BioCRNpyler: Compiling Chemical Reaction Networks from Biomolecular Parts in Diverse Contexts

William Poole,[*],[†] Ayush Pandey,[†] Andrey Shur,[†] Zoltan A. Tuza,[‡] and Richard M. Murray[†]

E-mail: wpoole@caltech.edu

Abstract

Biochemical interactions in systems and synthetic biology are often modeled with Chemical Reaction Networks (CRNs). CRNs provide a principled modeling environment capable of expressing a huge range of biochemical processes. In this paper, we present a software toolbox, written in python, that complies high-level design specifications to CRN representations. This compilation process offers four advantages. First, the building of the actual CRN representation is automatic and outputs Systems Biology Markup Language (SBML) models compatible with numerous simulators. Second, a library of modular biochemical components allows for different architectures and implementations of biochemical circuits to be represented succinctly with design choices propagated throughout the underlying CRN automatically. This prevents the often occurring mismatch between high-level designs and model dynamics. Third, high-level design specification can be embedded into diverse biomolecular environments, such as cell-free extracts and *in vivo* milieus. Finally, our software toolbox has a parameter database, which allows users to rapidly prototype large models using very few parameters which can be customized later. By using BioCRNpyler, users can easily build, manage, and explore sophisticated biochemical models using diverse biochemical implementations, environments, and modeling assumptions.

1 Introduction

Chemical Reaction Networks (CRNs) are the workhorse for modeling in systems and synthetic biology.[1] The power of CRNs lies in their expressivity; CRN models can range from physically realistic descriptions of individual molecules to coarse-grained idealizations of complex multi-step processes.[2] However, this expressivity comes at a cost. Choosing the right level of detail in a model is more an art than a science. The modeling process requires careful consideration of the desired use of the model, the available data to parameterize the model, and prioritization of certain aspects of modeling or analysis over others. Additionally, Biological CRN models can be incredibly complex including dozens or even hundreds or thousands of species, reactions, and parameters. Maintaining complex hand-built models is challenging and errors can quickly grow out of control for large models. Software tools can answer many of these challenges by automating and streamlining the model construction process.

Due to CRN's rich history and diverse applications, the available tools for a CRN modeler are vast and includes: extensive software to generate and simulate CRNs, databases of models, model analysis tools, and many more.[3–7] However, relatively few tools exist to aid in the automated construction of general CRN models from simple specifications. For example, even though synthetic biologists have adopted a module and part-driven approach to their laboratory work[8], models are still typically built by hand on a case-by-case basis. Recognizing the fragile nature of hand built models, several synthetic biology design automation tools have been developed for specific purposes such as implementing transcription factor or integrase-based logic.[9,10] These tools indicate a growing need for design and simulation automation in synthetic biology, as part and design libraries are expanded.

As the name would suggest, the BioCRNpyler (pronounced bio-compiler) package is a Python package that compiles CRNs from simple specifications of biological motifs and contexts. This package is inspired by the molecular compilers developed by the DNA-strand displacement community and molecular programming communities which, broadly speaking, aim to compile models of DNA circuit implementations from simpler CRN specifications[11–13] or rudimentary programming languages.[14,15] However, BioCRNpyler differs from these tools for two main reasons: first, it is not focused only on DNA implementations of chemical computation, and second, it does not take the form of a traditional programming language. BioCRNpyler combines specifications consisting of synthetic biological parts and systems biology motifs that can be reused and recombined in diverse biochemical contexts at customizable levels of model complexity. In other words, BioCRNpyler compiles detailed CRN models from abstract specifications of a biochemical system. Importantly, BioCRNpyler is not a CRN simulator—models are saved in the Systems Biology Markup Language (SBML)[16] to be compatible with the user's simulator of choice. Figure 1 provides motivating examples for the utility of BioCRNpyler by demonstrating the rapid construction of diverse CRNs by reusing common parts and modifying the modeling context.

There are many existing tools that provide some of the features present in BioCRNpyler. Systems Biology Open Language (SBOL)[17] uses similar abstractions to BioCRNpyler but is fundamentally a format for sharing DNA-sequences with assigned functions and does not compile a CRN. The software package iBioSim[18,19] compiles SBOL specifications into SBML models and performs analysis and simulation. Although BioCRNpyler is capable of similar kinds of compilation into SBML, it is not a simulator. Importantly, BioCRNpyler does not hard code how models are compiled—instead it should be viewed as a customizable software compilation language that can be applied to compile many kinds of systems beyond genetic networks. The rule-based modeling framework BioNetGen[20] allows for a system to be defined via interaction rules which can then be simulated directly or compiled into a CRN. Internally, BioCRNpyler functions similarly to this rule based-modeling compilation.

Similarly to PySB[21], BioCRNpyler provides a library of parts, mechanisms, and biomolecular contexts that allow for models to be succinctly produced without having to manually specify and verify many complex rules. Finally, the MATLAB TX-TL Toolbox[22] can be seen as a prototype for BioCRNpyler but lacks the objected-oriented framework and extendability beyond cell-free extract systems.

BioCRNpyler is a purposefully suited to *in silico* workflows because it is an extendable objected-oriented framework that integrates existing software development standards and allows complete control over model compilation. Simultaneously, BioCRNpyler accelerates model construction with extensive libraries of biochemical parts, models, and examples relevant to synthetic biologists, bio-engineers, and systems biologists. The BioCRNpyler package is available on GitHub[23] and can be installed via the Python package index (PyPi).

2 Motivating Examples

This section highlights the ease-of-use of BioCRNpyler through several well-known synthetic biology examples. As a summary, Figure 1 demonstrates the utility of compiling CRNs with BioCRNpyler. The names of python classes are highlighted `typographically` and are defined more thoroughly in later sections. Time-course simulations in Figure 1 were done with Bioscrape[24] and circuit diagrams were created with DNAplotlib.[25]

2.1 Inducible Repression, Toggle Switch and Repressilator

Models A, B, and C show three archetypal motifs from synthetic biology: inducible repression, a bistable toggle switch[26], and the repressilator[27]. All three of these examples are created by reusing the same `Components` wired together in different ways as described in Section 6. The ability to reuse `Components` allows for convenient design-space exploration of different circuit architectures. Furthermore, as explained in Section 2.2 and 2.4, examples D, E and F show how these `Components` can be tested in different contexts by chang-

