BioSyntax: A Genomic Compiler

IntelliJ Custom Language Plugin

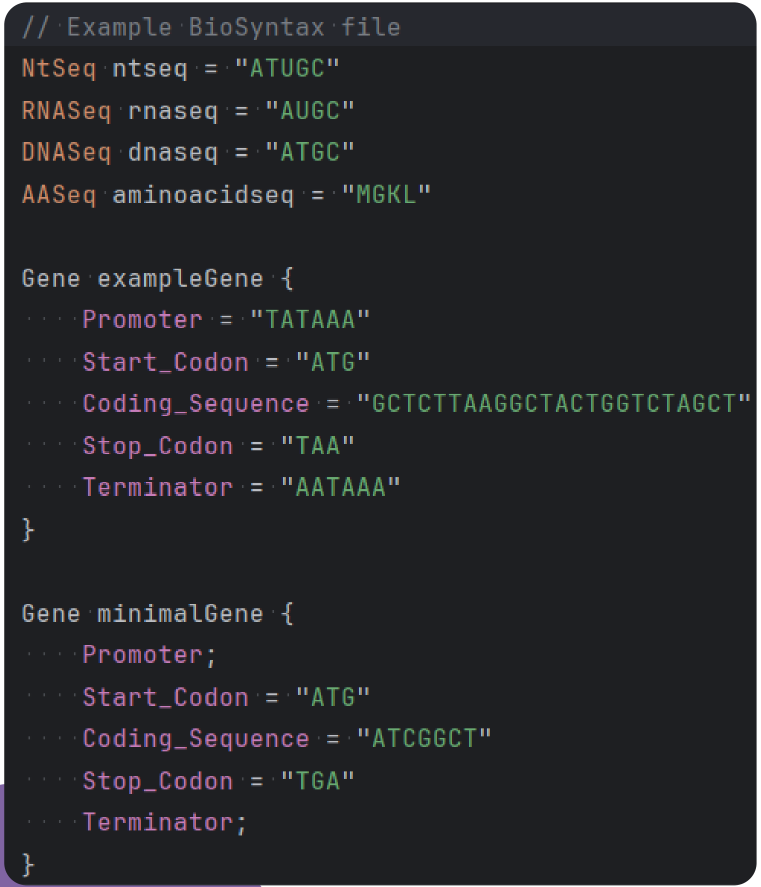
Jiwoo Jung

Introduction

[1]

IntelliJ Custom Language plugin:

