A compilation scheme to provide high level language abstractions to the synthesis process of biological devices

Alek Frohlich[1], **Gustavo Biage**[1]

[1]Departamento de Informatica e Estatística – Universidade Federal de Santa Catarina
Florianopolis – SC – Brazil

{alek.frohlich,gustavo.c.biage}@grad.ufsc.br

Abstract. *The growth of research areas such as synthetic biology and systems biology leads to an increased tendency to develop larger mathematical models for describing complex biological behavior. In order for those models to be implementable in living systems, scientists must have access to a synthesis process of biological devices that is robust and automated. This process ought to provide higher levels of abstraction than the ones found by tinkering with low level components such as promoters and transcription factors. The present work aids the development of such processes by translating a subset of the gro programming language into SMBL documents, and thus speeding up the synthesis process due to facilitating the design of biological devices through the use of a higher level language.*

1. Introduction

The development of an automated synthesis process is essential to the scalability of synthetic biological devices. Without it, each device has to be designed individually, leading to an error-prone process not well suited for the development of large devices. Many substitutes to this primitive procedure have been proposed, one of them, the Cello framework, automates genetic circuit design with the assistance of a genetic component library. The framework searches the component library for parts compliant with the specifications of the Verilog code to produce an equivalent implementable genetic circuit that is also minimum cost [Nielsen et al. 2016].

Cello was heavily inspired by the synthesis process of semi-conductor based electronics. The inspiration is made clearer by the choice of Verilog as the framework's interface. Such an approach may not be ideal as there clearly isn't a one to one correspondence between both processes. An alternative would be to model the high level behavior of biological devices through a language capable of accounting for their unique aspects, e.g. the uncertainty associated with it's reactions occurrence[1].

Systems Biology Markup Language (SBML) is a useful language for describing biochemical models of synthetic and systems biology [Hucka et al. 2003]. Although not being able to fully represent the aforementioned aspects of biological devices, SBML does manage express them in more natural terms. However, one may question it's expressiveness as a programming language considering that even small models can be cumbersome to write. The present work proposes a solution that circumvents SBML's lack of high level abstractions, enabling it to be used as the entry point to the physical implementation

[1]The concentration and distribution of molecules dictate the whether biochemical reactions occur

stage of a synthesis process. The solution consists in abstracting many of SBML's core language features in favor of higher level constructs found in the gro programming.

Gro is a language for programming, modeling, specifying and simulating the behavior of cells in growing microcolonies of microorganisms. Gro models are comprised of global variables, program declarations and bacterial instantiations. Programs behave similarly to functions in imperative programming languages. They are used to describe a functionality that will be executed at every time step of the simulation. Inside such construct, there may appear variable definitions and guarded code blocks. Guarded blocks are blocks of code that only execute in case the Boolean expression guarding them evaluates to true. A bacteria can be instantiated with any amount of said programs [Jang et al. 2012].

The language parser works in conjunction with a framework called Tellurium. Tellurium is a Python-based modeling environment for systems and synthetic biology [Choi et al. 2018]. The language parser verifies the correctness of the gro source code. Valid programs are translated to an intermediate format which, in turn, becomes the final SBML document through the assistance of Tellurium.

The rest of the paper is structured as follows: Section 2 introduces the language parser and explain the process of converting gro files to SBML models from start to finish. Section 3 presents a case study on the application of the proposed language parser along with an evaluation of its abstractions, and Section 4 presents our final considerations and future works.

2. Code Generation

In this section we describe gro's compilation process (depicted at Figure 1): First, the gro source code is validated, that is, it's checked against syntax errors, such as missing semicolon in rate statements and semantic error such as using undeclared variables; Then the code gets translated to an intermediate representation called Antimony [Smith et al. 2009]; Finally, Tellurium uses libSBML and libAntimony to convert the given Antimony file down to an SBML document.

