Eugene 2.0: A Domain-specific Language to Specify Constraint Synthetic Biological Devices

E. Oberortner, H. Huang, S. Bhatia, D. Densmore
Department of Electrical and Computer Engineering, Boston University
ernstl@bu.edu, winstar127@gmail.com, swapnilb@bu.edu, dougd@bu.edu

SynBERC Synthetic Biology Engineering Research Center

The Eugene language aims to support Synthetic Biologists to automate the specification of complex devices [1]. To facilitate the automation of design processes in the Synthetic Biology domain, Eugene offers a rich set of facilities to design **Devices**, **Parts** and their **Properties**, as well as to define **Rules** to constrain the composition of Parts into Devices.

Eugene 2.0 offers, in comparison to its predecessor, a more mature set of Rules that can be defined on Devices, Parts, instances of Parts, and Properties. A new feature of Eugene 2.0 is its **SBOL compliance**, making it possible to exchange newly specified Devices with other SBOL compliant Synthetic Biological tools.

Eugene 2.0 – Definitions

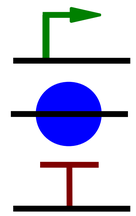
Properties

```
Property ID(txt);
Property Sequence(txt);
Property Length(num);
```

e.g. BBa_J123456
e.g. A T C G A T A G C
e.g. 10

Parts

```
Part Promoter(ID, Sequence, Length);
Part RBS(ID, Sequence);
Part Terminator(ID, Sequence, length);
```

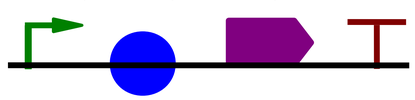


Part Instances

```
Promoter pAmtR("pAmtR","cgacgtacgg...", N);
RBS rbs1("rbs1","attacttagatc...");
Terminator term1("term1","...atctat");
```

Devices

```
Device D1(pAmtR,rbs1,...,term1);
Device AbstractDevice
              (Promoter,  RBS,   CDS,  Terminator);
```



References:

[1] Bilitchenko L et al. "*Eugene – A Domain Specific Language for Specifying and Constraining Synthetic Biological Parts, Devices, and Systems*". PLoS ONE 6(4), 2011

[2] Collins JJ et al. "*Construction of a genetic toggle switch in Escherichia coli*". *Nature* 403 (6767): 339–342, 2000

Rule	Examples

CONTAINS	`MyDevice CONTAINS Promoter`
	`MyDevice CONTAINS RBS`
	`MyDevice CONTAINS Terminator`

AFTER	`ON MyDevice: RBS AFTER Promoter` `ON MyDevice: Terminator AFTER CDS`
BEFORE	`ON MyDevice: RBS BEFORE Promoter` `ON MyDevice: Terminator BEFORE CDS`

NEXTTO	`ON MyDevice: Promoter NEXTTO RBS`
	`ON MyDevice: RBS NEXTTO CDS`
	`ON MyDevice: CDS NEXTTO Terminator`

STARTSWITH	`MyDevice STARTSWITH Promoter`

WITH	`ON MyDevice: Promoter WITH Terminator`
	`ON MyDevice: CDS WITH RBS`
	`ON MyDevice: Promoter WITH CDS`

ENDSWITH	`MyDevice ENDSWITH Terminator`
THEN	`ON MyDevice: Promoter THEN Terminator` `ON MyDevice: Promoter THEN CDS`
MORETHAN	`ON MyDevice: NOT Promoter MORETHAN 1` `ON MyDevice: Terminator MORETHAN 0`
EQUALS	`ON MyDevice:` `  NOT Promoter EQUALS Terminator`
MATCH	`ON MyDevice:` `  NOT CDS MATCH RBS`

<, <=, >=, >	`ON MyDevice:`
	`  CDS.sequence >= RBS.sequence`
	`ON MyDevice: Promoter < CDS`

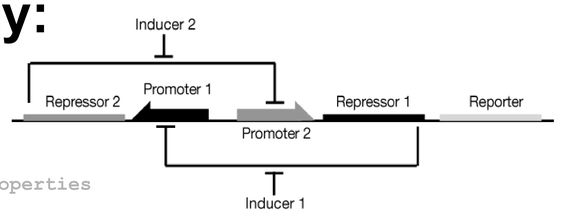
Composition of Rules: AND, OR, XOR

Device-specific Rules: ON <device>:

Negation of Rules: NOT

Case Study:

Toggle Switch [2]




```
// Specification of Properties
Property name(txt);
Property sequence(txt);
Property represses(txt);
Property repressedBy(txt);

// Specification of Parts
Part Promoter(name, repressedBy);
Part Repressor(name, represses);
Part Reporter(name);

// Specification of Part Instances
Repressor repressor2("Repressor2", "Promoter2");
Promoter promoter1("Promoter1", "Repressor1");
Promoter promoter2("Promoter2", "Repressor2");
Repressor repressor1("Repressor1", "Promoter1");
Reporter GFP("GFP");

// Specification of Rules
Rule R1(STARTSWITH repressor2);
Rule R2(repressor1.represses EQUALS promoter1.name);
Rule R3(repressor2.represses EQUALS promoter2.name);
Rule R4(promoter1.repressedBy EQUALS repressor1.name);
Rule R5(promoter2.repressedBy EQUALS repressor2.name);
Rule R6(promoter1 AFTER repressor2);
Rule R7(promoter2 AFTER promoter1);
Rule R8(repressor1 AFTER promoter2);
Rule R9(Reporter AFTER promoter2);
Rule R10(ENDSWITH GFP);
Rule validToggleSwitch (R1 AND ((R2 AND R3) OR (R4 AND R5)) AND
                        R6 AND R7 AND R8 AND R9 AND R10);

// Specification of a ToggleSwitch Device
Device EugeneToggleSwitch(
     repressor2, promoter1, promoter2, repressor1, GFP);

// Check if the EugeneToggleSwitch Device is valid
if(ON EugeneToggleSwitch: validToggleSwitch) {
    print("Device EugeneToggleSwitch is valid!");
}
```

Work in Progress

- Collections and Namespaces
- Loops and Conditional Statements
- Function Prototyping
- Observing the Dynamic Behavior of Devices

www.synberc.org