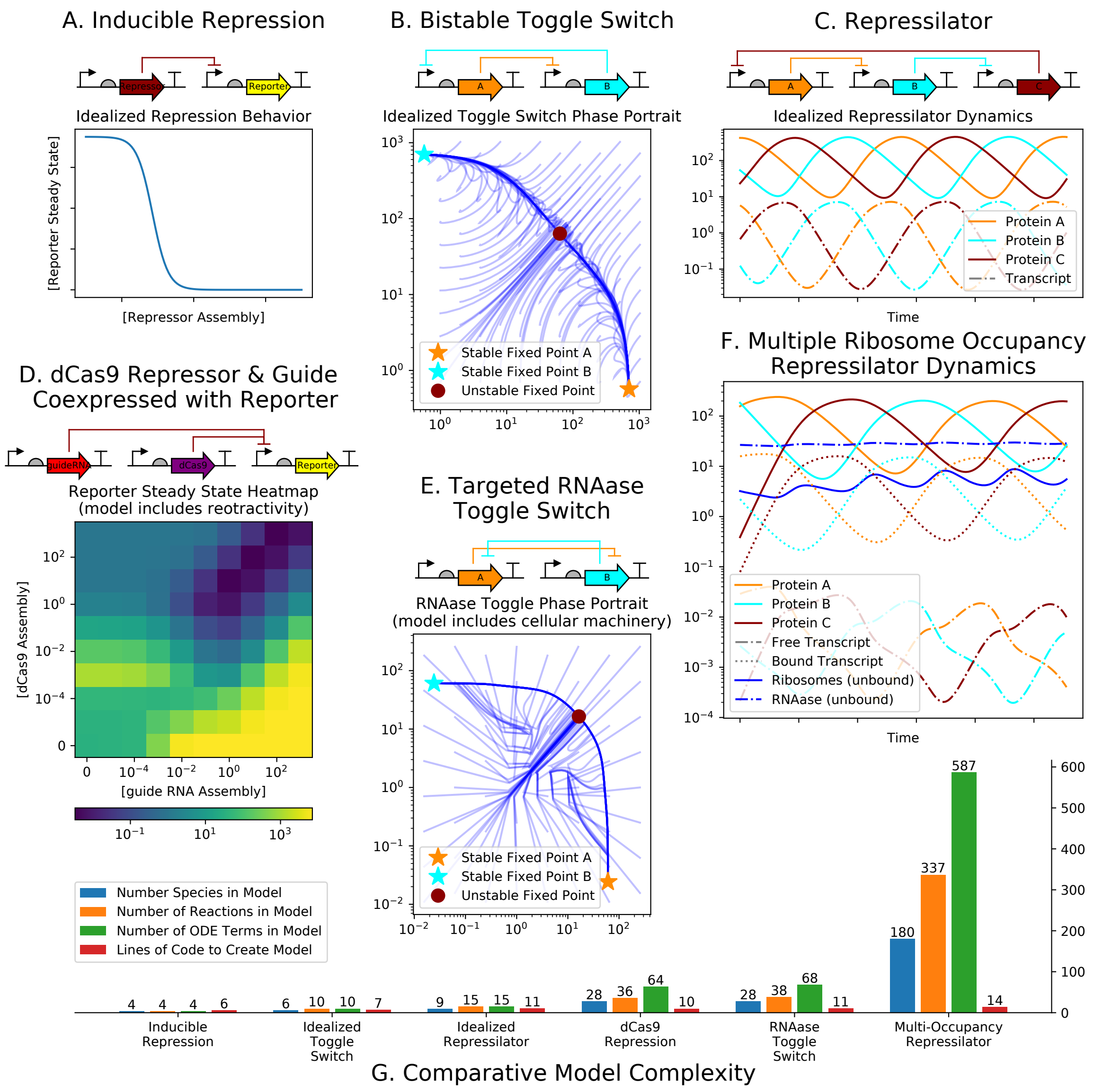

Figure 1: Motivating Examples. The idealized models (A, B, and C) do not model the cellular environment; genes and transcripts transcribe and translate catalytically. **A.** Schematic and simulation of a constiutively active repressor gene repressing a reporter. **B.** Schematic and simulations of of a toggle switch created by having two genes, A and B, mutually repress each other. **C.** Schematic and dynamics of a 3-repressor oscillator. The detailed models (D, E, & F) model the cellular environment by including ribosomes, RNAases and background resource competition for cellular resources. **D.** A dCas9-guideRNA complex binds to the promoter of a reporter and inhibiting transcription. Heatmap shows retroactivity caused by varying the amount of dCas9 and guide-RNA expressed. The sharing of transcription and translational resources gives rise to increases and decreases of reporter even when there is very little repressor. **E.** A proposed model for a non-transcriptional toggle switch formed by homodimer-RNAase; the homodimer-RNAase made from subunit A selectively degrades the mRNA producing subunit B and visa-versa. **F.** A model of the Repressillator exploring the effects of multiple ribosomes binding to the same mRNA. **G.** Histogram comparing the sizes of models A-F and the amount of BioCRNpyler code needed to generate them.

ing the `Mechanisms` and `Mixtures` used to compile the `Components` resulting in nuanced implementation-specific and context-specific models.

2.2 dCas9 Repressor and Guide RNA Coexpressed with Reporter

Figure 1D builds upon the repression Model A by modeling an implementation consisting of a guide-RNA and dCas9 `ChemicalComplex` that also acts as a repressor by inhibiting RNA-polymerase binding to the reporter `DNAassembly`.[28] Model A also includes more intracellular context such as nucleases and ribosomes. By including cellular machinery in the model, the co-expression of dCas9 and the guide RNA is able to influence the reporter via loading effects and retroactivity, which can cause unintended increases and decreases in reporter expression[29] even when only the guide-RNA or dCas9 is present.

2.3 Targeted RNAase Toggle Switch

Figure 1E models a hypothetical toggle switch that functions at the RNA level instead of the transcriptional level. Each `DNAassembly` expresses a subunits A and B of two homodimer-RNAase. The homodimer-RNAase made from subunit A selectively degrades the mRNA producing subunit B and visa-versa. Such a system could potentially be engineered via RNA-targeting Cas9[30] or more complex fusion proteins.[31]

2.4 Multiple Ribosome Occupancy Repressilator Dynamics

Figure 1F illustrates how BioCRNpyler can be used to easily generate more realistic and complex models of biochemical processes in order to validate if model simplifications are accurate. It is common practice in transcription and translation models to use an enzymatic process consisting of a single ribosome (R) to a transcript (T) which then produces a single protein (P). This translation `Mechanism` could be written as: $R + T \leftrightharpoons R : T \rightarrow R + T + P$. Indeed, both example models D and E use such a simplification. However, experiments show that in fact many ribosomes can co-occupy the same mRNA.[32] By changing the underlying translation `Mechanism` to model multiple ribosomal occupancy of a single mRNA, a considerably more complex Repressilator model was created. Importantly, this model exhibits very similar behavior to the simpler model, suggesting that multi-occupancy of ribosomes on mRNA can be neglected in these kinds of genetic regulatory circuits.

2.5 Network complexity

Finally, the bottom histogram Figure 1G shows that even as the size of the underlying CRN grows, the amount of BioCRNpyler code that is needed to generate the model remains very small. This enables the generation of large and complex models with greater accuracy and lower chance of human error. For example, imagine writing down ODEs with hundreds of terms and then trying to systematically modify the equation: human error is nearly inevitable. By using CRN compilation, models can be easily produced, modified, and maintained.

2.6 Parameter Database

Importantly, all these examples in this Section make use of the same underlying set of 10-20 default parameters demonstrating how BioCRNpyler's `ParameterDatabase` makes model construction and simulation possible even before detailed experiments or literature review.

3 Framework and Compilation Overview

BioCRNpyler is an open-source Python package that compiles high-level design specifications into detailed CRN models, which then are saved as an SBML files.[33] BioCRNpyler is written in python with a flexible object-oriented design, extensive documentation, and detailed examples which allow for easy model construction by modelers, customization and extension by developers, and rapid integration into data pipelines. As Figure 2 shows, un-

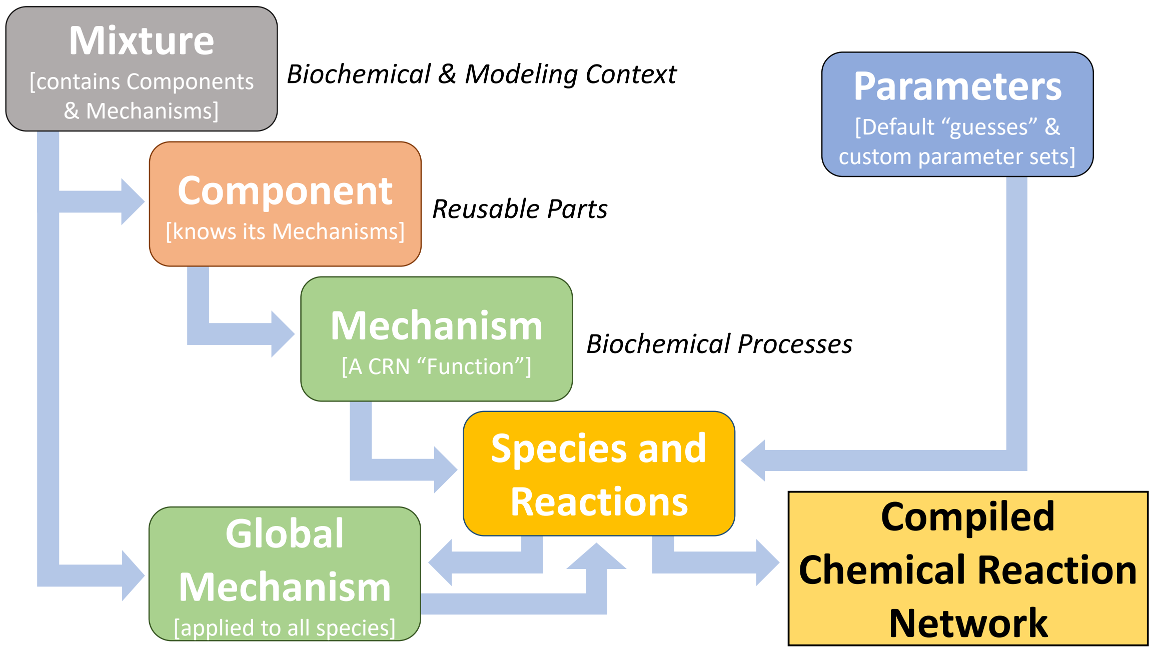

Figure 2: The hierarchical organization of python classes in the BioCRNpyler. Arrows represent direction of compilation: from high-level design specifications (`Components`) in a modeling context (`Mixtures`) and biochemical processes (`Mechanims`) to a CRN representation.

derlying BioCRNpyler is a comprehensive `ChemicalReactionNetwork` class allowing for the direct creation and manipulation of `Reactions` and the participating `Species` to represent molecular interactions at many levels of complexity. For example, an entire gene may be modeled as a single `Species` or as an `OrderedPolymerSpecies` with multiple binding specific sites.