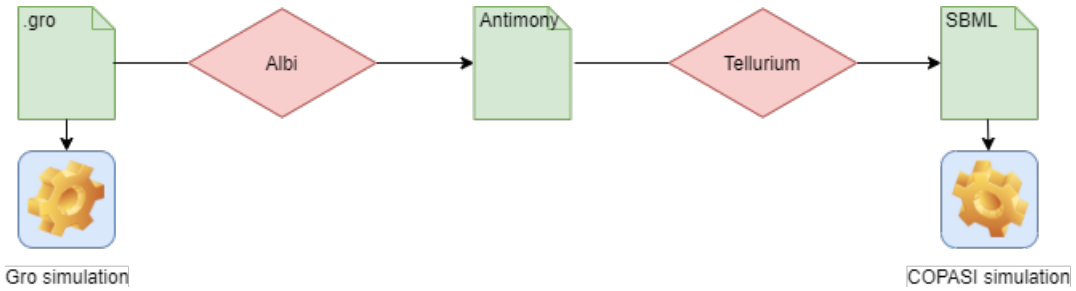

Figure 1. Flow chart of the compilation steps.

2.1. Parsing

As mentioned before, the code submitted to the compiling process is parsed and translated to an intermediate language called Antimony, a modular model definition language [Smith et al. 2009]. The proposed parser recognizes a subset of gro's syntax. The subset is composed of global variable declarations, program definitions and E. coli instantiations. Parameters, messages and guarded statements are also recognized, however no semantic

value is assigned to them given that control statements and user interaction do not fit the context of an SBML model. The following is an example of gro code containing the syntactical constructs just described.

1. Example gro code.

```
include gro

program prog() := {
    E := 25;
    F := 75 + E;
    K := 2 + F + E;
    rate(K) : { E := E + 1, F := F - 1 };
};

program dupl(X) := {
    E := 25 + X;
    F := 75 + E;
    K := 2 + F + E;
    rate(K*40) : { E := E + 1, F := F - 1 };
};

ecoli([], program prog() + dupl(10));
ecoli([], program prog());
```

2.2. Translating

After the parsing, the code is translated to Antimony. The resultant file will be further submitted to Tellurium. Gro code translation into Antimony occurs as follows:

(i) Gro files are represented as an Antimony file.
(ii) E. coli instantiated in the gro source are declared as compartments.
(iii) Rate statements are represented as $s_1\,\mathrm{R}_1 + s_2\,\mathrm{R}_2 + ... + s_n\,\mathrm{R}_n \longrightarrow t_1\,\mathrm{P}_1 + t_2\,\mathrm{P}_2 + ... + t_n\,\mathrm{P}_n$. Where the stoichiometric constants s_i and t_j denote the number of times the variables R_i and P_j were decremented/incremented inside the guarded block.
(iv) Variables that do not appear in such reactions are considered global constants.
(v) Code blocks guarded by gro's rate function appear next to their reactions.

Also, symbols declared inside program definitions are renamed to avoid name space clashes between multiple programs instantiated in the same compartment.

2.3. Compiling

As mentioned before, after translation the code is submitted to Tellurium to generate the final SBML document. Listing 2 shows a cleaned version of the generated SBML where some reactions and SBML syntax has been omitted to avoid cluttering[2]. The gro to SBML mapping is presented bellow:

(i) Gro files are represented as models.
(ii) E. coli are represented as compartments.
(iii) Variables are identified as species when participating in a reaction, otherwise they are marked as constant and identified as parameters.

[2]The reaction shown bellow is the one from *prog*, instantiated in the first E. coli. Other tags not shown include the preamble declaration of the XML and the SBML document.

(iv) Reactions are marked irreversible[3] and have their lists of reactants and products filled based on what variables are being decremented and incremented, respectively.

(v) Expressions inside rate functions that guard blocks of code become kinetic law constants inside reactions.