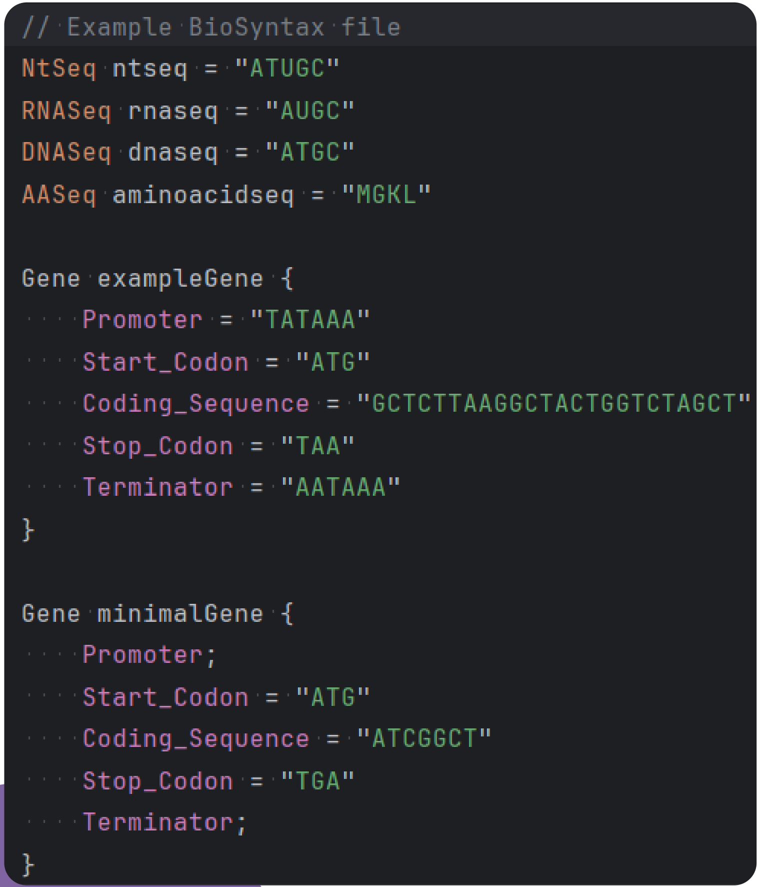
- **BNF Grammar (.bnf file):** Defines the structure and syntax of the language using Backus-Naur Form notation
- **Parser**: Generated from the BNF grammar using Grammar-Kit
- **Lexer:** Created using JFlex, the lexer breaks down the input text into tokens
- **Gradle Grammar-Kit plugin**: Automates the generation of lexer and parser code from the BNF and JFlex definitions. Constructs an Abstract Syntax Tree (AST) from the tokens produced by the lexer.



```
// Example BioSyntax file
NtSeq ntseq = "ATUGC"
RNASeq rnaseq = "AUGC"
DNASeq dnaseq = "ATGC"
AASeq aminoacidseq = "MGKL"

Gene exampleGene {
    Promoter = "TATAAA"
    Start_Codon = "ATG"
    Coding_Sequence = "GCTCTTAAGGCTACTGGTCTAGCT"
    Stop_Codon = "TAA"
    Terminator = "AATAAA"
}

Gene minimalGene {
    Promoter;
    Start_Codon = "ATG"
    Coding_Sequence = "ATCGGCT"
    Stop_Codon = "TGA"
    Terminator;
}
```

BioSyntax

- A custom language plugin for IntelliJ IDEA aimed at simplifying genetic modification design
- Provides a user-friendly interface for biologists and programmers to work with DNA sequences
- Aims to bridge the gap between computational design and biological implementation



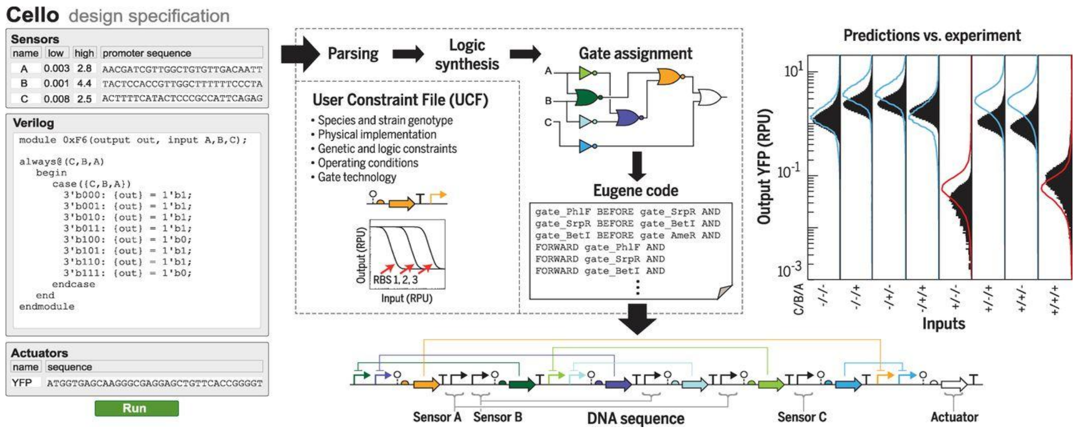
```
// Example BioSyntax file
NtSeq ntseq = "ATUGC"
RNASeq rnaseq = "AUGC"
DNASeq dnaseq = "ATGC"
AASeq aminoacidseq = "MGKL"

Gene exampleGene {
    Promoter = "TATAAA"
    Start_Codon = "ATG"
    Coding_Sequence = "GCTCTTAAGGCTACTGGTCTAGCT"
    Stop_Codon = "TAA"
    Terminator = "AATAAA"
}

Gene minimalGene {
    Promoter;
    Start_Codon = "ATG"
    Coding_Sequence = "ATCGGCT"
    Stop_Codon = "TGA"
    Terminator;
}
```

