ctRSD-simulator-2.0

Release 2.0.0

Samuel Schaffter, Terence Murphy

May 01, 2023

GENERAL:

Download ctRSD-simulator -> DOWNLOAD

The following packages are used internally within the simulator and will need to be installed and available in the Python environment the simulator is running within: *numpy*, *scipy*, *matplotlib*, *re*, *math*, *xlrd*

CHAPTER

ONE

TABLE OF CONTENTS

Here you will be able to access information on all of the different components of the ctRSD-simlator 2.0. Please take a look at our overview to become familar with the simulator.

1.1 Overview

Welcome to the documentation for ctRSD-simulator-2.0!

Download ctRSD-simulator-2.0.0 version -> DOWNLOAD

ctRSD-simulator-2.0 was designed to be a comprehensive, scalable, and user-friendly model for simulating the kinetics of contranscriptionally encoded RNA strand displacement (ctRSD) circuits.

ctRSD circuits are an emerging technology for programmabale and scalable molecular computations. ctRSD circuits can execute multilayer cascades, logic, and signal amplification, and these elementary functions can be integrated to orchestrate sophisticated information processing tasks like digital calculations and pattern recognition.

The goal of the simulator is to aid the user to design, predict, and understand the behavior of ctRSD circuits. Also, providing experimentalists with a functional model allowing for the efficient *in silico* prototyping of ctRSD systems.

In addition to the kinetic simulations, the simulator package also has a *sequence compiler function* that can be used to assemble the DNA sequences necessary to test a specific component / design in experiments.

1.2 Model Reactions and Implementation

1.2.1 Model overview

The reactions currently incorporated in the model are shown below. The simulator converts all the reactions below into the appropriate rate equations derived from mass action kinetics. This produces a system of ordinary differential equations that is then numerically integrated with the *solve_ivp()* function in Python's SciPy package. Concentrations in the simulator are in *nM* and time is in *seconds*. The default values for rate constants can be found *here*

The curly bracket notation specifies the nomenclature of the species where i,j,k,…etc represent the domain numbers with the first index specifying the input domain and the second index specifying the output domain {input,output}. The names of species and the rate constants in the schematics below match how they are defined as variables in the model. The model is matrixed based so the index in the curly brackets also corresponds to the matrix index where the species concentrations are stored, or where the value of individual rate constants are stored in the rate constant matrices. See *here* for more details on matrix definitions.

The model uses a universal toehold for all strand displacement reactions. So from the model's perspective all outputs have the same toehold and any output with an output domain index that matches the input domain index of a gate can react with that gate, i.e., O{3,1} will react with G{1,4} and G{1,5}. If it is desired to model a system in which multiple

toeholds with different strand displacement rates, the appropriate index of the krsd rate constant matrix can be changed in molecular_species for each gate.

The model treats input species as outputs with identical input-output indices, *i.e.,* I{1} = O{1,1}, I{4} = O{4,4}. The user can specify input species in the model functions, such as when defining species in *molecular_species()* or when pulling concentrations for analysis with *output_concentration()*, but internally these will be handled in the output matrices. This implementation means other species that produce outputs with identical input-output indices, such as G{1,1}, AG{3.2,2} should not be used in the model as their outputs will behave as inputs rather than outputs.

There are a few things to note about the way the rate constant matrices are defined and implemented.

1) For RNA strand displacement reactions, unless otherwise stated, the forward rate constants are associated with the gates. This means the rate constant for a forward strand displacement reaction for a specific gate, for example G{1,3}, can be changed and that rate constant will be used with all outputs that react with that gate, i.e., O{4,1} and O{5,1} will both use the forward rate constant associated with G{1,3}. It is not possible to give O{4,1} and O{5,1} unique forward rate constants for a strand displacement reaction with G{1,3} (or any gate with input domain index 1). **Changes to forward RNA strand displacement reactions for individual gates should be done in the** *molecular_species()* **function when defining the gates.** Changing the forward RNA strand displacement rate constant is not a valid option for outputs in *molecular_species()*.

2) The reverse rate constants for RNA strand displacement reactions are associated with the outputs. This means the rate constant for a reverse strand displacement reaction for a specific output, for example O{1,3}, can be changed and that rate constant will be used with all GO{} species that react with the output, i.e., GO{4,1} and GO{5,1} will both use the reverse rate constant associated with O{1,3}. It is not possible to give GO{4,1} and GO{5,1} unique reverse rate constants for a strand displacement reaction with O{1,3} (or any output with input domain index 1). **Changes to reverse RNA strand displacement reactions for individual outputs should be done in the** *molecular_species()* **function when defining the outputs, or when defining the gates – the reverse rate that is defined with a gate with correspond to the output of that gate.** NOTE: because inputs are stored on the diagonal of the output matrix the diagonal of the reverse RNA strand displacement rate constant matrix is set to 0. This is because inputs only possess an output domain index and cannot participate in reverse RNA strand displacement reactions

1.2.2 ctRSD Reaction Schematics

Below the subscript "temp" indicates a DNA transcription template that encodes for the RNA species specified in its name.