BioCRNpyler also compiles `ChemicalReactionNetwork` objects from high-level specifications defined by modular `Components` combined together in a `Mixture` representing a biochemical context (e.g. cell lysate extract). Modeling assumptions and specific knowledge of biochemical processes are defined via `Mechanisms` which can be placed inside `Components` and `Mixtures`. This class structure allows for the biochemical parts (e.g. `Components`) to be reused to quickly produce numerous different architectures and implementations, such as those described in the motivating examples. These different architectures and implementations can further be tested in different contexts providing easily customizable levels of biochemical and modeling complexity represented by `Mixtures` and `Mechanisms`.

The `Mixture`, `Component`, and `Mechanism` classes are hierarchical. `Mixtures` represent biological context by containing `Components` to represent the biochemical environment and `Mechanisms` to represent the modeling detail. For example, the `TxTlExtract` subclass of `Mixture` represents bacterial cell extract and contains Ribosomes, RNA Polymerase, and RNAases `Components` as well as transcription, translation, and RNA-degredation `Mechanisms` (illustrated in the gray box of Figure 3). Additionally, `Components` can be added to a `Mixture` to produce a particular biochemical system of interest in a particular context. For example, a `DNAassembly` `Component` representing a piece of synthetic DNA encoding a circuit could be added to the `TxTlExtract`. During compilation `Components` represent biochemical function-

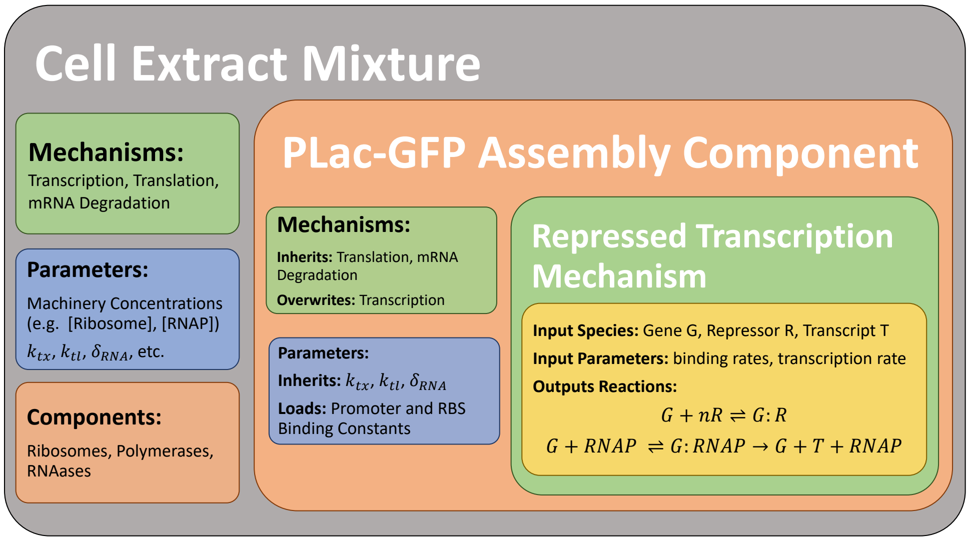
Cell Extract Mixture
Mechanisms: Transcription, Translation, mRNA Degradation
Parameters: Machinery Concentrations (e.g. [Ribosome], [RNAP]) $k_{tx}, k_{tl}, \delta_{RNA}$, etc.
Components: Ribosomes, Polymerases, RNAases
PLac-GFP Assembly Component
Mechanisms: Inherits: Translation, mRNA Degradation Overwrites: Transcription
Parameters: Inherits: $k_{tx}, k_{tl}, \delta_{RNA}$ Loads: Promoter and RBS Binding Constants
Repressed Transcription Mechanism
Input Species: Gene G, Repressor R, Transcript T
Input Parameters: binding rates, transcription rate
Outputs Reactions:
$G + nR \rightleftharpoons G{:}R$
$G + RNAP \rightleftharpoons G{:}RNAP \rightarrow G + T + RNAP$

Figure 3: A schematic of the high level specifications used by BioCRNpyler. A `Mixture` (gray) contains `Mechanisms` (green), `Parameters` (blue) and `Components` (orange). Each `Component` may also contain its own `Mechanisms` and `Parameters`. `Mechanism` are chemical reaction schemas (yellow) and represent specific models of biochemical processes in order to output `Species` and `Reactions` which are compiled into a CRN.

ality by calling `Mechanisms` to produce `Species` and `Reactions`. The `Components` class may use the `Mechanisms` in their `Mixture` or have their own `Mechanisms` to have more differentiated functionality. For example, a `RepressiblePromoter` (a subclass of `Component`) might rely on the `Mixture` for its translation `Mechanism` but use a custom transcription `Mechanism` to represent genetic regulation (illustrated in the orange box of Figure 3.). Finally, BioCRNpyler uses flexible `ParameterDatabases` contained in both `Mixtures` and `Components` to allow for rapid model prototyping using just a few default parameters, which can later customized for each `Component` and `Mechanism`.

3.1 Internal CRN Representation

Formally, a CRN is a set of species $S = \{S_i\}$ and reactions $R : \{I \xrightarrow{\rho(s;\theta)} O\}$ where I and O are multisets of species, ρ is the rate function or propensity, s is a vector of species' concentrations (or counts), and θ are rate parameters. Typically, CRNs are simulated using as ordinary differential equations (ODEs) and numerically integrated.[2] A stochastic semantics also allows CRNs to be simulated as continuous-time Markov chains.[34] Besides their prevalence in biological modeling, there is rich theoretical body of work related to CRNs from the mathematical[35], computer science[36], and physics communities.[37] Despite these theoretical foundations, many models are phenomenological in nature and lack mechanistic details of various biological processes. The challenge of constructing correct models is compounded by the difficulty in differentiating between correct and incorrect models based upon experimental data.[38–40]

BioCRNpyler is designed to compile CRNs that can be saved as SBML[16] for simulation with many different simulators. The CRN classes inside BioCRNpyler provide useful functionality so that users can easily modify CRNs produced via compilation, produce entire CRNs by hand or interface hand-produced CRNs with compiled CRNS. These functionalities include the classes to represent `Species` bound together as `ComplexSpecies`, lists of Species organized as `OrderedPolymerSpecies` and many more. Additionally, user-friendly printing functionality allows for the easy visualization of CRNs in multiple text formats or as reaction graphs formatted and drawn using Bokeh and ForceAtlas2.[41,42]

3.2 Mechanisms are Reaction Schemas

When modeling biological systems, modelers frequently make use of massaction CRN kinetics which ensure that parameters and states have clear underlying mechanistic meanings. However, for the design of synthetic biological circuits and analysis using experimental data, phenomenological or reduced-order models are commonly utilized as well.[2] Empirical phenomenological models have been proven to be successful in predicting and analyzing complex circuit behavior using simple models with only a few lumped parameters.[43–45] Bridging the connections between the different modeling abstractions is a challenging research problem. This has been explored in the literature using various approaches such as by direct mathematical comparison of mechanistic and phenomenological models[46–48] or by studying particular examples of reduced models.[2] BioCRNpyler provides a computational approach using reaction schemas to easily change the mechanisms used in compilation from massaction to coarse-grained at various level of complexity.

Reaction schemas refer to BioCRNpyler's generalization of switching between different mechanistic models: a single process can be modeled using multiple underlying motifs to generate a class of models. `Mechanisms` are the BioCRNpyler objects responsible for defining reaction schemas. In other words, various levels of abstractions and model reductions can all be represented easily by using built-in and custom `Mechanisms` in BioCRNpyler. For example, to model the process of transcription (as shown in Figure 4), BioCRNpyler allows the use of various phenomenological and massaction kinetic models by simply changing the choice of reaction schema. Notably, this provides a unique capability to quickly compare system models across various levels of abstraction enabling a more nuanced approach to circuit design and exploring system parameter regimes.

