Expressive pattern matching with LOGOL

Application to the modelling of -1 Ribosomal Frameshift events

Catherine Belleannée - Dyliss team, Rennes 1 University
Olivier Sallou - GenOuest plateform, Rennes 1 University
Jacques Nicolas - Dyliss team, Inria
What is Logol?

- A new tool for **pattern matching** on DNA, RNA, and proteins

- **sequences**:
  - attccgtctacc
  - ctttgctacg
  - taggctggcttcggatt
  - tcggcatttgattcpga
  - cggatcgattcttttac

- **matches in the sequences**:
  - attcc**ggtct**acc
  - ctttg**tcacg**
  - tagg**ctggctt**cggatt
  - tcgg**catttg**attcpga
  - cggat**catttt**tac

- **model pattern**: atttgctacg
Why a new tool?

- Towards more **expressive patterns** beyond motifs \( \text{TAT-} [\text{A} \mid \text{T}] \text{-T-} \text{XXX-} \text{AATTCCC} \) towards real biological models

- While remaining **practicable**
  - accept real sequences (e.g. full genomes)
  - in reasonable time
Outline

1. Logol language
   - Foundations
   - Some elements

2. Logol tool
   - Availability
   - Design of a pattern
   - Specifications of the tool

3. An example: modelling « -1 frameshifting sites »

4. Conclusion
1. Logol language
Foundations of the language

• Make the structure of motifs explicit -> Grammatical models
  *Describe «the language of gene»* cf David Searls

• with an accurate level of grammar -> String Variable Grammars (SVG)

<table>
<thead>
<tr>
<th>String Variables</th>
<th>direct copy</th>
<th>reverse complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X\ldots X$</td>
<td>atcgttatgtatgtatga</td>
<td>atcgttatgtagataaccga</td>
</tr>
<tr>
<td>$X\ldots \sim X$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SVG : beyond context-free grammars: « middly context sensitive »

| regular grammars | motif (TAT-[A|T]-T-xxx-AATTCCC) |
|------------------|----------------------------------|
| context-free grammars | + palindrome (stem-loops) |
| SVG | + copy, repeat |

• Previous languages/tools using String Variables
  Patscan[Dsouza&al, 97], Patsearch[Pesole&al,00] limited expressivity
  Genlang[Dong&Searls, 94], Stan[Nicolas&al, 05] or no more maintained
  -> Logol : in the lineage of Genlang
Some elements of the language 1/3

• **A first grammar**: looking for « aaaaa » anywhere in the sequence

```plaintext
mod1() =>* > SEQ1
mod1() => "aaaaa"  act aaaaa tgg aaaaagta
```

• **Inexact matches**: 2 counters => mismatch ($), indel ($$)

```plaintext
-> mod1() => "aaaaa":"{[$0,1]}  act agaaa tgg  cost=1 mismatch
-> mod1() => "aaaaa":"{$$[0,1]}  act aaaaacatgg  distance=1 insert
```

• **String Variables**: looking for 2 copies of a string (x1) separated by a gap (.*

```plaintext
mod1() => x1:{# [5,8]},  .*, x1  act atcaatg gatcaagta
```

• **Morphisms**: to convert a string into another string  *personal morphisms allowed*

"wc": Watson Crick complement,  "-" : reverse string, "wobble" : wobble cplt
"-wc": reverse complement

```plaintext
mod1() => "-wc" "aaccce"  act ttgg ggtt gatcaagta
```
Some elements of the language 2/3

• A constraint approach
Constraints: begin (@), end (@@), content (?), length (#), cost ($), distance ($$), composition(%)  

\[ mod1() \Rightarrow X1: \{ ^{\%} \text{"cg"} : 70 \} \quad \text{X1 must contain at least 70\% of 'c' and 'g'} \]

• Variables can denote instances  \( \text{Instance} = \text{string} + \text{components} \)

\[ \text{Mark an instance} \ (_{\text{VARNAME}}) \text{ and reuse it} \ RSVPAR\text{NAME}, ^{\%}\text{VARNAME}... \]

The second string must exactly match the previous instance  \( \text{actaataaaataatcactacct} \)
\[ mod1() \Rightarrow "aaaaa":\{^[0,1], _{\text{SAVE1}}\}, ^{?}\text{SAVE1} \]

'acgt' must be located at least 50 nt further than 'aaaaa'  
\[ mod1() \Rightarrow "aaaaa":_{\text{SAVE1}}, \ . , ^{\%}\text{acgt}:\{@[\text{SAVE1+50}, \text{SAVE1+100}]\} \]