We model RNA transcription and degradation using a first order approximation for enzyme kinetics. From this approximation the transcription rate is linearly proportional to the template concentration, *e.g.,* ktxnO{k,i}*[Otemp{k,i}] where ktxn{k,i} is the apparent first order rate constant for production of RNA from Otemp{k,i}. Likewise, the degradation rates (if included in the model) are linearly proportional to the concentration of RNA, *e.g.,* kssdO{i,j}*[O{i,j}].

Base ctRSD Reactions

Fuel Reactions

AND Gate Reactions

Degradation Reactions

Thresholding Reactions

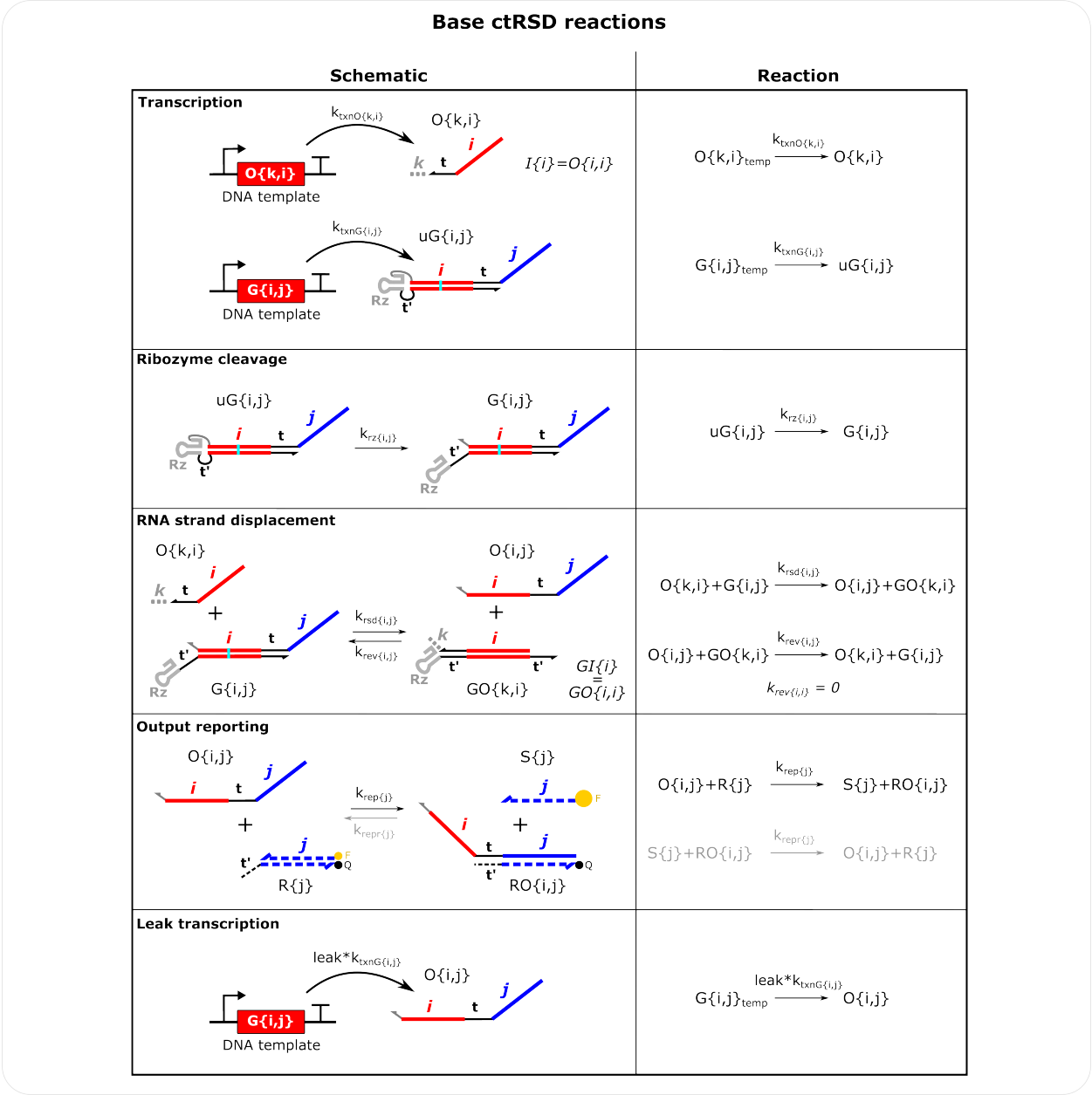

Fig. 1: **Base ctRSD Reactions Schematics** i,j,k represent input domain and output domain indices within the matrices and vectors that make up the model. They can be any integer value from 1 to N, where N is the total number of domains defined in model initialization. In the model inputs are not a separate species. Instead, inputs are modeled as outputs with the same input and output domain (tracked on the diagonal of the output matrix). For example, I{1} is modeled as O{1,1}. The user can still specify inputs in *molecular_species()* and *output_concentration()* and the simulator will convert these inputs to the appropriate output. The reverse strand displacement reaction for reporters is shown in gray because the krepr rate constants are initialized as 0 but they can be changed in *global_rate_constants()* or *molecular_species()* to simulate reversible reporting reactions. For the leak transcription reaction *leak* is a percentage (default 0.03). Note the krev rate constant for inputs is 0 as they only have an output domain and cannot reverse a strand displacement reaction. **The production of species S{j} can be compared to experiments with DNA reporters.**