The ability to generate chemical `Species` and `Reactions` via customizable `Mechanisms` is one of the key features making BioCRNpyler distinct from other frameworks. Hierarchical SBML and supporting software provide [49] a noteable exception—however BioCRNpyler contains library of reusable chemical reaction motifs, while Hierarchical SBML is a standard for

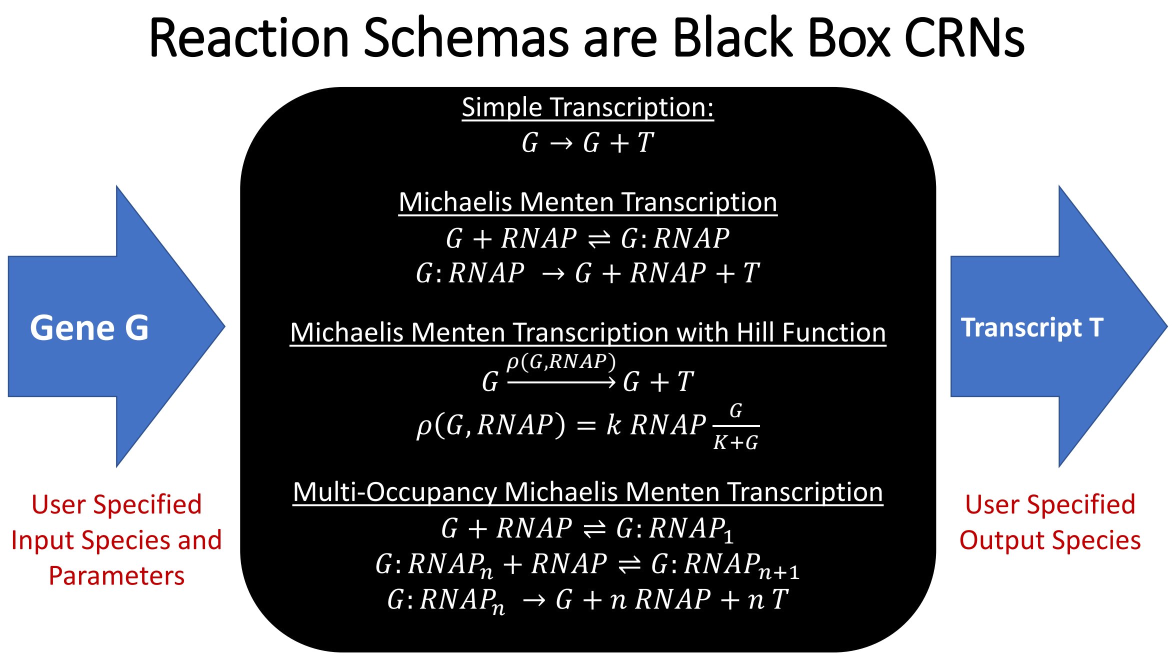
Reaction Schemas are Black Box CRNs
Gene G
User Specified Input Species and Parameters
Simple Transcription:
$G \rightarrow G + T$
Michaelis Menten Transcription
$G + RNAP \rightleftharpoons G{:}RNAP$
$G{:}RNAP \rightarrow G + RNAP + T$
Michaelis Menten Transcription with Hill Function
$G \xrightarrow{\rho(G,RNAP)} G + T$
$\rho(G,RNAP) = k\, RNAP \frac{G}{K+G}$
Multi-Occupancy Michaelis Menten Transcription
$G + RNAP \rightleftharpoons G{:}RNAP_1$
$G{:}RNAP_n + RNAP \rightleftharpoons G{:}RNAP_{n+1}$
$G{:}RNAP_n \rightarrow G + n\, RNAP + n\, T$
Transcript T
User Specified Output Species

Figure 4: Mechanisms (Reaction Schemas) representing transcription.

describing embedded CRN models. Formally, reaction schemas are functions that produce CRN species and reactions: $f : (S', \theta) \rightarrow (S, R)$. Here the inputs S' are chemical species and θ are rate constants. The outputs $S \supseteq S'$ is an increased set of species and R is a set of reactions. Figure 4. gives different examples of a reaction schema's representing transcription. This functionality allows modelers to generate CRNs at different levels of complexity and reuse CRN motifs for some `Components` while customizing Mechanisms for others. Importantly, BioCRNpyler contains a large and growing library of existing `Mechanisms` extensively documented via examples making them easy to use and repurpose without extensive coding. Developing custom `Mechanisms` is also as easy as making a subclass of `Mechanism` and defining three functions to produce the desired CRN:

```python
class CustomMechanism(Mechanism):
    def __init__(self, ... ):
        Mechanism.__init__(self, name = "name", mechanism_type = "type", **kwargs)
        # python code to set internal variables


    def update_species(self, ... ):
        # python code to create Species objects
        return species_list

    def update_reactions(self, ... ):
        # python code to create Reaction objects
        return reaction_list
```

Internally, each `Mechanism` class has a type (e.g. transcription) which defines the input and output species it requires. `GlobalMechanisms` are a special subclass of `Mechanism` called at the end of compilation to represent processes which act on large subsets of CRN Species such as dilution in cellular models.

3.3 Components Represent Functionality

In BioCRNpyler, `Components` are biochemical parts or motifs, such as promoters, enzymes and chemical complexes. `Components` represent biomolecular functionality; a promoter enables transcription, enzymes perform catalysis, and chemical complexes must bind together. `Components` express their functionality by calling particular `Mechanism` types during compilation. Importantly, `Components` are not the same as CRN `Species`; one `Species` might be represented by multiple `Components` and a `Component` might produce multiple `Species`! `Components` are very flexible and can behave differently in different contexts or behave context-independently. For dynamic-context behavior, define `Components` to use mechanisms and parameters provided by the `Mixture`. For context-independent behavior, define `Components` to have their own internal `Mechanisms` and (`ParameterDatabases`). The BioCRNpyler library includes many `Component` subclasses to model enzymes (`Enzyme`), chemical complexes (`ChemicalComplexes`) formed by molecular binding, Promoters (`Promoter`), Ribosome Binding Sites (`RBS`), complex genetic architectures (such as `DNA_construct` illustrated in Figure 5), and more. It is also very easy to make custom `Components`: simply subclass `Component` and define three functions:

```python
class CustomComponent(Component):
    def __init__(self, ... ):
        Component.__init__(self, ... , **kwargs)
        # python code to set internal variables

    def update_species(self):
        # python code calls mechanism.update_species( ... )
        return species_list

    def update_reactions(self):
        # python code calls mechanism.update_reaction( ... )
        return reaction_list
```

3.4 Mixtures Represent Context

`Mixtures` are collections of default `Components`, default `Mechanisms`, and user-added `Components`. `Mixtures` can represent chemical context (e.g. cell extract vs. *in vivo*), as well as modeling resolution (e.g. what level of detail to model transcription or translation at) by containing different internal `Components` and `Mechanisms`. `Mixtures` also control CRN compilation by requesting `Species` and `Reactions` for each of their `Components`. After receiving all these `Species` and `Reactions`, Mixtures then apply `GlobalMechanisms` which act on all the Species produced by `Components`. BioCRNpyler comes with a variety of `Mixtures` to represent cell-extracts and cell-like sysetems with dilution with multiple levels of modeling complexity. Making custom `Mixtures` is also easy - it can be done via simple scripts by adding `Components` and `Mechanisms` to a `Mixture` object:

```python
MyMixture = Mixture("customized mixture",
                  components = [List Components],
                  mechanisms = dict("mechanism_type":Mechanism))
```

The `Mixture` class can also be easily subclassed by rewriting the constructor:

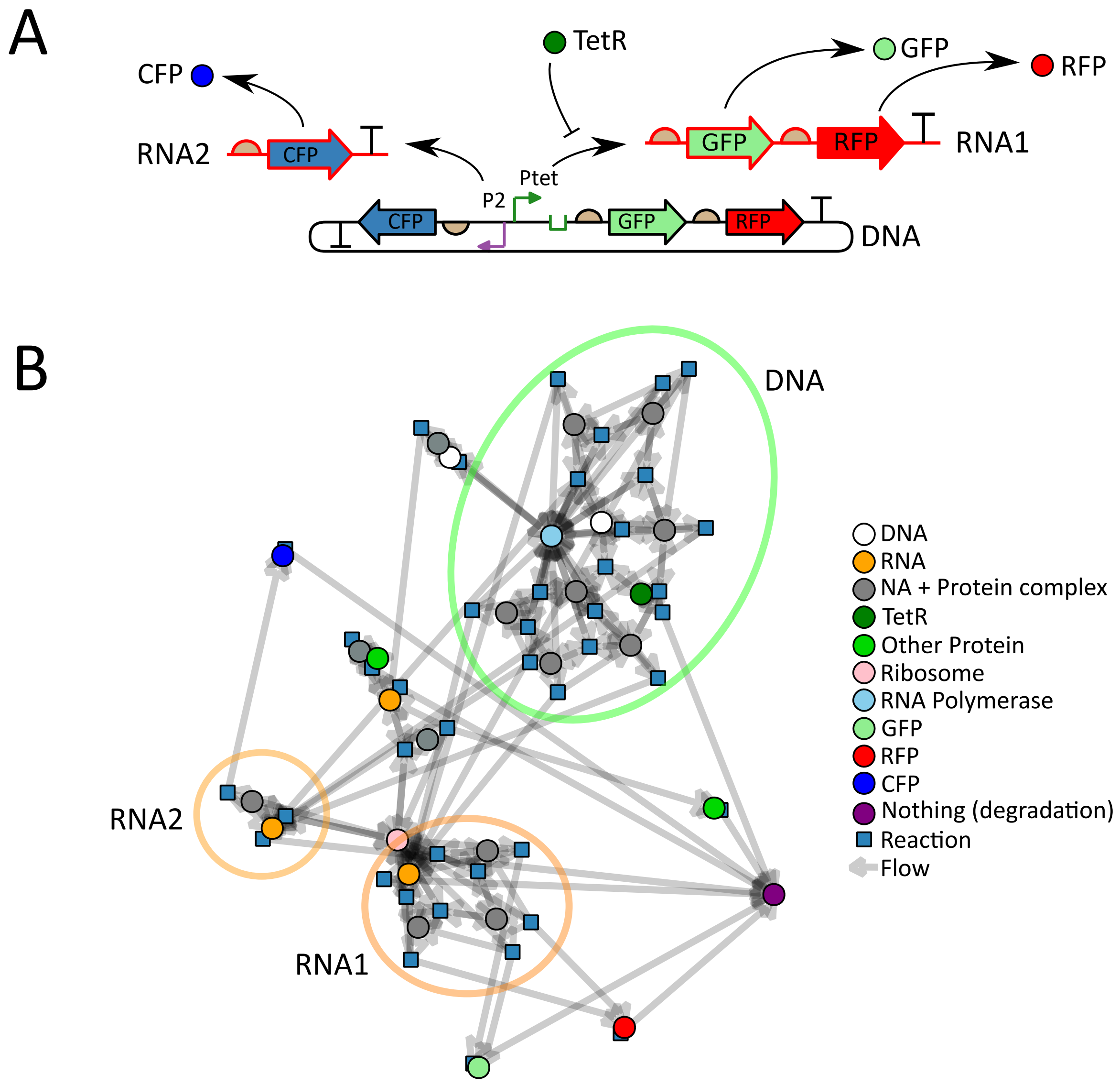

Figure 5: Using the `DNA_construct` class, `DNA_parts` (a subclass of `Component`) can be arranged in the same order as they would be in a DNA molecule and compiled into a CRN. **A.** A complicated `DNA_construct` contains two transcriptional units in different directions, one of which makes a bicistronic mRNA that makes GFP and RFP, and can be repressed by TetR. **B.** A directed graph representation of the compiled CRN from the `DNA_construct`. `Species` represented by circles participate in `Reactions` represented by squares. Circled groups of `Species` and `Reactions` involve the `DNA_construct` labeled "DNA", or the `RNA_constructs` labeled "RNA1" or "RNA2". The `DNA_construct` class creates all the necessary `Species` and `Reactions` to simulate transcription and translation from linear and circular DNA, taking into account the compositional context of `DNA_parts`.

```
class CustomMixture(Mixture):
    def __init__(self, ... , **kwargs):
        #python code to set up internal variables,
        # create Components, and default Mechanisms
        Mixture.__init__(self, mechanisms = dict("mechanism_type":Mechanism),
                                components = [List of Components], **kwargs)
```

3.5 Flexible Parameter Databases

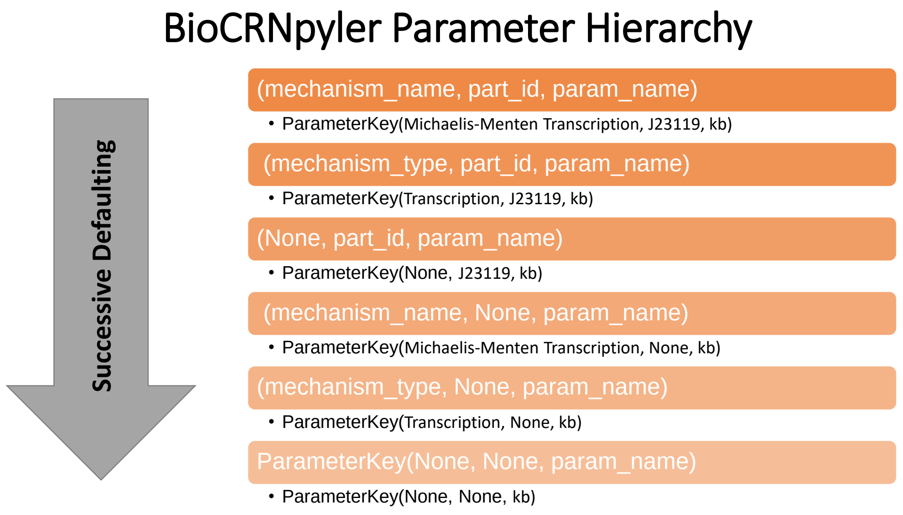
BioCRNpyler Parameter Hierarchy
Successive Defaulting
(mechanism_name, part_id, param_name)
• ParameterKey(Michaelis-Menten Transcription, J23119, kb)
(mechanism_type, part_id, param_name)
• ParameterKey(Transcription, J23119, kb)
(None, part_id, param_name)
• ParameterKey(None, J23119, kb)
(mechanism_name, None, param_name)
• ParameterKey(Michaelis-Menten Transcription, None, kb)
(mechanism_type, None, param_name)
• ParameterKey(Transcription, None, kb)
ParameterKey(None, None, param_name)
• ParameterKey(None, None, kb)

Figure 6: BioCRNpyler Parameter Defaulting Heirarchy. If a specific `ParameterKey` (orange boxes) cannot be found, the `ParameterDatabase` automatically defaults to other `ParameterKeys`. This allows for parameter sharing and rapid construction of complex models from relatively relatively few non-specific (e.g. lower in the hierarchy) parameters.

Developing models is a process that involves defining then parameterizing interactions. Often, at the early stage of model construction, exact parameter values will be unavailable. BioCRNpyler has a sophisticated parameter framework which allows for the software to search user-populated `ParameterDatabases` for the parameter that closest matches a specific `Mechanism`, `Component`, and parameter name. This allows for models to be rapidly constructed and simulated with "ball-park" parameters and then later refined with specific parameters derived from literature or experiments later. This framework also makes it easy to incorporate diverse parameter files together and share parameters between many chemical reactions.

4 Building an Open-source Community

BioCRNpyler aims to be a piece of open-source community driven software that is easily accessible to biologists and bioengineers with varying levels of programming experience as well as easily customizable by computational biologists and more advanced developers. Towards these ends, the software package is available via GitHub and PyPi, requires very minimal software dependencies, contains extensive examples and documentation in the form of interactive Jupyter notebooks, and automated testing to ensure stability. Furthermore this software has been extensively tested via inclusion in a bio modeling course and bootcamp with dozens of users ranging from college freshmen and sophomores with minimal coding experience to advanced computational biologists. Developing new software functionality is also a simple process documented on the GitHub contributions page.

4.1 Integrated Testing

BioCRNpyler uses Travis-CI[50] and Codecov[51] to automate testing on GitHub. Whenever the software is updated, a suite of tests is run including extensive unit tests and functional testing of tutorial and documentation notebooks. Automated testing unit testing ensures that changes to the core BioCRNpyler code preserve functionality of the package. The integration of Jupyter notebooks into testing allows users to easily define new functionality for the software and document that functionality with detailed explanations which are simultaneously tests cases.

4.2 Documentation and Tutorials

The BioCRNpyler GitHub page contains over a dozen tutorial Jupyter notebooks and presentations explaining everything from the fundamental features of the code to specialized functionality for advanced models to how to add to the BioCRNpyler code-base. This documentation has been used successful in multiple academic courses and is guaranteed to be up-to-date and functional due to automatic testing.

5 Future Directions

BioCRNpyler is an ongoing effort which will grow and change with the needs of its community and extending this community via outreach, documentation, and an ever expanding suite of functionalities is central to the goals of this project. In the immediate future, we aim to add SBOL compatibility so BioCRNpyler can automatically compile models of circuits designed and built in a laboratory setting. This approach will be generalization and extension of[52] - in particular due to the modular BioCRNpyler compilation process, it will be possible have programatic control over the SBML model produced from BioCRNpyler. We also plan on extending the library to include more realistic Mixtures (particularly experimentally validated models of circuits in *E. coli* and in cell extracts), advanced Components to model such as integrases and RNA-splicing which require a more dynamic compilation process, and additional mechanisms to model the reactions underlying these systems.