Looking for 3 strings, successively deriving from each other  \( \text{actaaaaaaataataataaca} \)
\[ mod1() \Rightarrow X1:\{^[5,8], _{S1}\}, ^{?}\text{S1}:\{_{S2}\}:\{^[1,1]\}, ^{?}\text{S2}:\{^[1,1]\} \]

Looking for a \textbf{stem-loop}, with sizes of: stem in [5,11], loop in [1,9],  
stem strands \text{ linked by Watson-Crick pairing, 2 mismatch + 1 indel allowed in the stem}  
\[ mod1() \Rightarrow \text{STEM5}:\{^[5,11], _{S5}\}, ^{\%}\text{"wc" }\{^[0,2], ^{\%}\text{S5}:\{^[0,1]\} \]
Some elements of the language 3/3

- **Repeats**: looking for "acgt" repeated between 0 and 5 times. The instances may be separated by a Spacer between 0 to 2 nt

  \[ \text{mod1}() \Rightarrow \text{repeat}("acgt", [0,2]) + [0,5] \quad \text{actacgtggacgtcagtcctac}\]

- **Negative contain constraints (!)**: looking for a string with length between 2 and 5 which is not "ag"

  \[ \text{mod1}() \Rightarrow \neg \text{"ag"}:{#}[2,5]\]

- **Put constraints on several strings**

  ➢ **VIEW**: constraints on consecutive segments
  
  The total size of X1::X2::X3 must be between 8 and 20
  
  \( (X1:{#}[1,10], \ X2:{#}[1,10], \ X3:{#}[1,10]) : \ {#}[8,20] \)

  ➢ **Control panel**: constraints on non consecutive segments

- **Superposition of complementary models**: Multiple model 'points of view'

  \[ \text{mod1}(\text{VAR1}).\text{mod2}(\text{VAR1},\text{VAR2}).\text{mod3}() \Rightarrow \text{SEQ1} \]
2. Logol tool
Availability

- **On the web**: Logol can be used on the GenOuest web site (with restrictions) [http://webapps.genouest.org/LogolDesigner/](http://webapps.genouest.org/LogolDesigner/)

- **Via Linux command-line on GenOuest plateform** with a GenOuest account

- **Download on your own computer** NEW! NEW! NEW!
  Logol software is **free** and **open source**, under CeCILL license
  It includes a Linux command line tool and a graphical designer

Logol is a fully maintained tool (development manager: Olivier SALLOU)

**Main logol page:**

Design of a pattern

- **Grammatical model**
  - Text file `mymodel.lgg`

- **Graphical model**
  - With a graphical designer `mymodel.lgd`

```plaintext
mod4() ==>("aaa")|("ccc")|("uuu")|("ggg")
mod2() ==>mod4(),("aaa")|("uuu")
! "g":[#1,1]
```

http://webapps.genouest.org/LogolDesigner/
Specifications of the tool

• **Input**: a Logol model (graphical or grammatical model)
  a Fasta sequence

• **Runs on** a computer or a grid (Linux)
  Configurable to support multi-core architectures and to use multiple nodes to parallelize treatments when possible.
  Sequences may be split for more parallelization

• **Output**: a compressed XML file, contains all matches of the model
  With the details of each match (position of each word, size, number of errors compared to model…)
  Possibility to convert it to Fasta (sequence only) or GFF output

---

**Main pipeline**
- a **Java program** transforms the model file into a Prolog program
- the **Prolog program** parses the sequence (to find the instances of the model)
  it uses
  - a **Prolog library** (with predicates to operate morphisms, % calculus…)
  - a **suffix array** indexation (with “Vmatch” or home product “Cassiopee”)
3. An example: modelling « -1 frameshifting sites »
Programmed -1 ribosomal frameshifting

A translational recoding strategy

- one mRNA may produce two distinct proteins

Ribosome may switches from the translation of the standard ORF (in the 0-frame) to an overlapping ORF (in the -1 frame)

There, the ribosome may slip of 1 nucleotide to the left

slippery site

standard protein (from the 0-frame)

alternative protein

- begining : built from the 0 frame
- end : built from the -1 frame
Typical frameshifting site

Located in messenger RNA

- slippery motif
  heptamer X XXY YYZ
- spacer
- stimulatory structure
  mainly H-type pseudo-knot
  (two overlapping stem-loops)

5' AUG — — — — — — X XXY YYZ NNNNNNN

start    slippery motif    spacer    pseudo-knot

3'
Modelling **slippery motif**

Heptamer XXXYYYYZ with  
X: any nucleotide,   Y: ‘a’ or ‘u’  
Z: not a ‘g’