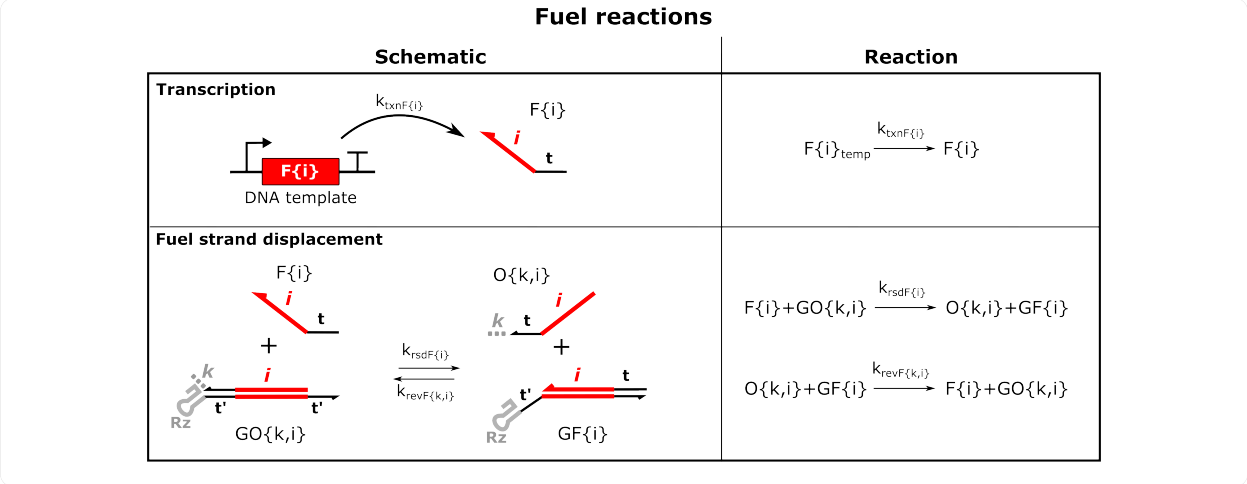

Fig. 2: **Fuel Reactions Schematics** Note the forward fuel strand displacement rate constant is associate with the fuel strand. Fuel reactions with AGs are shown below and are defined similarly.

Comparator Gate Reactions

1.2.3 Detailed model description

1.3 Installing and Initializing Simulator

The following packages are used internally within the simulator and will need to be installed and available in the Python environment the simulator is running within: *numpy*, *scipy*, *matplotlib*, *re*, *math*, *xlrd*

If you are new to Python we recommend downloading Anaconda <https://www.anaconda.com/>`, which should automatically have the necessary auxiliary Python packages/libraries installed. We primarily used the Spyder IDE to develop this code and would also recommend that for new users of Python. However, there can be issues with getting Spyder to launch from Anaconda, particularly for Mac users.

Jupyter Notebook is another good option running routine simulations - the simulation code can be split into small managable chucks that typically only need minor edits across simulations: initialize model, change rate constants and populate molecular species / conditions, simulate, and plotting.

Before you can use all the functions and features of ctRSD-simulator-2.0, you must first follow these steps:

1. *download* ctRSD-simulator-2.0
2. import the simulator in your Python script
3. Instantiate the simulator's *class* -> input is # of domains (default=5)
4. Now use all the functions/features available in ctRSD-simulator-2.0!