6 Supplemental: Code for Examples

This section provides code from the examples Figure 1. The first three models are idealized in the sense that they are represented by hill functions and include no cellular machinery such as ribosomes or polymerases. Producing these models in BioCRNpyler is easy and just requires the reuse of a few parts: `DNAassembly` represents a simple transcriptional unit with a promoter, transcript and optionally an ribosome binding site (RBS) and protein product. `RepressiblePromoter` creates a promoter modeled by a hill function. `Species` creates CRN species used in the models. Notice that only `Species` which are shared between different `Components` need created by hand - BioCRNpyler takes care of the rest. Finally, everything is added together into a subclass of `Mixture` and compiled into a `ChemicalReactionNetwork`. The second three models build off the general architectures of the first three but add in more complicated context and implementation details. Instead of using `ExpressionDilutionMixture` `SimpleTxTlDilutionMixture`, these models use the considerably more complex `TxTlDilutionMixture` which includes molecular machinery such as RNAP, ribosomes, RNAases, and background cellular processes. Additional implementation details in the form of `Components` and `Mechanisms` are also added to these models.

6.1 Inducible Repression

Here a repressor is constituitively produces from a `DNAassembly`. This repressor is then linked to a `RepressiblePromoter` which models repression using a hill function.

```
#Models a piece of DNA that constiuitively produces the species R
repressor = Species("R")
const_rep = DNAassembly(name="const_rep", promoter="medium", rbs="medium",
    protein=repressor)
#R represses RepressiblePromoter which is placed into another DNAassembly reporter
prom = RepressiblePromoter(name="pR", repressor=repressor)
reporter = DNAassembly(name = "Reporter", promoter=prom, rbs="strong", initial_conc=1)
#ExpressionDilutionMixture models gene expression without transcription/translation
mixture = ExpressionDilutionMixture(components=[reporter, const_rep],
    parameter_file="params.txt")
CRN = mixture.compile_crn()
```

6.2 Toggle Switch

In the following example, a toggle switch is created by connecting two instances of `RepressiblePromoter` together. Notice that string names passed to promoter and rbs are used to help find parameters. BioCRNpyler comes with many default parameters to enable rapid model prototyping.

```
#Creates A and is repressed by B
repA = Species("A")
promA = RepressiblePromoter(name="pA", repressor=repB)
assemblyA = DNAassembly(name="A", promoter=promA, rbs="medium", protein=repA,
    initial_conc=1)
#Creates B and is repressed by A
repB = Species("B")
promB = RepressiblePromoter(name="pB", repressor=repA)
assemblyB = DNAassembly(name="B", promoter=promB, rbs="medium", protein=repB,
    initial_conc=1)
#SimpleTxTlDilutionMixture includes transcription and translation but no machinery
mixture = SimpleTxTlDilutionMixture(components=[assemblyA, assemblyB], parameter_file=
    "params.txt")
CRN = mixture.compile_crn()
```

6.3 Repressilator

The code to create a 3-node repession oscillator is really just adding one more unit and rewiring the toggle switch example.

```
#Create Repressors
repA = Species("A")
repB = Species("B")
repC = Species("C")
#Create Promoters
promA = RepressiblePromoter(name = "pA", repressor = repC)
promB = RepressiblePromoter(name = "pB", repressor = repA)
```

```
promC = RepressiblePromoter(name = "pC", repressor = repB)
#Create DNAassemblies
assemblyA = DNAassembly(name = "A", promoter = promA, rbs = "medium", protein = repA,
    initial_conc = 1)
assemblyB = DNAassembly(name = "B", promoter = promB, rbs = "medium", protein = repB,
    initial_conc = 1)
assemblyC = DNAassembly(name = "C", promoter = promC, rbs = "medium", protein = repC,
    initial_conc = 1)
#Place it all in a Mixture & Compile
mixture = SimpleTxTlDilutionMixture(components = [assemblyA, assemblyB, assemblyC],
    parameter_file = "params.txt")
crn = mixture.compile_crn()
```

6.4 Cas9 Repressor and Guide RNA Coexpressed with Reporter

Modeling a dCas9-guideRNA repressor in bioCRNpyler requires that the dCAs9 and guide RNA know to bind together. This is accomplished via the `Component` subclass `ChemicalComplex` which models binding between multiple species. The resulting dCas9-guideRNA `ComplexSpecies` is used as a repressor.

```
#Only one dCas9-guideRNA complex binds to the promoter at once
params = {
    ("negativehill_transcription", None, "n"):1
}

#Create guide RNA and dCas9 Species
guide = Species("guide", material_type = "rna")
dcas = Species("dCas9")
#These species will bind together by placing them in the ChemicalComplex Component
#the attribute "notdegradable" ensures that RNAases do not degrade guide RNAs bound to
    dCas9.
repressor = ChemicalComplex([dcas, guide], attributes = ["notdegradable"])
```

```
reporter = Species("reporter")
#Constuitive Assemblies to produce dCas9 and the guideNRA
assembly_dcas = DNAassembly(name = "dcas", promoter = "medium", rbs = "medium", protein
    = dcas)
assembly_guide = DNAassembly(name = "guide", promoter = "strong", rbs = None, transcript
    = guide)
#Create a repessible promoter
pReg = RepressiblePromoter(name = "pA", repressor = repressor, parameters = params)
assembly_rep = DNAassembly(name = "reporter", promoter = pReg, rbs = "strong",
    initial_conc = 1, protein = reporter)
# Place the Components in a Mixture
extract = TxTlDilutionMixture("e coli", components = [assembly_rep, assembly_dcas,
    assembly_guide, repressor], parameter_file = "params.txt")
#Compile the CRN
crn = extract.compile_crn()
```

6.5 Targeted RNAase Toggle Switch

The targeted RNAase toggle switch model is a hypothetic model similar to a normal toggle switch but with regulation at the RNA level instead of the transcriptional level. This is accomplished by creating two constitutively expressed RNAases (which are `ChemicalComplexes` made up of two subunits) and adding custom `Mechanisms` to the `Mixture` modeling the degradation of any species with the attribute "tagA" and "tagB" by RNAase A and RNAase B, respectively.

```
#Create an RNA species with degradation tag sequence tagB
TA = Species("A", attributes = ["tagB"], material_type = "rna")
#Create homodimer subunit A
A = Species("A", material_type = "protein")
#RNAase A is a homodimer made up of two identical subunits
rnaaseA = ChemicalComplex([A]*2)
#create a DNAassembly that produces A
```

```
assemblyA = DNAassembly(name = "A", promoter = "strong", transcript = TA, rbs =
    "medium", protein = A, initial_conc = 1)

#Same as above but for Species B
TB = Species("B", attributes = ["tagA"], material_type = "rna")
B = Species("B", material_type = "protein")
rnaaseB = ChemicalComplex([B]*2)
assemblyB = DNAassembly(name = "B", promoter = "strong", transcript = TB, rbs =
    "medium", protein = B, initial_conc = 1)
#add all the Components to a Mixture
mixture = TxTlDilutionMixture("e coli", components = [assemblyA, assemblyB, rnaaseA,
    rnaaseB], parameter_file = "default_parameters.txt")
#Create a Deg_Tagged_Degredation Mechanisms which tells rnaaseA to degrade any species
    with attribute "tagA"
mixture.add_mechanism(Deg_Tagged_Degredation(mechanism_type = "tagA_degredation",
    deg_tag = "tagA", protease = rnaaseA.get_species()))
create a Deg_Tagged_Degredation Mechanisms which tells rnaaseB to degrade any species
    with attribute "tagB"
mixture.add_mechanism(Deg_Tagged_Degredation(mechanism_type = "tagB_degredation",
    deg_tag = "tagB", protease = rnaaseB.get_species()))
#Compile the CRN
CRN = mixture.compile_crn()
```

6.6 Multiple Ribosome Occupancy Repressilator Dynamics

Simulating multiple-ribosome occupancy in the Repressilator mostly reuses the code from Section 6.3 with the main addition of a new `Mechanism` to model transcription being placed into the `Mixture`.

```
#Create Repressors
repA = Species("A")
repB = Species("B")
```

```
repC = Species("C")
#Create Promoters
promA = RepressiblePromoter(name = "pA", repressor = repC)
promB = RepressiblePromoter(name = "pB", repressor = repA)
promC = RepressiblePromoter(name = "pC", repressor = repB)
#Create DNAassemblies
assemblyA = DNAassembly(name = "A", promoter = promA, rbs = "strong", protein = repA,
    initial_conc = 1)
assemblyB = DNAassembly(name = "B", promoter = promB, rbs = "strong", protein = repB,
    initial_conc = 1)
assemblyC = DNAassembly(name = "C", promoter = promC, rbs = "strong", protein = repC,
    initial_conc = 1)
#Extra parameters for the Multi_tx Mechanism
extra_params = {"max_occ":10, ("multi_tx", None, "k_iso"):50, ("multi_tl", None,
    "k_iso"):50, "cooperativity":2}
#Add Everything to a Mixture
mixture = TxTlDilutionMixture("e coli", components = [assemblyA, assemblyB, assemblyC],
                           parameter_file = "default_parameters.txt", parameters =
                               extra_params, overwrite_parameters = True)
#Add the multi_t translation mechanism to the mixture, overwriting the old one.
mixture.add_mechanism(multi_tl(name = "multi_tl", ribosome =
    mixture.ribosome.get_species()), overwrite = True)
#Compile the CRN
crn = mixture.compile_crn()
```

7 Supplemental: Tables of Features

This section lists many of the different `Mixture`, `Component` and `Mechanism` classes available in BioCRNpyler. For more details about these classes and examples using many of them, check out the examples folder on Github.

