Logic Analysis and Verification of n-input Genetic Logic Circuits

Hasan Baig and Jan Madsen
Department of Applied Mathematics and Computer Science, Technical University of Denmark

Abstract — **Nature is using genetic logic circuits to regulate the fundamental processes of life. These genetic logic circuits are triggered by a combination of external signals, such as chemicals, proteins, light and temperature, to emit signals to control other gene expressions or metabolic pathways accordingly. As compared to electronic circuits, genetic circuits exhibit stochastic behavior and do not always behave as intended. Therefore, there is a growing interest in being able to analyze and verify the logical behavior of a genetic circuit model, prior to its physical implementation in a laboratory. In this paper, we present an approach to analyze and verify the Boolean logic of a genetic circuit from the data obtained through stochastic analog circuit simulations. The usefulness of this analysis is demonstrated through different case studies illustrating how our approach can be used to verify the expected behavior of an n-input genetic logic circuit.**

I. Introduction and Motivation

Biologists and engineers are working together on synthetic biology [1] to design new and useful biological systems. The ability to re-engineer living cells has created completely new ways of manufacturing biological systems and materials. The applications of synthetic biology include tumor destruction [2], bio-fuels [3], consuming toxic wastes [4], etc.

Synthetic genetic circuits – an application of synthetic biology, are composed of a group of genetic components of DNA (promoter, terminator, etc), which interact with the external signals to control the behavior of a living cell. Genetic circuits produce output proteins, based on the presence of input proteins. Figure 1(a) shows a genetic circuit [14], which behaves as a 2-input electronic AND gate, shown in Figure 1(b).

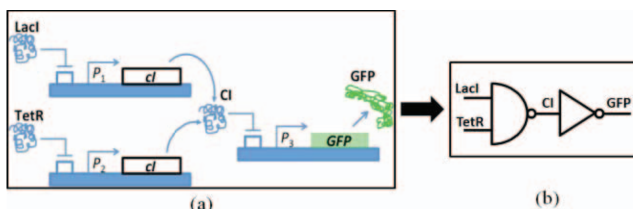


Figure 1. Genetic AND gate circuit. (a) Genetic implementation [14]. (b) Schematic symbol.

In Figure 1(a), P_1 and P_2 are *promoters*, which are the regions of DNA that initiates the process of transcription (or production) of a particular gene. In this example, when two types of proteins, LacI and TetR, are present in significant amount in the re-engineered cell, they inhibit promoters P_1 and P_2 to produce the output gene CI. When the concentration of CI falls below a certain level, promoter P_3 is activated and produces the output protein i.e. a green fluorescent protein (GFP).

State-of-the-art is to design such circuits directly in the laboratory, through trial and error, which is a time consuming and costly process, as thousands of circuits may have to be tested to find a few that works. To overcome this, researchers are currently working on developing *genetic design automation* (GDA) tools [5], to automate the design and test process of genetic circuits – a process like electronic design automation (EDA) where new circuits are simulated before they are fabricated on-chip. The field of genetic circuit design is still immature and only small circuits, containing limited number of genes, can be constructed in the laboratory.

As the number of molecules involved in the chemical reactions inside a cell is small, standard ODE cannot be used to model and solve these reactions [6]. Instead, a stochastic simulation algorithm (SSA) [7] must be used. SSA efficiently handles the reactions occurring randomly for small and discrete number of species.

In this paper, we are interested in validating the logic function of a given genetic logic circuit based on the stochastic simulation traces obtained by applying all different input combinations. The simulation traces can be obtained by any of the many GDA simulation tools that support stochastic simulations. In this work, we have chosen to use D-VASim [8] that is developed for the simulation and analysis of genetic logic circuit models represented in the Systems Biology Markup Language (SBML) [9]. We present a logic analysis and validation algorithm which extracts the logic behavior from the simulations and provide a fitness value that can be used to infer how likely it is that the circuit will actually work after implementation in the laboratory.

The presented algorithm is scalable and able to analyze n-input genetic logic circuits. The logic analysis of genetic circuits is useful in two ways – first, it allows the user to verify complex genetic logic circuits, build by cascading several genetic logic gates; secondly, it helps in extracting the Boolean logic of a circuit even when the user does not have any prior knowledge about its expected behaviour.