2. Generated SBML document.

```
<model metaid="__main" id="__main">
  <listOfCompartments>
    <compartment id="ECOLI0" spatialDimensions="3" size="1" constant="true"/>
    <compartment id="ECOLI1" spatialDimensions="3" size="1" constant="true"/>
  </listOfCompartments>
  <listOfSpecies>
    <species id="ECOLI0_prog_E" compartment="ECOLI0" initialConcentration="25"
        constant="false"/>
    <species id="ECOLI0_prog_F" compartment="ECOLI0" initialConcentration="100"
        constant="false"/>
    <species id="ECOLI0_dupl_E" compartment="ECOLI0" initialConcentration="35"
        constant="false"/>
    <species id="ECOLI0_dupl_F" compartment="ECOLI0" initialConcentration="110"
        constant="false"/>
    <species id="ECOLI1_prog_E" compartment="ECOLI1" initialConcentration="25"
        constant="false"/>
    <species id="ECOLI1_prog_F" compartment="ECOLI1" initialConcentration="100"
        constant="false"/>
  </listOfSpecies>
  <listOfParameters>
    <parameter id="ECOLI0_prog_K" value="127" constant="true"/>
    <parameter id="ECOLI0_dupl_K" value="147" constant="true"/>
    <parameter id="ECOLI1_prog_K" value="127" constant="true"/>
  </listOfParameters>
  <listOfReactions>
    <reaction id="_J0" reversible="true" fast="false">
      <listOfReactants>
        <speciesReference species="ECOLI0_prog_F" stoichiometry="1" constant="true"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="ECOLI0_prog_E" stoichiometry="1" constant="true"/>
      </listOfProducts>
      <kineticLaw>
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <ci> ECOLI0_prog_K </ci>
        </math>
      </kineticLaw>
    </reaction>
    ...
  </listOfReactions>
</model>
```

3. Case Study: The Repressilator

This section presents the experiment used to evaluate the proposed compilation process. The experiment consists on executing the Repressilator (a well-known biological device) in gro's built-in simulator and comparing the Octave output to the simulation of the SBML document obtained by compiling the code according to the process described at Section 2. The SBML simulations were done through Copasi, a pathway simulator [Hoops et al. 2006].

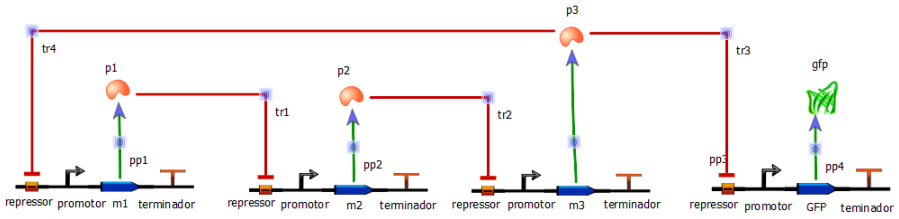

Figure 2. Repressilator's genetic regulatory network.

3.1. Repressilator Model

The Repressilator is a genetic regulatory network constituted by stitching together promoter-repressor pairs. It was first conceptualized by [Elowitz and Leibler 2000], who used the genes LacI and letR from E. coli and cI from phage lambda to build a three-repressor scheme. As illustrated in Figure 2, the Repressilator works based on the fact that each gene expresses a protein which represses the next gene in the loop, meaning that a constituent gene in high concentration represses itself indirectly, leading to oscillatory behavior. In the original experiment, there was another biological component involved: the reporter, a green fluorescent protein (GFP) producing gene. It was used to visualize the network's oscillation.

3.2. Mathematical Model

As shown in [Elowitz and Leibler 2000], the high-level model from Figure 2 could also be presented as the following system of Ordinary Differential Equations (ODE).

$$\begin{aligned}
\frac{dm_1(t)}{dt} &= \alpha_0 + \frac{\alpha}{1+p_3(t)^n} - m_1(t) & \frac{dp_1(t)}{dt} &= \beta \cdot m_1(t) - \beta \cdot p_1(t) \\
\frac{dm_2(t)}{dt} &= \alpha_0 + \frac{\alpha}{1+p_1(t)^n} - m_2(t) & \frac{dp_2(t)}{dt} &= \beta \cdot m_2(t) - \beta \cdot p_2(t) \\
\frac{dm_3(t)}{dt} &= \alpha_0 + \frac{\alpha}{1+p_2(t)^n} - m_3(t) & \frac{dp_3(t)}{dt} &= \beta \cdot m_3(t) - \beta \cdot p_3(t)
\end{aligned} \tag{1}$$