Note:

You will need to import the simulator package in your local script to use it. After downloading a local copy of the package you can import the package with the code below. Note *ctRSD_simulator_200* refers to the name of latest simulator package downloaded from GitHub

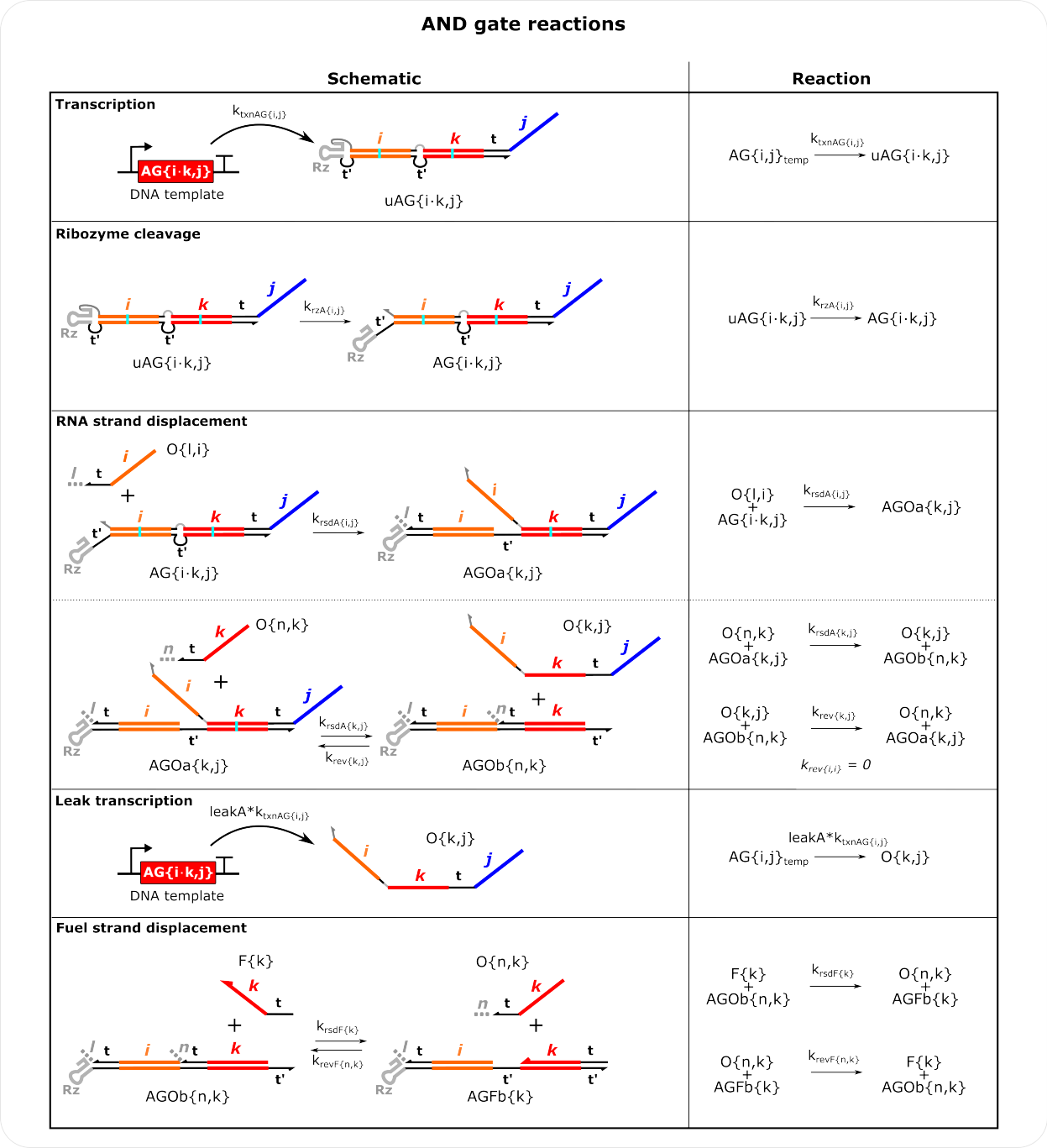

Fig. 3: **AND Gate Reactions Schematics** To reduce the number of species that needed to be tracked a few simiplications were made: The reaction of AG with the first input is considered irreversible. The final output of AG is defined by the second input domain and the output domain so this output will be lumped with outputs from single input gates that have the same indices *i.e.*, in the model O{k,j} from AG{i.k,j} is the same as O{k,j} from G{k,j}. Note there is not a unique reverse strand displacement rate constant for outputs from AG. The same matrix used for outputs from a single input gate is used. Fuel reactions with the first input domain of AGs is not considered, but in experiments such a reaction could occur. As with G, the leak transcription reaction *leakA* is a percentage (default 0.06). Note AGOa is defined by the second input domain and the output domain of AG while AGOb is defined by the indices of the second output bound to the gate.