II. Methodology

Threshold value and *propagation delay* of I/O species are two important parameters required to obtain a correct Boolean expression of a circuit. The threshold value defines a significant amount of concentration, which categorizes the analog concentrations into digital logics 0 and 1. Propagation delay specifies the time required to reflect the changes in input species concentrations on the concentration of output species. During the experimentation, if the input species concentrations are applied below their threshold levels and each of the input combination is changed before the propagation delay has elapsed, then the circuit never produces a correct output for some of the input combinations. D-VASim supports the capability of analyzing the threshold value and propagation delays [10]. In this work, we used this functionality to obtain a threshold value and a propagation delay of a circuit. We used these results to perform experiments on the genetic circuit models and log all experimental simulation data, which were then given to the proposed algorithm to extract the logical behavior of a circuit.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

Algorithm 1 shows the pseudo code of the *main* procedure of the logic analysis and verification. Some initial parameters (N, SD_{An}, Th_{VAL}, FOV_{UD}, I_S, *and* O_S) are required to execute the algorithm; where N corresponds to the total number of input species, SD_{An} refers to the simulation data of all I/O species, Th_{VAL} denotes the threshold value of I/O species, $\mathrm{FOV_{UD}}$ is the user-defined percentage of acceptable variation in the output data (described later), and I_S and O_S specify the names of input and output species, respectively. By giving users an ability to select the input and output species, they can perform Boolean logic analysis on the entire circuit as well as on the intermediate circuit components.

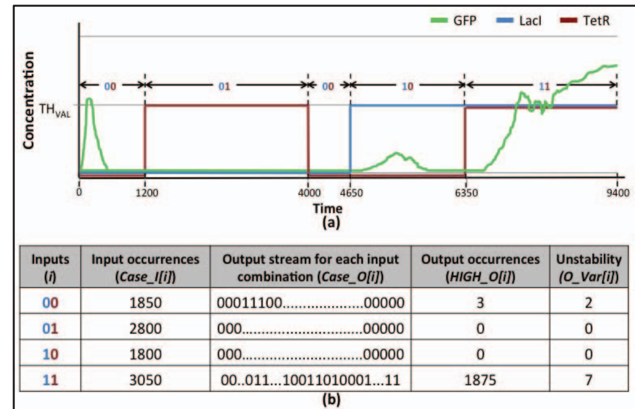
Algorithm 1: Pseudo code of the logic analysis and verification algorithm.

```
  Logic Analysis and Verification Algorithm - Main
1 BEGIN
2   INITIALIZE (N, I_S, O_S, SD_An, Th_VAL, FOV_UD)
    /* N = Total number of input species, I_S = Names of input species, O_S =
       Name of output specie, SD_An = Analog simulation data of I/O species,
       Th_VAL = Threshold value of I/O species, FOV_UD = fraction of
       acceptable variation defined by user */
3   SD_size = Calculate the size of analog simulation data, SD_An
4   SD_Dig = ADC (N, SD_An, SD_size, Th_VAL)
    /* SD_Dig = digital simulation data */
5   (nc, Case_O, Case_I) = CaseAnalyzer (N, SD_size, SD_Dig)
    /* Case_O = Array to hold the output values for each input combination,
       Case_I = Array to hold the number of occurrences of each input
       combination, nc = total number of input cases (or combinations): 2^N */
6   (O_Var, HIGH_O) = VariationAnalyzer (nc, SD_Dig, Case_O)
     /* O_Var = Array to monitor variations in output for each case, nc
        High_O = Array to hold the number of times the output is high for
                 each case, nc */
7   (BoolExpres, PFoBE) = ConsBoolExpr (O_Var, Case_I, HIGH_O, nc,
                                        N, FOV_UD)
     /* BoolExpres = Contains the estimated Boolean expression of a DUT
        PFoBE = Specifies the percentage fitness of estimated Boolean
                expression in the simulation data */
8 END
```

In the simulation of electronic circuits, a logical abstraction is typically applied in which it is only considered if the wire is in high or low state, instead of tracking the exact voltage value. To utilize a similar abstraction level here, the algorithm first converts the analog simulation data into digital data with the help of threshold values extracted from D-VASim. This step is shown as the sub-procedure *ADC* at line 4 in Algorithm 1. The algorithm scans the chosen N input and an output species and converts their analog values in to digital values, based on the threshold value provided. Once the analog data is converted to logic high and low, the exact concentration of proteins are no longer needed to obtain the Boolean logic of a genetic circuit.