**Graphical model**

**Grammatical model**

\[ \text{mod4}() \Rightarrow (\text{"aaa"} | \text{"ccc"} | \text{"uuu"} | \text{"ggg"}) \]

\[ \text{mod2}() \Rightarrow \text{mod4()},(\text{"aaa"} | \text{"uuu"}),! \text{"g"}:\{\#[1,1]\} \]
Modelling **spacer**

**Spacer**: a string that contains from 1 to 10 nucleotides

<table>
<thead>
<tr>
<th>Grammatical model</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mod5() =&gt; SPACER1:#{[1,10]}</code></td>
</tr>
<tr>
<td><strong>String variable with a size constraint</strong></td>
</tr>
</tbody>
</table>
Modelling **pseudo-knot 1/3**  
A first model

Two overlapping stem-loops,  
with - sizes of: stem1 in [4,16], loop1 in [1,5],  
**stem2** in [3, 8], loop2 in [0,4], loop3 in [4,40]  
- stem strands linked by Watson-crick pairing,  
- 4 mismatches allowed in stem1, 2 allowed in stem2

**Graphical model**

**Grammatical model**

**STEM15**: {#[4,16], _S15}, .*:{#[1,5]}, **STEM25**: {#[3,8], _S25},  
.*:{#[0,4]}, -"wc" ?S15 :{$[0,4]}, .*:{#[4,40]}, -"wc" ?S25 :{$[0,2]}

Logol operators: String variable, size constraint, cost constraint, morphism wc

**BUT: test on data=> too much false positives. The model is not selective enough**
Modelling **pseudo-knot 2/3**

A more realistic model at a glance
Modeling pseudo-knot 3/3

Logol components of a more realistic model

• Precise error and constraint handling
  prohibit mismatches at stem extremities:
  a stem is divided in 3 parts: <first nt, main part of the stem, last nt>
  and accept wobble pairing (g-u) (wobble morphism def)

• Check global statistical constraints
  Allows describing stem stability
  A majority of ‘gc’ pairing is required
  -> counting ‘gc’ in a stand (at least 50% of ‘g’ and ‘c’ on <A5,STEM5’,Z5>)
    use contain constraint on adjacent strings (i.e. on a view)
    mod3() ==>
    (A5: [#1,1], _SA5),  STEM15: [#2,14], _S15),
    Z5: [#1,1], _SZ5) : {"gc": 50}

  -> counting ‘c’ in both stands (at least 25% of ‘c’ on <A5,STEM5’,Z5> +
    <Z3,STEM3’,A3>) because ‘g’ may be involved in a (weak) wobble pairing
    use contain constraint on non adjacent strings (stem5’ + stem3’)

LOGOL  C. Belleannée, O. Sallou, J. Nicolas  Jobim  3 juillet 2012
Modelling **frame alignments 1/3** Specifications

- 2 alternatives translations

```
- - X
```

```
 XXY
```

```
 YYZ
```

All translation in 0 frame

Shifting in -1 frame

⇒ Superposition of 3 mandatory models

- SlipperySite model: a start, then a slippery site in the -1 frame

- ORF0 model: from the same start, a stop is needed in the 0 frame

- ORFminus model: from the same start, a stop is needed in the -1 frame

Between start and stop, intermediate codons should not contain a stop codon
Modelling **frame alignments**  2/3  *In Logol*

- Superposition of the 3 models
  -> using **multiple models** in the Logol main ‘rule’

- Alignment on the same start
  -> **mark variable** `START` and reuse its position (`@START`)  
  -> transfer **parameters** between models (the variable `START`)
Modelling frame alignments 3/3  

- A nonstop codon: a string of size 3nt, which is not a stop
- stop is defined as a model by an alternative (uga | uag | uaa)
- then use a view with size constraint (size = 3) and negative contain constraint (≠ stop)
- => Nonstop model

- Set of successive nonstop codons
- => consecutive repeat (from 5 to 300 times) of the nonstop model
Test

• Test on 30 positive sequences
  30 « validated -1 frameshift » sequences
  From the database Recode2  http://recode.ucc.ie/
  Size: from 5 000 nt to 30 000 nt
    ➢ Hits: about 3 sites and 100 hits per sequence, including the good one
      with an additional post-filter ordering the hits according to stem quality, the
      official hit is ranked 1st for 20 sequences
    ➢ Time (on Intel X5550, 144Go RAM): 1mn 30s for the biggest sequence
      immediate response on dedicated KnotInFrame site