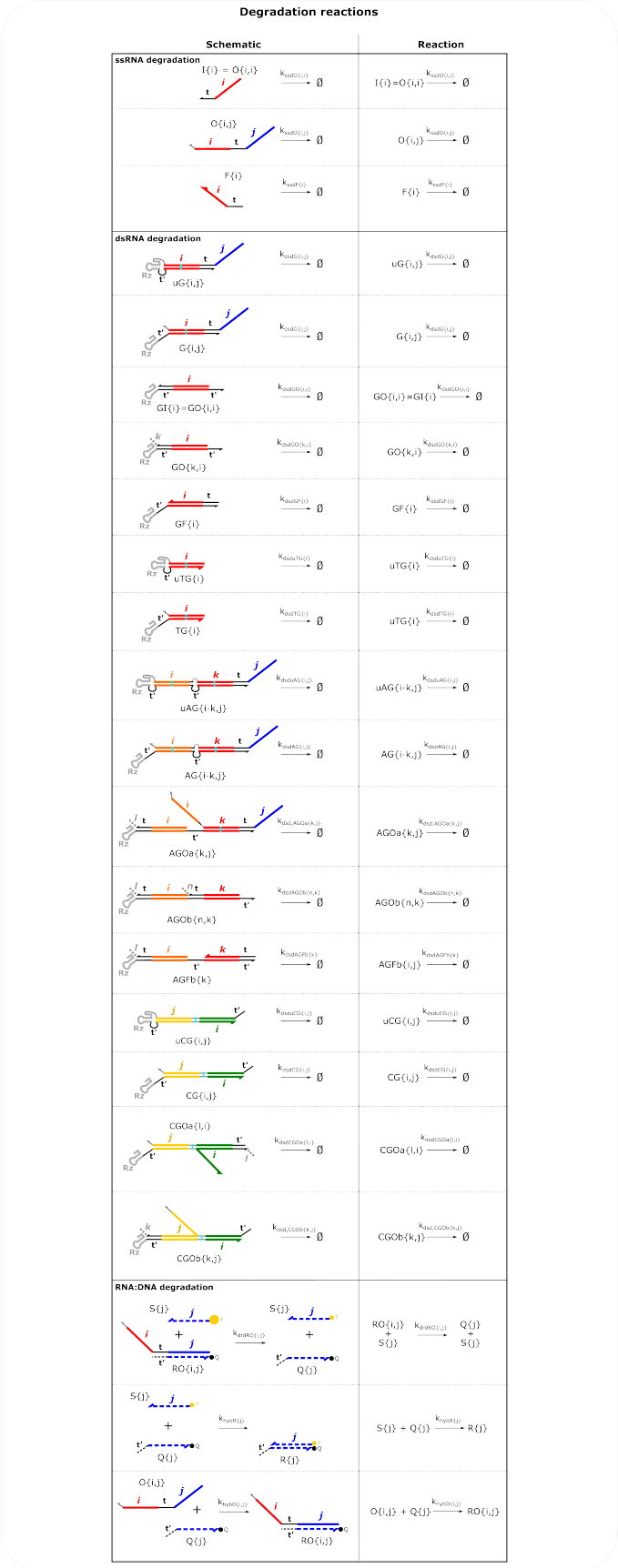

Fig. 4: **Degradation Reactions Schematics** Inputs (I) and GI species are shown separately here because they can be specified in *molecular_species()* and *output_concentration()* but they are actually modeled as O and GO, respectively, with identical input and output domain indices.