The response time of a genetic circuit is important to obtain the correct behavior. Therefore, each input combination must be applied for enough time to observe its correct response on the output species. In electronic circuits, the signals propagate in separate wires and applied voltage remains constant. However, the signals in genetic circuits are molecules drifting in the same volume of a cell and easily merge with the concentrations of other compounds. Due to this, the concentrations of species in genetic circuit always vary, and may go up and down below the threshold level over time. Because of this unstable behavior, for each input combination, it is required to obtain continuous binary streams of output species to extract the correct behavior of a genetic circuit.

The sub-procedure, *CaseAnalyzer* (line 5, Algorithm 1) analyzes the number of times each input combination occurs and logs their corresponding output binary data streams. To understand this procedure, consider the sample simulation plots in Figure 2(a), which are produced from the 2-input genetic AND gate of Figure 1. *CaseAnalyzer*, processes the data and generates output as depicted in the first three columns in Figure 2(b). These columns express, for each input combination, the number of simulated data points as well as the output digital data stream of logic-0 and 1 converted according to the threshold levels. In this example, the case of input combination 00 appears about 1850 times in total. The small glitch between 4650-6350 time units (in Figure 2(a)) indicates the stochastic nature of the model. It shows that the logic-0 of GFP may refer to a concentration which is less than its threshold value but may not be sharply zero. Also, the output of some genetic circuit models is initially high which gradually reduces to zero, as shown in Figure 2(a). These unwanted high peaks should be filtered out to obtain the correct Boolean expression.



Inputs (*i*)	Input occurrences (*Case_I[i]*)	Output stream for each input combination (*Case_O[i]*)	Output occurrences (*HIGH_O[i]*)	Unstability (*O_Var[i]*)
00	1850	00011100....................00000	3	2
01	2800	000.............................00000	0	0
10	1800	000.............................00000	0	0
11	3050	00..011...10011010001...11	1875	7

(b)

Figure 2. Analysis and verification process. (a) Sample plots of 2-input genetic AND gate. (b) Sample data for illustrating the input case and variation analysis.

For each input combination, the corresponding data stream of the output species is also extracted, as shown in the third column of the table shown in Figure 2(b). In this example, the output data stream contains binary 1's for two input combinations – 00 and 11. Furthermore, Figure 2(a) depicts a short period of time in which the output oscillates around the threshold value (between 6350-9400 time units), before entering into a stable logic-1. This happens when both inputs are triggered high (i.e. 11). To examine such scenarios, the digital output data streams, corresponding to each input combination, are analyzed for stability through the sub-procedure, *VariationAnalyzer,* (line 6, Algorithm 1).

VariationAnalzyer examines the output data stream and counts how many times the output oscillates (or varies) between logic-1 and 0. It first calculates the number of times a logic-1 appears for a specific input combination. In the example shown in Figure 2(b), the logic-1 appears for 3 and 1875 times for the input combinations 00 and 11, respectively. It then analyses for each of these input combinations, how many times the output varies, i.e. changing 0-to-1 and 1-to-0. In Figure 2(b) this happens 2 times for input combination 00 and 7 times for 11. Since the output is high when both the inputs are the same, one may end up estimating the logical behavior of this circuit to be an XNOR gate if the simulation data is not filtered out correctly. To obtain a correct Boolean expression, two filtrations of the data are performed by the sub-procedure,

ConstBoolExpr (line 7, Algorithm 1). The first one is the calculation of *fraction of variation* through equation (1);

$$\mathrm{FOV}_{ESTi} = \mathrm{O_Var}[i]/\mathrm{Case_I}[i] \quad (1)$$

Where, i is the input combination at which the output is high at least once; *O_Var[i]* corresponds to the number of variations in the output, for each *i; and Case_I[i]* is the number of times the input combination i occurs in the simulation data. Note that the value of Case_I[i] will always be equivalent to the length of its corresponding output data stream.