3.3. Gro Code

The gro code for the Repressilator is presented below.

```
rate(alfa0) : { m1 := m1 + 1, m2 := m2 + 1, m3 := m3 + 1 };
rate(alfa / (1 + p3^n)) : {m1 := m1 + 1};
rate(m1) : {m1 := m1 - 1};
rate(alfa / (1 + p1^n)) : {m2 := m2 + 1};
rate(m2) : {m2 := m2 - 1};
rate(alfa / (1 + p2^n)) : {m3 := m3 + 1};
rate(m3) : {m3 := m3 - 1};
rate(beta*m1) : {p1 := p1 + 1};
rate(beta*p1) : {p1 := p1 - 1};
rate(beta*m2) : {p2 := p2 + 1};
rate(beta*p2) : {p2 := p2 - 1};
rate(beta*m3) : {p3 := p3 + 1};
rate(beta*p3) : {p3 := p3 - 1};
```

[3]Gro reactions are marked irreversible since it does not support reversible ones.

Moreover, it is possible to trace a relation of the mathematical model presented at Section 3.2 and the above code snippet. Each reaction in the source code groups variables whose derivatives contain the same sub-expression. The rate of such reaction is the common sub-expression. The decision of which variable is a reactant and which is a product is determined by the sign of the sub-expression. An example of such construction is depicted bellow:

$$\xrightarrow{\alpha_0} \mathrm{m}_1 + \mathrm{m}_2 + \mathrm{m}_3 \tag{2}$$

The sub-expression α_0 appears at three left equations of (1). As in all three equations α_0 is positive, it's concluded that m_1, m_2 and m_3 are all products and there isn't an explicit reactant. This means that (2) is an open reaction on the rate of α_0. By repeating this process until no sub-expression remains, the resulting SBML would have 7 reactions. However, as gro does not support reversible reactions directly through it's *rate* function, each reversible reaction is thus split into two new reactions: One for the forward part and another for the reverse. On doing so, the final gro model packs a total of 13 rate statements.

4. Results and Discussion

This section presents a simulation of the gro model shown in the last section and analyses the results. As gro's simulation method is stochastic, it is harder to compare it against a deterministic simulation of the differential equations in (1). Therefore, an octave model was used to compare the compiled SBML document execution on Copasi to the results of a deterministic simulation of (1).

4.1. Gro Simulation

Figure 3 presents the results of simulating the gro model for the Repressilator. Simulation parameters taken from [Ingalls 2013]: $\alpha_0 = 0.03$ (molecules per $cell \cdot min^{-1}$), $\alpha = 298.2$ (molecules per $cell \cdot min^{-1}$), $\beta = 0.2$ (min^{-1}), $n = 2$ and $\delta t = 0.0001$. From it, one may notice that the period and amplitude of the oscillations varied during the course of time, probably due to stochastic fluctuations. Nonetheless, the graph still reminds of oscillatory behavior.

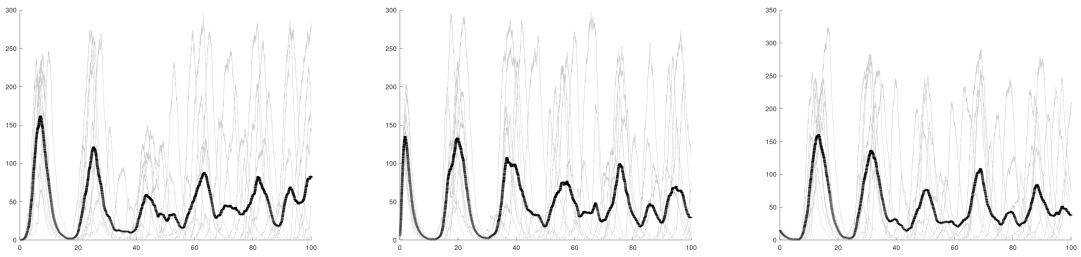

Figure 3. Multiple simulations of the Repressilator gro model. From left to right: p1, p2, p3. Black lines are used to indicate an average simulation, whilst the various grey lines are used to indicate the concentration of the respective protein at each individual simulation.