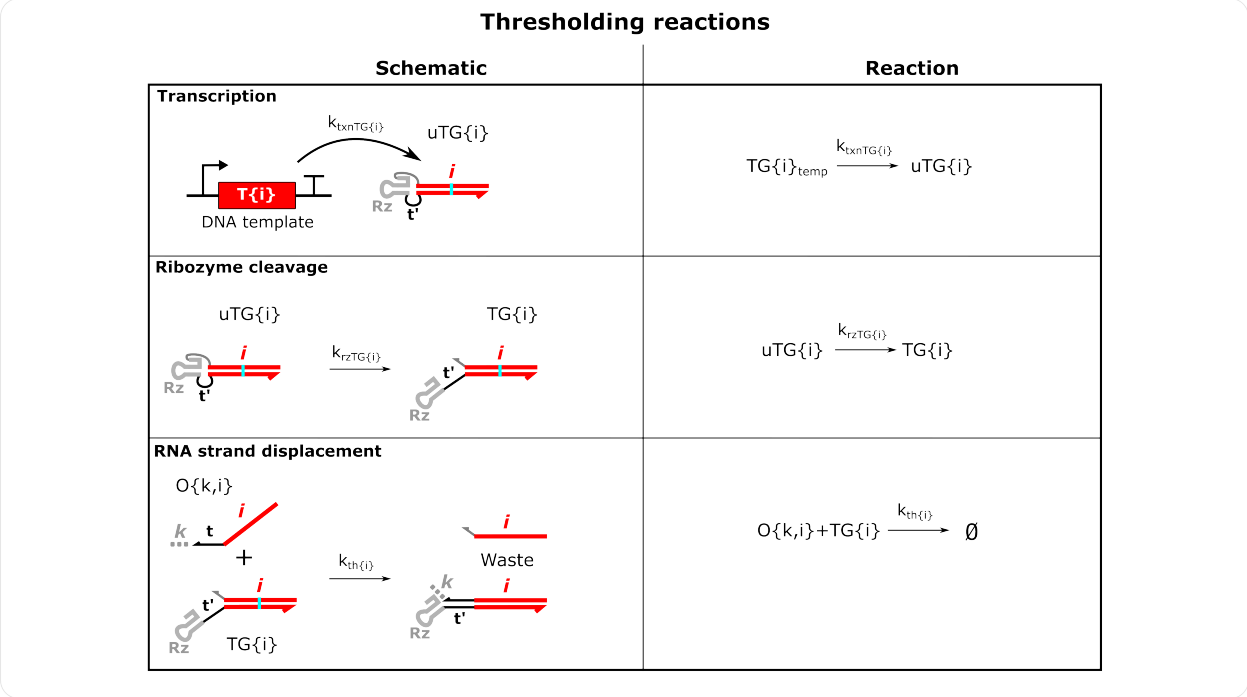

Fig. 5: **Thresholding Reactions Schematics** These reactions represent irreversible reactions without an output, essentially a sink for specific signals in a circuit. So these gates only have an input index.

```
import sys
sys.path.insert(1,'filepath of simulator on your computer')
import ctRSD_simulator_200 as RSDs
```

Please use figure below as an example:

```
# importing any additional packages for specific use in this script
import numpy as np
import matplotlib.pyplot as plt

# STEP 2 - add the path to the local copy of the simulator (an example path is shown␣
↪below)
#                       - import the simulator
import sys
sys.path.insert(1,'C:\\Users\\Name\\Documents\\SimFolder')
import ctRSD_simulator_200 as RSDs

'''
As will most Python classes, the model must be instantiated before accessing its␣
↪functions/features.

The model's class has an input that is the number of domains necessary for simulating a␣
↪system
(the default if no input is provided is 5 domains)
The number of domains needs to be >= the highest domain index of the components␣
```

(continues on next page)