In the example shown in Figure 2, the estimated fraction of variations – $\mathrm{FOV}_{\mathrm{EST}}$, for input combinations 00 and 11, are 2/1850 and 7/3050, respectively. This indicates that only a small fraction of output, in comparison to its whole size for specific input combination, is varied. This estimated fraction of variation, $\mathrm{FOV}_{\mathrm{EST}}$, is compared with the user-defined fraction of variation, $\mathrm{FOV}_{\mathrm{UD}}$, and the results are accepted if the estimated value is less than the user-defined one. In our experiments, we allowed up to 25% variation ($\mathrm{FOV}_{\mathrm{UD}} = 0.25$) in the output data streams.

However, this filter alone is not sufficient to obtain the correct Boolean logic of a model. As in the case of the example shown in Figure 2, the algorithm will consider obtaining the output high for both input combinations 00 and 11, based on the estimated value of $\mathrm{FOV}_{\mathrm{EST}}$, and end up obtaining the XNOR logic for this circuit model. Therefore, to handle this situation, another filter is applied according to equation (2), which checks if the number of 1s' in the output binary data stream, for the specific input combination, are greater than half the size of the whole output data stream.

$$HIGH_O[i] > \mathrm{Case_I}[i]/2 \quad (2)$$

Here, i is the input combination at which the output stream is being checked; *HIGH_O[i]* defines the number of 1's in the output stream corresponding to the input combination *i*; and *Case_I[i]* specifies the number of times the input combination *i* occurs in the simulation data. This is equivalent to the length of corresponding output data stream. For our example, this condition holds false for the input combination 00 ($3 \ngtr 1850/2$), but turns true for the input combination 11 ($1875 > 3050/2$). This filter also helps in making sure that the output, for a specific input combination, is certain – either high or low. However, this filtration technique may also produce wrong results if not applied together with the first technique.

Input Combinations (i)	Output Data Stream Case_O[i]	Number of 1s HIGH_O[i]	Case_I[i] = size of Case_O[i]	O_Var[i]
00	0101011111	7	10	5
11	0001111111	7	10	1

Input Combinations (i)	Filtering Condition 1 FOV_{EST_i} = O_Var[i]/Case_I[i]	Filtering Condition 2 HIGH_O[i] >Case_I[i]/2
00	*= 5/10 = 0.5 (50% variable output)*	7> (10/2) ➔ TRUE
11	=1/10 = 0.1 (10% variable output)	7> (10/2) ➔TRUE

Figure 3. An example showing how both filters are useful, when applied together, in obtaining the correct Boolean expression.

To understand this, consider the example case shown in Figure 3, where the output binary data streams of two different input cases, 00 and 11, are shown. The number of 1s in the output stream, for both the cases, is the same; however, the output is highly oscillatory for the input case, 11. The algorithm therefore discards (in this case if $\mathrm{FOV}_{\mathrm{UD}} \leq 0.5$) this unstable output and do not consider it while constructing the Boolean expression.

To filter out the results, both abovementioned conditions should be satisfied. The Boolean expression is then constructed for each filtered result. In the end, the algorithm estimates the *percentage fitness of estimated Boolean expression* (PFoBE), in the simulation data, according to equation (3).

$$\mathrm{PFoBE} = 100 - \left(\textstyle\sum_i FOV_{EST_i}/nc\right) \times 100 \quad (3)$$

Here, i is the input combination at which the filtered output stream is high; $\mathrm{FOV}_{\mathrm{EST}i}$ is the estimated fraction of variation in the output data stream for i^{th} input combination; and *nc* denotes the total number of input combinations.

III. Experimental Results

The proposed algorithm is tested on the SBML models of 15 genetic circuits. This set includes 1 to 3-inputs genetic logic circuits, which are composed of 1-7 genetic logic gates containing 3-26 genetic components. The five genetic circuit models are obtained from [12] and the remaining 10 are the models of real genetic circuits acquired from [11]. The circuits from [11] were first designed on a tool, named *Cello*, which generates the *Synthetic Biology Open Language* (SBOL) [13] file. Unlike SBML, the SBOL representation does not describe the behavior of a biological model. The behavioral description of a model in SBML is expressed in terms of mathematical equations of the reaction kinetics between molecular species. We, therefore, first used the SBOL-SBML converter [14] to generate the behavioral model of the real genetic circuits [11]. The SBML files generated from this process are then loaded in D-VASim to perform the experimentation followed by the logic verification of these circuits.