7.1 Mixtures

Mixture Name	**Description**
ExpressionExtract	A model for gene expression without machinery such as ribosomes, polymerases, etc. Here transcription and translation are lumped into one reaction: expression.
SimpleTxTlExtract	A model for transcription and translation in a cell-free extract without machinery such as ribosomes, polymerases, etc. RNA is degraded via a global mechanism.
TxTlExtract	A model for transcription and translation in a cell-free extract with machinery for ribosomes, polymerases, and endonucleases action. This model does not include any energy buffer.
ExpressionDilutionMixture	A model for *in-vivo* gene expression without any machinery such as ribosomes, polymerases, etc. Transcription and translation are lumped into one reaction and a global mechanism is used to dilute all non-DNA species.
SimpleTxTlDilutionMixture	Mixture with continuous dilution for non-DNA species. Transcription (TX) and Translation (TL) are both modeled as catalytic with no cellular machinery. mRNA is also degraded via a separate reaction to represent endonucleases.
TxTlDilutionMixture	Transcription and translation with ribosomes, polymerases, and endonucleases labelled as cellular machinery. Also includes a background load which represents innate loading effects in the cell. Effects of loading on cell growth are not modeled. It has global dilution for non-DNA and non-machinery species. This model does not include any energy.

7.2 Components

Component Type	Component Name	Description
Promoter	Promoter	Constitutive $\sigma70$ promoter
Promoter	RegulatedPromoter	Repressible or activatable promoter such as P_{tet}
Promoter	ActivatablePromoter	Activatable promoter using a positive Hill function
Promoter	RepressiblePromoter	Repressible promoter using a negative Hill function
Promoter	CombinatorialPromoter	Flexible promoter mechanism allowing various transcription factor binding configurations to allow or prevent transcription
Ribosome Binding Site	RBS	Simple RBS using a translation mechanism
Coding Sequence	CDS	Protein coding part used for `DNA_construct`. Doesn't affect CRN
Terminator	Terminator	Transcriptional terminator used for `DNA_construct`. Doesn't affect CRN
DNA	DNA	Basic component that represents a DNA sequence
DNA	DNAassembly	A relatively simple DNA sequence containing one promoter, RBS, and a product
DNA	DNA_construct	A more complex DNA sequence that can have any number of `Components` in any order
RNA	RNA	Basic component that represents an RNA sequence
RNA	RNA_construct	A more complex RNA sequence that can have any number of `Components` in any order. Usually automatically generated by `DNA_construct`
Protein	Protein	Basic component that represents a protein
Chemical Complex	ChemicalComplex	A complex that represents the combination of several `Species`. Takes care of binding and unbinding reactions needed to form the complex
Enzyme	Enzyme	An enzyme that operates on a substrate to produce a product
Enzyme	MultiEnzyme	An enzyme that operates on a list of substrates to produce a list of products

7.3 Mechanisms

Mechanisms Type	**Mechanism Name**	**Description**
binding	Reversible_Bimolecular_Binding	$S_1 + S_2 \leftrightharpoons (S_1 : S_2)$
cooperative_binding	One_Step_Cooperative_Binding	$n\,S_1 + S_2 \leftrightharpoons (nS_1 : S_2)$
cooperative_binding	Two_Step_Cooperative_Binding	$n\,S_1 \leftrightharpoons (nS_1)$, $(nS_1) + S_2 \leftrightharpoons (nS_1 : S_2)$
cooperative_binding	Combinatorial_Cooperative_Binding	Allows a set of species S_i and cooperativities n_i to bind to a target T in any order to form $(n_1 S_1 : \ldots : n_k S_k : T$ along with all combinatorial intermediaries.
binding	One_Step_Binding	$S1 + S2 \ldots SN \leftrightharpoons S1 : S2 : \ldots : SN$
catalysis	BasicCatalysis	$S + C \rightarrow P + C$
catalysis	BasicProduction	$C \rightarrow P + C$
catalysis	MichaelisMenten	$Sub + Enz \leftrightharpoons Sub : Enz \rightarrow Enz + Prod$
catalysis	MichaelisMentenReversible	$Sub + Enz \leftrightharpoons Sub : Enz \leftrightharpoons Enz : Prod \leftrightharpoons Enz + Prod$
copy	MichaelisMentenCopy	$Sub + Enz \leftrightharpoons Sub : Enz \rightarrow Sub + Enz + Prod$
transcription	OneStepGeneExpression	$G \rightarrow G + P$
transcription	SimpleTranscription	$G \rightarrow G + T$
translation	SimpleTranslation	$T \rightarrow T + P$
transcription	PositiveHillTranscription	$G \rightarrow [r]G + P \;\; r = kG(R^n)/(K + R^n)$
transcription	NegativeHillTranscription	$G \rightarrow [r]G + P \;\; r = kG/(K + R^n)$
transcription	Transcription_MM	$G + RNAP \leftrightharpoons G : RNAP \rightarrow G + RNAP + mRNA$
translation	Translation_MM	$mRNA + Rib \leftrightharpoons mRNA : Rib \rightarrow mRNA + Rib + Protein$
transcription	multi_tx	$DNA : RNAp_n + RNAp \leftrightharpoons DNA : RNAp_n^{closed} \rightarrow DNA : RNAp_{n+1} DNA : RNAp_n \rightarrow DNA : RNAp_0 + nRNAp + nmRNA DNA : RNAp_n^{closed} \rightarrow DNA : RNAp_0^{closed} + nRNAp + nmRNA$ for $n = \{0, \max_{occ}\}$
translation	multi_tl	$mRNA : RBZ_n + RBZ \leftrightharpoons mRNA : RBZ_n^{closed} \rightarrow mRNA : RBZ_{n+1} mRNA : RBZ_n \rightarrow mRNA : RBZ_0 + nRBZ + nProtein mRNA : RBZ_n^{closed} \rightarrow mRNA : RBZ_0^{closed} + nRBZ + nProtein$ for $n = \{0, \max_{occ}\}$
dilution	Dilution	$s \rightarrow \emptyset$
rna_degredation_mm	Degredation_mRNA_MM	$T + Nuclease \leftrightharpoons T : Nuclease \rightarrow Nuclease$. Global mechanism effects all RNA species T. ComplexSpecies containing RNA species are broken apart via the reaction $T : X + Nuclease \leftrightharpoons T : X : Nuclease \rightarrow X + Nuclease$. for any X.
degredation	Deg_Tagged_Degredation	$X + Protease \leftrightharpoons X : Protease \rightarrow Protease$. Here X is any Species with the deg_tag attribute passed into the constructor of this GlobalMechanism.

Acknowledgement

We would like to thank the Caltech BE240 class and the Murray Biocircuits lab for extensive testing of this software and discussions of relevant models, library of parts, and parameters. In particular, we would like to thank Matthieu Kratz, Liana Merk, and Ankita Roychoudhury for contributing to the software library. The authors W.P. and A.P. are supported by NSF grant number: CBET-1903477. A.P. is also supported by the Defense Advanced Research Projects Agency (Agreement HR0011-17-2-0008). The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

(1) Alon, U. *An introduction to systems biology: design principles of biological circuits*; CRC press, 2019.

(2) Vecchio, D. D.; Murray, R. M. *Biomolecular Feedback Systems*; Princton University Press, 2014.

(3) The MathWorks, Inc, MATLAB Simbiology Toolbox.

(4) Somogyi, E. T., et al. libRoadRunner: a high performance SBML simulation and analysis library. *Bioinformatics* **2015**, *31*, 3315–3321.

(5) Le Novere, N., et al. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic acids research* **2006**, *34*, D689–D691.

(6) Hoops, S., et al. COPASI-a complex pathway simulator. *Bioinformatics* **2006**, *22*, 3067–3074.

(7) Choi, K., et al. Tellurium: an extensible python-based modeling environment for systems and synthetic biology. *Biosystems* **2018**, *171*, 74–79.