• Test on Bacillus Subtilis complete genome
  Reference: Str 168 NC_000964.3
  Size : 4 215 606 nt
    ➢ Hits : about 7 000 hits  (with a looser model)
    ➢ Time (on Intel X5550, 144Go RAM): 2 hours
      no response on KnotInFrame site
4. Conclusion
Logol:

- A general purpose modelling language, for every type of sequences
  - with quite important expressivity
  - practicable and maintained

- A new birth for String Variable Languages

- An ongoing project:
  - still in progress
  - your returns are expected to go on improving the language and the tool

A single address:
http://logol.genouest.org/web/app.php/logol

Test it!
5. Supplementary data
Job sequence

Grammar interpreter
(convert grammar to prolog)

Job Manager
(split sequence if multi core)

Sequence Parser
(Execute prolog program)

Vmatch
Or
Cassiopee

Logol grammar file

1 sequence per job
Parallel jobs if on cluster or sequential if local

Fasta sequences

Result file

Merged results
Grammar analysis

- The analyzer is a Java program used to parse a Logol grammar file or model (graphical) and transform it in a Prolog program.
- The Prolog program uses a Prolog library that contains predicates to match expressions (fixed content, morphisms,…).
- Logic programming: generated program tries to match all expressions up to a final match, else it goes backward to test the next possibility.
- The analyzer extracts a maximum of information for a variable to limit the range of search (max size, base content defined later on in the grammar,…).
Execution flow

- Left to right file reading but accepts variable use before being determined (\_R1,"acgt":\{?R1\})
- Parse the sequence file by position and try to match an expression (content with error, constraints on a group of variable,...). When all expressions are matched, record it as a result.
- A match is recorded as an object. This object records all the information of the match all along the parsing: Object = [pattern1, pattern2,...]
  - patternX is itself an object holding match information (position, errors,...) but also sub patterns if applicable (models, repeats,...).
- In case of gap (not a position+1):
  - Calls Vmatch (suffix array) or cassiopee when content is known (possibly with errors)
    - Vmatch is a performant suffix array search tool with distance/error support
    - Cassiopee is a basic Ruby library with distance/error/ambiguity support, but not optimized for large sequences.
      - Continue by position with all possibilities when looking for content not defined at this time
- Optional final step: filtering to keep optimal results only.
Zoom on a specificity of Logol variables

\[ X:: [0,1], \quad X:: [0,1] \quad \text{tandem repeat} \]

-> It accepts ATTA with X= “AA” (or X= “TT”). This is impossible with Genlang (X, X:: $1$). So, Logol allows to make the distinction between entity (abstract) and instance (concrete)

Control pannel: To put constraints on non adjacent strings

Counting ‘c’ in both stands (at least 25% of ‘c’ on \(<A5,STEM5’,Z5> + <Z3,STEM3’,A3>\)) because ‘g’ may be involved in a (weak) wobble pairing

use contain constraint on non adjacent strings (stem5’ + stem3’)

controls:

```plaintext
```
# Définition du morphisme wobble

def: {
    morphism(wcw,a,t)
    morphism(wcw,t,a)
    morphism(wcw,c,g)
    morphism(wcw,g,c)
    morphism(wcw,g,t)
    morphism(wcw,t,g)
}

# Définition des contrôles

controls: {
}

# Définition du modèle Logol

mod4() => (("aaa") |("ccc") |("ttt") |("ggg"))
mod2() => mod4(), (("aaa") |("ttt")) ! "g":{#1[1,1]}
mod3() => (A5:{#1[1,1],_SA5}, S15:{#2[1,14],_STEM15}, Z5:{#1[1,1],_SZ5}):{%"gc":50}, L1:{#1[1,5]},
    (X5:{#1[1,1],_SX5}, S25:{#1[1,6],_STEM25}, Y5:{#1[1,1],_SY5}):{%"gc":50}, L11:{#0[4,4]},
    -"wcw" Z3:{_SZ5,_SZ3}, -"wc" _STEM15:{_STEM13}:{p$[0,34]}, -"wcw" A3:{_SA5,_SA3}, L2:{#4[4,40]},
    -"wcw" Y3:{_SY5,_SY3}, -"wc" _STEM25:{_STEM23}:{p$[0,34]}, -"wcw" X3:{_SX5,_SX3}
mod1() => CC1:{#2[2]}, mod2(), SPACER2:{#1[1,10]}, mod3()
mod1() => *=> SEQ1