin, and W. Makalowski. Overlapping genes in vertebrate genomes. *Computational Biology and Chemistry*, 29(1):1–12, 2005.
- [20] H. Mangalam. tacg - a grep for dna. *BMC Bioinformatics*, 3:8, march 2002.
- [21] J. Mrázek and S. Xie. Pattern locator: a new tool for finding local sequence patterns in genomic dna sequences. *Bioinformatics*, 22(24):3099–3100, 2006.
- [22] F. Nicolas and E. Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *J. Discrete Algorithms*, 3(2-4):390–415, 2005.
- [23] J. Nicolas, P. Durand, G. Ranchy, S. Tempel, and A.-S. Valin. Suffix-tree analyser (stan): looking for nucleotidic and peptidic patterns in chromosomes. *Bioinformatics*, 21(24):4408–4410, 2005.
- [24] P. A. Pevzner and S.-H. Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278. AAAI Press, 2000.
- [25] W. Plandowski. An efficient algorithm for solving word equations. In J. M. Kleinberg, editor, *STOC*, pages 467–476. ACM, 2006.
- [26] A. Rajasekar. Applications in constraint logic programming with strings. In *Principles and Practice of Constraint Programming*, pages 109–122, 1994.
- [27] I. Rice, P. Longden and A. Bleasby. Emboss: The european molecular biology open software suite. *Trends in Genetics*, 16(6):276–277, June 2000.
- [28] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Recent methods for RNA modeling using stochastic context-free grammars. In *Proceedings of the Asilomar Conference on Combinatorial Pattern Matching*, pages 289–306, New York, NY, 1994. Springer-Verlag.
- [29] I. Salvador and J.-M. Benedi. RNA modeling by combining stochastic context-free grammars and n-gram models. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(3):309–315, 2002.
- [30] D. B. Searls. Formal grammars for intermolecular structure. In *First international symposium on intelligence in neural and biological systems. INBS'95*, pages 30–37, Los Alamitos, CA., 1995. IEEE Computer Society Press.
- [31] D. B. Searls. String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming*, 24(1 & 2):73–102, July/August 1995.
- [32] D. B. Searls and S. Dong. A syntactic pattern recognition system for DNA sequences. In C. R. Cantor, H. A. Lim, J. Fickett, and R. J. Robbins, editors, *Proceedings 2nd International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 89–101. World Scientific, 1993.

- [33] D. Strothmann, S. A. Gräf, S. Kurtz, and G. Steger. The syntax and semantics of a language for describing complex patterns in biological sequences. Technical report, Universität Bielefeld, Technische Fakultät, Arbeitsgruppe Praktische Informatik, Aug. 2000.
- [34] Y. Uemura, A. Hasegawa, S. Kobayashi, and T. Yokomori. Grammatically modeling and predicting RNA secondary structures. In *Proceedings of 6th Genome Informatics Workshop*, pages 67–76, Tokyo, 1995. Universal Academy Press.



---

Unité de recherche INRIA Rennes  
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes  
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399