(8) Benner, S. A.; Sismour, A. M. Synthetic biology. *Nature Reviews Genetics* **2005**, *6*, 533–543.

(9) Nielsen, A. A. K.; Der, B. S.; Shin, J.; Vaidyanathan, P.; Paralanov, V.; Strychalski, E. A.; Ross, D.; Densmore, D.; Voigt, C. A. Genetic circuit design automation. *Science* **2016**, *352*, aac7341–aac7341.

(10) Guiziou, S.; Pérution-Kihli, G.; Ulliana, F.; Leclère, M.; Bonnet, J. Exploring the design space of recombinase logic circuits. *bioRxiv* **2019**,

(11) Soloveichik, D.; Seelig, G.; Winfree, E. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences* **2010**, *107*, 5393–5398.

(12) Qian, L.; Winfree, E. Scaling up digital circuit computation with DNA strand displacement cascades. *Science* **2011**, *332*, 1196–1201.

(13) Badelt, S.; Grun, C.; Sarma, K. V.; Wolfe, B.; Shin, S. W.; Winfree, E. A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *Journal of the Royal Society Interface* **2020**, *17*, 20190866.

(14) Vasić, M.; Soloveichik, D.; Khurshid, S. CRN++: Molecular programming language. *Natural Computing* **2020**, 1–17.

(15) Spaccasassi, C.; Lakin, M. R.; Phillips, A. A logic programming language for computational nucleic acid devices. *ACS synthetic biology* **2018**, *8*, 1530–1547.

(16) Hucka, M.; Finney, A.; Sauro, H. M.; Bolouri, H.; Doyle, J. C.; Kitano, H.; Arkin, A. P.; Bornstein, B. J.; Bray, D.; Cornish-Bowden, A., et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **2003**, *19*, 524–531.

(17) Galdzicki, M.; Clancy, K. P.; Oberortner, E.; Pocock, M.; Quinn, J. Y.; Rodriguez, C. A.; Roehner, N.; Wilson, M. L.; Adam, L.; Anderson, J. C., et al. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology* **2014**, *32*, 545–550.

(18) Myers, C. J., et al. iBioSim: a tool for the analysis and design of genetic circuits. *Bioinformatics* **2009**, *25*, 2848–2849.

(19) Watanabe, L.; Nguyen, T.; Zhang, M.; Zundel, Z.; Zhang, Z.; Madsen, C.; Roehner, N.; Myers, C. iBioSim 3: a tool for model-based genetic circuit design. *ACS synthetic biology* **2018**, *8*, 1560–1563.

(20) Harris, L. A., et al. BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics* **2016**, *32*, 3366–3368.

(21) Lopez, C. F.; Muhlich, J. L.; Bachman, J. A.; Sorger, P. K. Programming biological models in Python using PySB. *Molecular systems biology* **2013**, *9*, 646.

(22) Tuza, Z. A., et al. An in silico modeling toolbox for rapid prototyping of circuits in a biomolecular "breadboard" system. 52nd IEEE Conference on Decision and Control. 2013; pp 1404–1410.

(23) BioCRNpyler. `https://github.com/BuildACell/BioCRNpyler`.

(24) Swaminathan, A., et al. Fast and flexible simulation and parameter estimation for synthetic biology using bioscrape. *bioRxiv* **2019**, 121152.

(25) Der, B. S.; Glassey, E.; Bartley, B. A.; Enghuus, C.; Goodman, D. B.; Gordon, D. B.; Voigt, C. A.; Gorochowski, T. E. DNAplotlib: programmable visualization of genetic designs and associated data. *ACS synthetic biology* **2017**, *6*, 1115–1119.

(26) Gardner, T. S.; Cantor, C. R.; Collins, J. J. Construction of a genetic toggle switch in Escherichia coli. *Nature* **2000**, *403*, 339–342.

(27) Elowitz, M. B., et al. A synthetic oscillatory network of transcriptional regulators. *Nature* **2000**, *403*, 335–338.

(28) Cress, B. F.; Toparlak, O. D.; Guleria, S.; Lebovich, M.; Stieglitz, J. T.; Englaender, J. A.; Jones, J. A.; Linhardt, R. J.; Koffas, M. A. CRISPathBrick: modular combinatorial assembly of type II-A CRISPR arrays for dCas9-mediated multiplex transcriptional repression in E. coli. *ACS synthetic biology* **2015**, *4*, 987–1000.

(29) Jayanthi, S.; Nilgiriwala, K. S.; Del Vecchio, D. Retroactivity controls the temporal dynamics of gene transcription. *ACS synthetic biology* **2013**, *2*, 431–441.

(30) Strutt, S. C.; Torrez, R. M.; Kaya, E.; Negrete, O. A.; Doudna, J. A. RNA-dependent RNA targeting by CRISPR-Cas9. *elife* **2018**, *7*, e32724.

(31) Dang, D. T.; Phan, A. T. Development of a ribonuclease containing a G4-specific binding motif for programmable RNA cleavage. *Scientific reports* **2019**, *9*, 1–7.

(32) Yan, X.; Hoek, T. A.; Vale, R. D.; Tanenbaum, M. E. Dynamics of translation of single mRNA molecules in vivo. *Cell* **2016**, *165*, 976–989.

(33) Hucka, M., et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **2003**, *19*, 524–531.

(34) Gillespie, D. T. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.* **2007**, *58*, 35–55.

(35) Gunawardena, J. Chemical reaction network theory for in-silico biologists. *Notes available for download at http://vcp. med. harvard. edu/papers/crnt. pdf* **2003**,

(36) Soloveichik, D.; Cook, M.; Winfree, E.; Bruck, J. Computation with finite stochastic chemical reaction networks. *natural computing* **2008**, *7*, 615–633.

(37) Schmiedl, T.; Seifert, U. Stochastic thermodynamics of chemical reaction networks. *The Journal of chemical physics* **2007**, *126*, 044101.

(38) Morrison, M. J.; Razo-Mejia, M.; Phillips, R. Reconciling Kinetic and Equilibrium Models of Bacterial Transcription. *arXiv preprint arXiv:2006.07772* **2020**,

(39) Cinquemani, E. Identifiability and reconstruction of biochemical reaction networks from population snapshot data. *Processes* **2018**, *6*, 136.

(40) Hsiao, V.; Swaminathan, A.; Murray, R. M. Control theory for synthetic biology: recent advances in system characterization, control design, and controller implementation for synthetic biology. *IEEE Control Systems Magazine* **2018**, *38*, 32–62.

(41) Bokeh Development Team, Bokeh: Python library for interactive visualization. 2020.

(42) Jacomy, M.; Venturini, T.; Heymann, S.; Bastian, M. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLoS ONE* **2014**, *9*, e98679.

(43) Moore, S. J.; MacDonald, J. T.; Wienecke, S.; Ishwarbhai, A.; Tsipa, A.; Aw, R.; Kylilis, N.; Bell, D. J.; McClymont, D. W.; Jensen, K., et al. Rapid acquisition and model-based analysis of cell-free transcription–translation reactions from nonmodel bacteria. *Proceedings of the National Academy of Sciences* **2018**, *115*, E4340–E4349.

(44) Meyer, A. J.; Segall-Shapiro, T. H.; Voigt, C. A. Marionette: E. coli containing 12 highly-optimized small molecule sensors. *bioRxiv* **2018**, 285866.

(45) Hu, C. Y.; Varner, J. D.; Lucks, J. B. Generating effective models and parameters for RNA genetic circuits. *ACS synthetic biology* **2015**, *4*, 914–926.

(46) Pasotti, L.; Bellato, M.; De Marchi, D.; Magni, P. Mechanistic models of inducible synthetic circuits for joint description of DNA copy number, regulatory protein level, and cell load. *Processes* **2019**, *7*, 119.

(47) Transtrum, M. K.; Qiu, P. Bridging mechanistic and phenomenological models of complex biological systems. *PLoS computational biology* **2016**, *12*, e1004915.

(48) Pandey, A.; Murray, R. M. Model Reduction Tools For Phenomenological Modeling of Input-Controlled Biological Circuits. *bioRxiv* **2020**,

(49) Smith, L. P.; Hucka, M.; Hoops, S.; Finney, A.; Ginkel, M.; Myers, C. J.; Moraru, I.; Liebermeister, W. SBML level 3 package: hierarchical model composition, version 1 release 3. *Journal of integrative bioinformatics* **2015**, *12*, 603–659.

(50) Travis-CI. `https://travis-ci.org/`.

(51) Codecov. `https://codecov.io/`.

(52) Roehner, N.; Zhang, Z.; Nguyen, T.; Myers, C. J. Generating systems biology markup language models from the synthetic biology open language. *ACS synthetic biology* **2015**, *4*, 873–879.