In our experiments, we ran each circuit for 10,000 simulation time units, assuming a value of 1000 time units for the propagation delay of all circuits. This means that during simulation, each input combination is applied for at least 1000 time units. Also, a threshold value of 15 molecules is used for all circuits. Due to space limitations, the simulation data analytics of only three circuits (0x0B, 0x04, and 0x1C), from [11], are shown in Figure 4. These analytics are used to obtain the logical behavior of the circuits. In Figure 4, *Case_I* indicate the number of times each input combination occurs during 10,000 time units of simulation. It further includes the number of times the output of a circuit remains high (*High_O*) for that particular input combination along with the number of variations in the output data (*Var_O*). The Boolean expressions as well as the percentage fitness for these circuits are also included in Figure 4. In this figure, the output variation is not too high for any of the output states of each of the three circuits. For example, in the case of circuit 0x0B, the output state appears to be logic-1 for the input combination 100 and seems quite stable having very low variation value of 2. The reason why the input combination is 100 has so many logic-1 output states is because the output is high for the previous input combination 011. When the input combination is changed from 011 to 100, the output starts to decay gradually, and remains high until it passes by the threshold level. This input combination should, therefore, be included in the Boolean expression, but however filtered out using equation (2), because for 3587 times of the input combination 100 occurs during the entire simulation, the

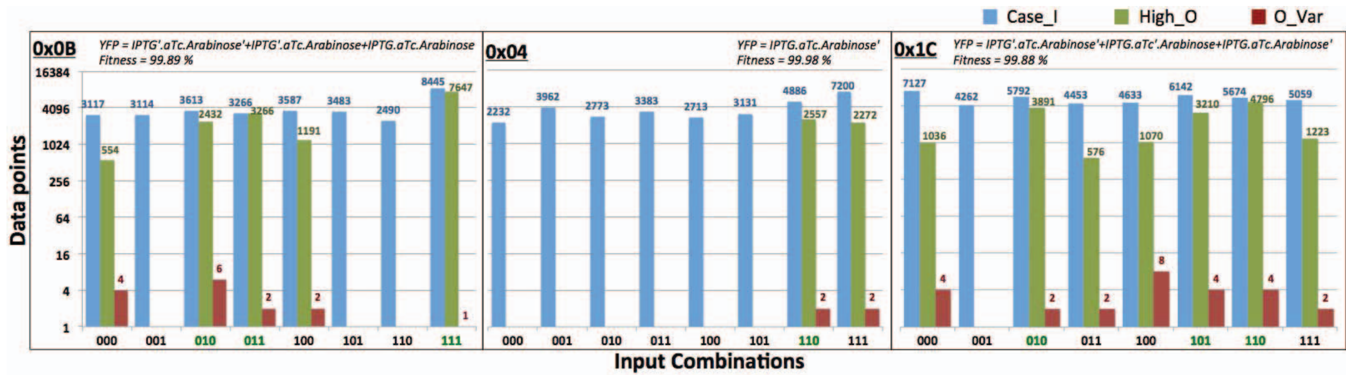


Figure 4. Analytical simulation data, Boolean expression and percentage fitness of three circuits (0x0B, 0x04 and 0x1C) obtained from [11].

corresponding output remains high for 1191 times (<3587/2). It is therefore obvious that like electronic circuits, where the output state may be incorrect if the inputs are changed before the propagation delay has elapsed, the correct behavior of a genetic circuit can only be obtained when each possible input combination is applied for sufficient amount of time. Similarly, the analytical data for the circuits 0x04 and 0x1C shows that the filters we have applied help in obtaining the correct Boolean logic. In Figure 4, the input combinations, at which the circuit's output is expected to be high, are highlighted in green color along the x-axis.

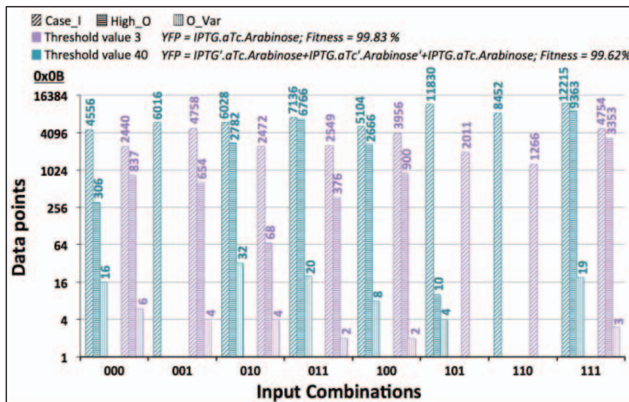


Figure 5. Analytical data of circuit 0x0B for threshold values 3 and 40.