BioSyntax

Features:

- Syntax highlighting for DNA sequences and genetic elements
- Custom error detection for invalid nucleotide sequences
- Auto-completion for common genetic motifs and structures
- Integration with existing bioinformatics tools

Motivation



Sensors

name	low	high	promoter sequence
A	0.003	2.8	AACGATCGTTGGCTGTGTTGACAATT
B	0.001	4.4	TACTCCACCGTTGGCTTTTTTCCCTA
C	0.008	2.5	ACTTTTCATACTCCCGCCATTCAGAG

Verilog

```
module 0xF6(output out, input A,B,C);

always@(C,B,A)
   begin
      case({C,B,A})
        3'b000: {out} = 1'b1;
        3'b001: {out} = 1'b1;
        3'b010: {out} = 1'b1;
        3'b011: {out} = 1'b1;
        3'b100: {out} = 1'b0;
        3'b101: {out} = 1'b1;
        3'b110: {out} = 1'b1;
        3'b111: {out} = 1'b0;
      endcase
   end
endmodule
```

Actuators

name	sequence
YFP	ATGGTGAGCAAGGGCGAGGAGCTGTTCACCGGGGT

Eugene code

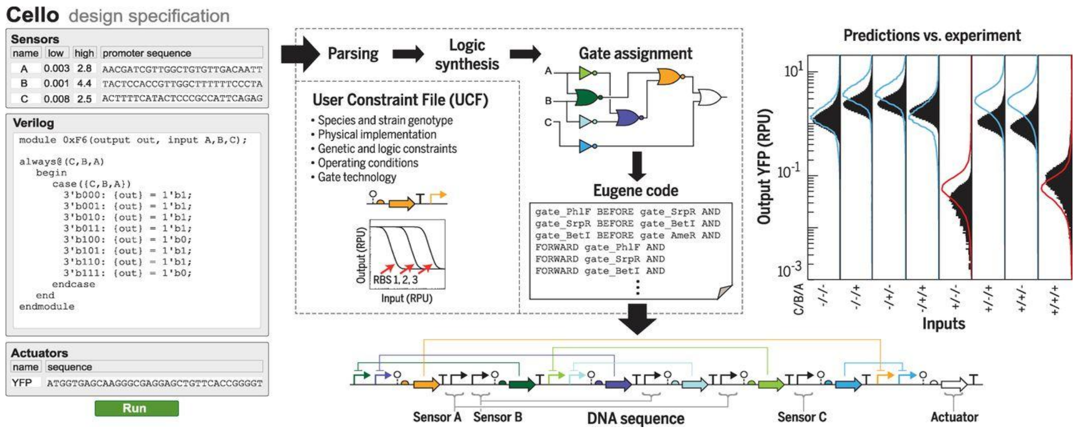
```
gate_PhlF BEFORE gate_SrpR AND
gate_SrpR BEFORE gate_BetI AND
gate_BetI BEFORE gate AmeR AND
FORWARD gate_PhlF AND
FORWARD gate_SrpR AND
FORWARD gate_BetI AND
                ⋮
```

[2]

Genetic Circuit Design Automation:

- Introduces *Cello*, a design environment that automates the programming of genetic circuits using a hardware description language (Verilog)
- Transforms high-level circuit specifications into DNA sequences for implementation in living cells

Motivation



Sensors

name	low	high	promoter sequence
A	0.003	2.8	AACGATCGTTGGCTGTGTTGACAATT
B	0.001	4.4	TACTCCACCGTTGGCTTTTTTCCCTA
C	0.008	2.5	ACTTTTCATACTCCCGCCATTCAGAG

Verilog

```
module 0xF6(output out, input A,B,C);

always@(C,B,A)
   begin
      case({C,B,A})
        3'b000: {out} = 1'b1;
        3'b001: {out} = 1'b1;
        3'b010: {out} = 1'b1;
        3'b011: {out} = 1'b1;
        3'b100: {out} = 1'b0;
        3'b101: {out} = 1'b1;
        3'b110: {out} = 1'b1;
        3'b111: {out} = 1'b0;
      endcase
   end
endmodule
```