4.2. Compiled SBML Simulation and Comparison

This section presents the results of a simulation of the compiled SBML document on Copasi, along with a comparison between it and a simulation of the original ODEs (1) in Octave. The Copasi simulation is presented on the left part of Figure 4 and the Octave one to the right.

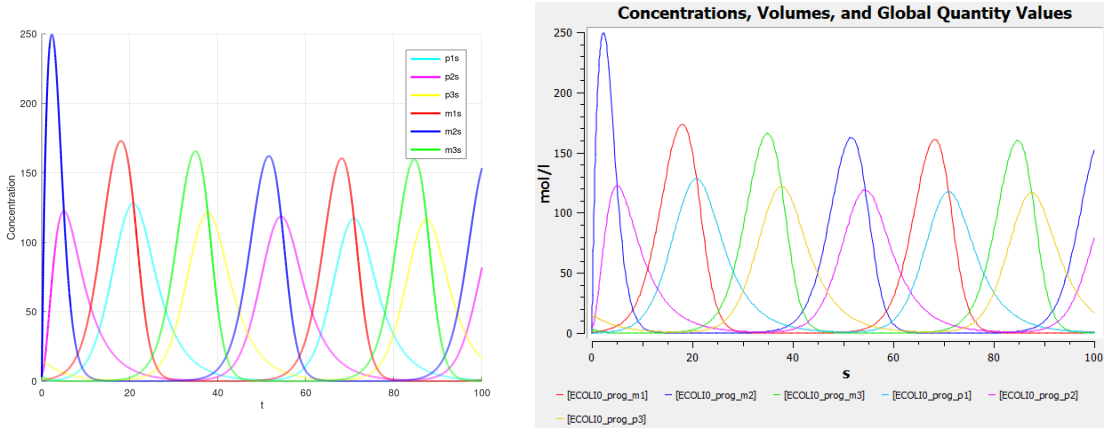

Figure 4. Comparison between the Octave model and compiled gro model for the Repressilator.

As depicted at Figure 4, both simulations yielded the same values on concentration over the simulation's duration. Such results demonstrate the effectiveness of the proposed compilation scheme, which besides reducing the coding effort for biological devices, grants a successful simulation of the generated SBML model.

5. Conclusion & Future Works

This paper presented a compilation scheme to the design and specification of biological devices using a framework that abstracts low level SBML constructs with higher level ones found in the gro programming language. The framework combines a code translator and Tellurium to map gro files to SBML documents. The proposed framework along with the gro mappings were illustrated through the compilation of a gro model for the Repressilator into an equivalent SBML document. They were also evaluated based on the comparison of two simulations: on for the resulting model and another for the original model (1).

As future work, we intend to address the set of unrecognized gro language features. In particular, we plan to incorporate signals into our framework, enabling cell to cell communication. We also want to annotate the generated SBML document in cases where the source code contains features not recognized by our framework. This should, at the very least, notify tools which use our framework that some part of the source code was not translated into the SBML document and in other cases help the tool figure out how to deal with the syntactical mismatch.

References

Choi, K., Medley, J. K., König, M., Stocking, K., Smith, L., Gu, S., and Sauro, H. M. (2018). Tellurium: An extensible python-based modeling environment for systems and synthetic biology. *Biosystems*, 171:74–79.

Elowitz, M. B. and Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338.

Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., , the rest of the SBML Forum:, Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J.-H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Novere, N. L., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.

Ingalls, B. (2013). *Mathematical modeling in systems biology : an introduction*. The MIT Press, Cambridge, MA.

Jang, S. S., Oishi, K. T., Egbert, R. G., and Klavins, E. (2012). Specification and simulation of synthetic multicelled behaviors. *ACS Synthetic Biology*, 1(8):365–374.

Nielsen, A. A. K., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016). Genetic circuit design automation. *Science*, 352(6281).

Smith, L. P., Bergmann, F. T., Chandran, D., and Sauro, H. M. (2009). Antimony: a modular model definition language. *Bioinformatics*, 25(18):2452–2454.