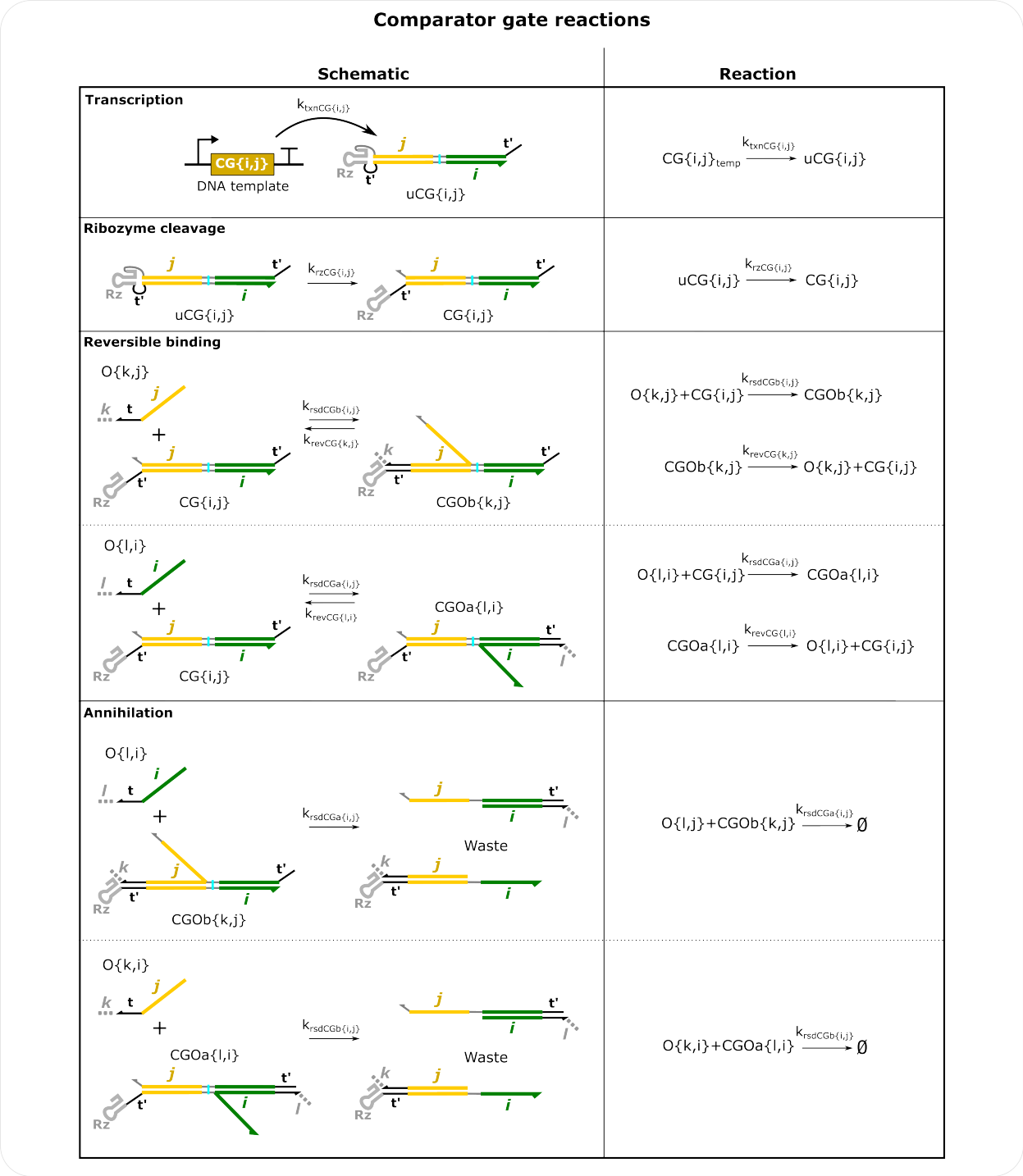

Fig. 6: **Comparator Gate Reactions Schematics** There are two forward strand displacement rate constants for each CG, krsdCGa that corresponds to reactions with the i domain and krsdCGb that corresponds to reactions with the j domain. Note the reverse strand displacement rate constants still follow the outputs, *e.g.*, the rate that O{k,j} dissociates from CG is defined by O{k,j} and not by the j domain of CG. This means outputs with the same output domain but different input domains can have different krevCG values. *molecular_species()* has an option to change the reverse reaction rate for all outputs with the same output domains when defining CG (krevCGa and krevCGb). Note CGOa and CGOb complexes are defined by the indices of the outputs bound to the comparator gate. For simulations containing more than one CG, the same input domain cannot be repeated in the same index for two gates. For example, CG{i,j} and CG{k,j} will result in an incorrect result because both gates have domain j in the second index. This should be ~~changed to CG{i,j} and CG{j,k} so that domain k is in a different index for the two gates. See the *Three comparator gate example*~~.