Actuators

name	sequence
YFP	ATGGTGAGCAAGGGCGAGGAGCTGTTCACCGGGGT

Eugene code

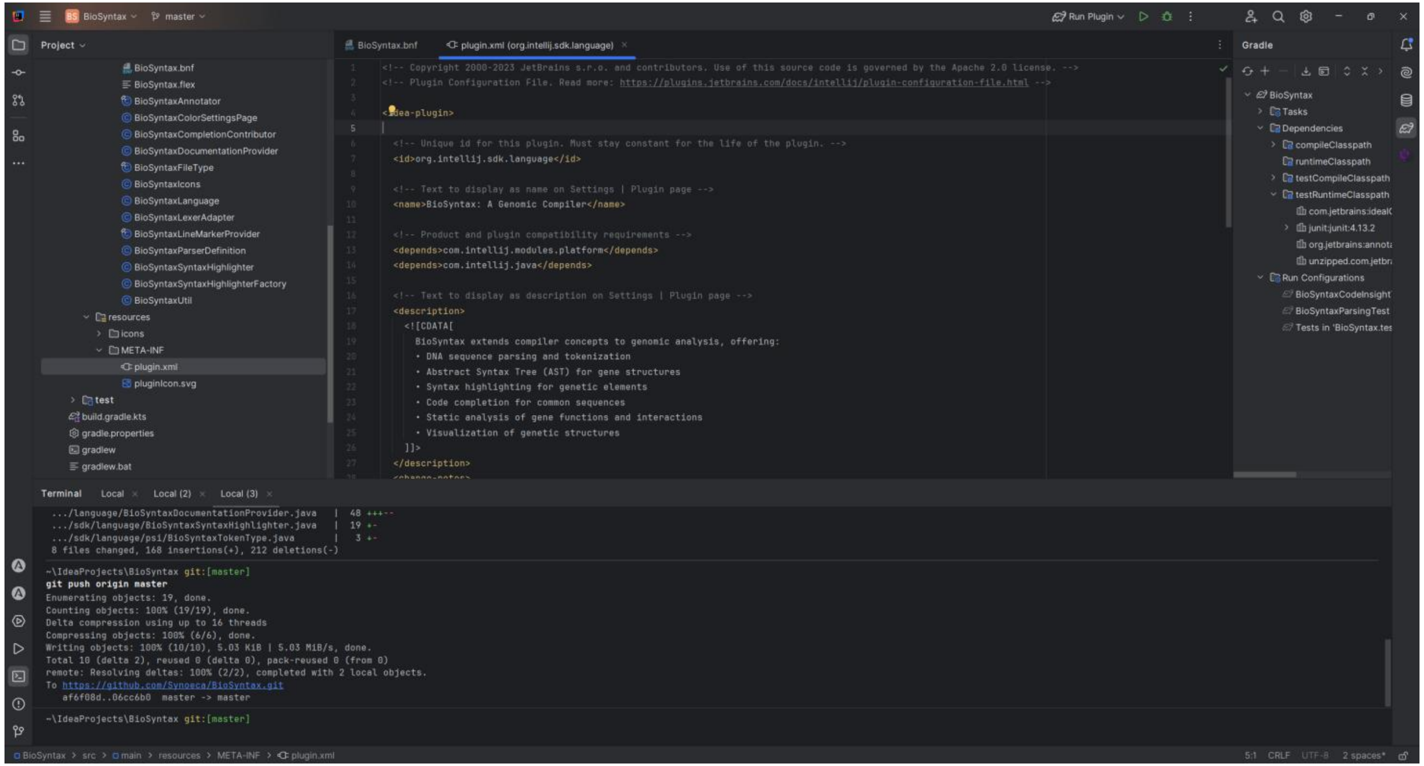
```
gate_PhlF BEFORE gate_SrpR AND
gate_SrpR BEFORE gate_BetI AND
gate_BetI BEFORE gate AmeR AND
FORWARD gate_PhlF AND
FORWARD gate_SrpR AND
FORWARD gate_BetI AND
                  ⋮
```