We further analyzed the behavior of genetic circuits by varying the threshold value of input concentrations to very low (3 molecules) and very high (40 molecules), and observed that the same circuits behave differently. Figure 5 shows the comparison of simulation data for the circuit 0x0B for the abovementioned two threshold values. In this figure, it can be noticed that the output response for a threshold value of 3 molecules, is entirely different and it behaves like a 3-input AND gate. This is because the applied input concentration is too weak to trigger the output concentration; but when applied together i.e., 111, the output is triggered high to satisfy the applied filters.

On the other hand, the 0x0B circuit has two wrong states (shown in the Boolean expression) when 40 molecules are applied as an input concentration. For this case of threshold value, the output response also seems to oscillate between logic-high and low for large number of times (Figure 5) as compared to other circuits (Figure 4). This is because the concentration levels of input and output species are not clearly distinguishable when the applied input concentration is high.

IV. Conclusion

In this research, we presented an algorithm to analyze and verify the intended behavior of genetic logic circuits. It is shown experimentally that the circuit may not behave as expected if the circuit parameter(s), like threshold value, are varied. This may help users to analyze the circuit's behavior and robustness for different parameter sets before creating them in the laboratory. We also observed that the proposed algorithm takes about 8.4 seconds to analyze the logic of a complex genetic circuit with significantly large-sized data. As the experimentation in the laboratory requires a couple of hours [11] to analyze even a single output state, the proposed simulation-based approach is likely to be useful for genetic circuit designers to analyze the intended logic of genetic circuits prior to their implementation and testing in the laboratory.

Acknowledgement

We would like to thank the CIDAR Lab (Boston University) and Chris Myers (University of Utah) for providing us the SBOL/SBML models of genetic circuits.

References

[1] A. Arkin, "Setting the standard in synthetic biology", *Nature Biotech*, no. 26, pp 771-774, 2008.

[2] J C. Anderson et al., "Environmentally controlled invasion of cancer cells by engineering bacteria", *J. Mol. Biol.,* 355, pp. 619-627, 2006.

[3] S. Atsumi and J. C. Liao, "Metabolic engineering for advanced biofuels production from Escherichia coli.", *Curr. Opin. Biotech.*, 19, 5, pp. 414-419, 2008.

[4] I Cases and V De Lorenzo, "Genetically modified organisms for the environment: stories of success and failure and what we have learned from them", *Int. Microbiol.,* 8, pp. 213-222, 2005.

[5] Mario A Marchisio *et al.*, "Computational design tools for synthetic biology", *Curr. Opin. Biotechnol.*, vol. 20, no. 4, 2009.

[6] H. H. McAdams and A. Arkins, "It's a noisy business! Genetic regulation at the nanomolar scale", *Trends Genet.,* vol. 15, issue 2, pp. 65-69, 1999.

[7] Gillespie, D.T., "Exact stochastic simulation of coupled chemical reactions", *J. Chem. Phys.*, vol. 81, no. 25, pp. 2340-2361, 1977.

[8] H. Baig and J. Madsen, "D-VASim – An interactive Virtual Laboratory Environment for the Simulation and Analysis of Genetic Circuits", *Bioinformatics,* September, 2016.

[9] The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core, October 06, 2010.

[10] H. Baig and J. Madsen, "Logic and Timing Analysis of Genetic Logic Circuits using D-VASim", *8th IWBDA*, August 16-18, 2016.

[11] AA Nielsen *et al.*, "Genetic circuit design automation", *Science*, vol. 352, issue 6281, 2016.

[12] Chris J. Myers, "Engineering Genetic Circuits", Chapman & Hall/CRC Press, July 2009.

[13] B. Bartley *et al.*, "Synthetic Biology Open Language (SBOL) version 2.0.0", *J. Integrative Bioinformat.*, vol. 12, no. 2, 2015.

[14] N. Roehner *et al.*, "Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language", *ACS Synth. Biol.*, vol. 4, no. 8, pp. 873-879, 2015.