[2]

Relation to BioSyntax

- BioSyntax aims to provide a custom language for genetic circuit design similar to Cello but focuses on nucleotide-level programming

Demo



```xml
<!-- Copyright 2000-2023 JetBrains s.r.o. and contributors. Use of this source code is governed by the Apache 2.0 license. -->
<!-- Plugin Configuration File. Read more: https://plugins.jetbrains.com/docs/intellij/plugin-configuration-file.html -->

<idea-plugin>

  <!-- Unique id for this plugin. Must stay constant for the life of the plugin. -->
  <id>org.intellij.sdk.language</id>

  <!-- Text to display as name on Settings | Plugin page -->
  <name>BioSyntax: A Genomic Compiler</name>

  <!-- Product and plugin compatibility requirements -->
  <depends>com.intellij.modules.platform</depends>
  <depends>com.intellij.java</depends>

  <!-- Text to display as description on Settings | Plugin page -->
  <description>
    <![CDATA[
      BioSyntax extends compiler concepts to genomic analysis, offering:
      • DNA sequence parsing and tokenization
      • Abstract Syntax Tree (AST) for gene structures
      • Syntax highlighting for genetic elements
      • Code completion for common sequences
      • Static analysis of gene functions and interactions
      • Visualization of genetic structures
    ]]>
  </description>
```

```
 .../language/BioSyntaxDocumentationProvider.java   |  48 +++--
 .../sdk/language/BioSyntaxSyntaxHighlighter.java   |  19 +-
 .../sdk/language/psi/BioSyntaxTokenType.java       |   3 +-
 8 files changed, 168 insertions(+), 212 deletions(-)

~\IdeaProjects\BioSyntax git:[master]
git push origin master
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 16 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (10/10), 5.03 KiB | 5.03 MiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Synoeca/BioSyntax.git
   af6f08d..06cc6b0  master -> master

~\IdeaProjects\BioSyntax git:[master]
```

Advantages

[2]

Based on Biological Principle

Disadvantages

[3]

Non-Deterministic Nature of Biological Systems

Arginine-vasopression-like (AVPL)

>Zatr AVPL XP_063904710.1

~~MYVKSNIQMQLKSIKAQTLLTKISSTKTTT~~MSKLATLIILLALSESIVSGCLITNCPRGGKRSKLALSENTI
KSCLNCGPGQTGQCFGPNICCGPFGCLLGTPETLRCQRDGFFHEREPCIAGTSPCRKNTGRCAFDGI
CCSQDSCHSDKACASEEKSRSFSEVPLDLYNLINYQAELVNDK

Disadvantages

[3]

Non-Deterministic Nature of Biological Systems

ILamide

>Zatr ILamide KAJ3666087.1

MNSELRAGAQAYIPLDQITPPKSIIKVPCSQSTQSSRYSRKTVRANNGTGDTPRPILGTKAPFPRAILG
RKEYAICENKENCTYPTKEYRSMNRNVHEVKINDEKSTPVKANTCSL

Conclusion

[5]

- Genome analysis tool
- Future implementations for integration of bioinformatics tools

References

[1] *Custom language support tutorial: Intellij platform plugin SDK*. IntelliJ Platform Plugin SDK Help. (n.d.). https://plugins.jetbrains.com/docs/intellij/custom-language-support-tutorial.html

[2] Nielsen, A. A., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., & Voigt, C. A. (2016). Genetic Circuit Design Automation. *Science*, *352*(6281). https://doi.org/10.1126/science.aac7341

[3] Teufel, F., Almagro Armenteros, J. J., Johansen, A. R., Gíslason, M. H., Pihl, S. I., Tsirigos, K. D., Winther, O., Brunak, S., von Heijne, G., & Nielsen, H. (2022). SIGNALP 6.0 predicts all five types of signal peptides using protein language models. *Nature Biotechnology*, *40*(7), 1023–1025. https://doi.org/10.1038/s41587-021-01156-3