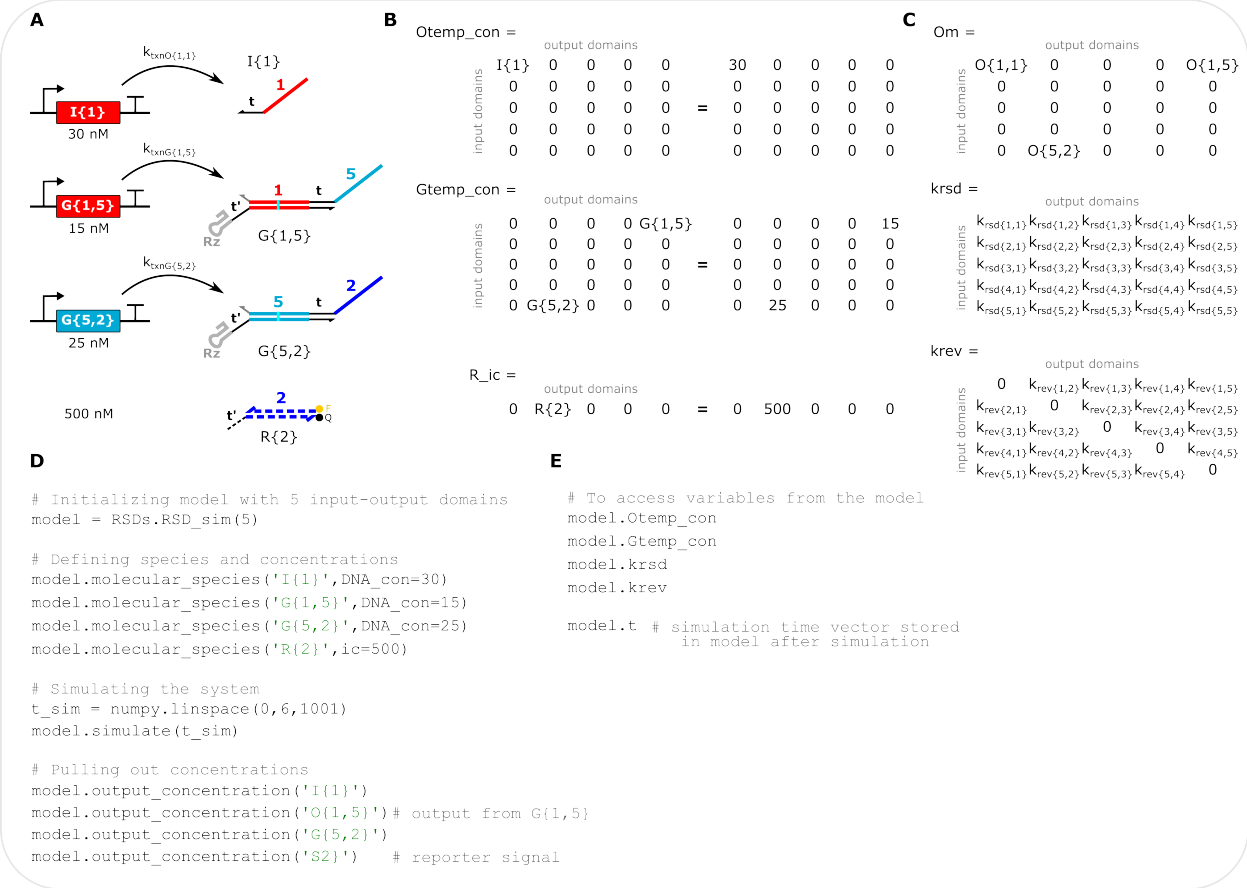

Fig. 7: **Example with selected matrix descriptions** (**A**) Schematic of a specific system to simulate, composed of and input, two cascaded gates, and a DNA reporter. Concentrations correspond to the DNA templates and reporter. (**B**) Matrix representation in the model of the defined system. Each column of a matrix corresponds to a specific output domain and each row corresponds to a specific input domain. Left: variables of the defined species with indexes shown. Right: variables with the defined concentrations shown. Otemp_con stores the concentrations of output (and input) DNA templates. Gtemp_con stores the concentrations of gate DNA templates. R_ic stores the initial concentration of DNA reporters. (**C**) Matrix representation of the output matrix (Om) and two rate constant matrices. Note Om stores both the outputs from the two gates in the system as well as the input because inputs are modeled as outputs with the same input and output domain. The krsd and krev matrices hold all the forward and reverse RNA strand displacement rate constants for all possible gates and outputs, respectively. The diagonal of the krev matrix, where the inputs are stored, is set to 0 because inputs cannot reverse RNA strand displacement reactions given that they only have output domains. The values of every entry in these matrices can be changed simultaneously with *global_rate_constants()* or the values of individual entries can be changed by specifying a new rate constant value for a specific species in *molecular_species()*. (**D**) Example code for setting up and simulating the system in panel **A**. (**E**). Commands to access the variables in panels **B** and **C**, see *Troubleshooting*.

(continued from previous page)

```
↪specified in molecular_species() for a given simulation
'''

# STEP 3 - create a model instance
model = RSDs.sim() # default number of domains (5)

'''
Now that a model instance has been created the model functions can be used
'''

# STEP 4 - set up a simulation

# molecular_species() function
model.molecular_species(I{1},DNA_con=25)
model.molecular_species(G{1,2},DNA_con=25)
model.molecular_species(R{2},ic=500)

# simulate() function
sim_time = np.linspace(0,3,1001)*3600 # seconds of simulation time
model.simulate(sim_time)

# output_concentration() function
S2 = model.output_concentration('S{2}')

# plotting
plt.plot(sim_time,S2)
plt.xlabel('Time (seconds)')
plt.ylabel('Reacted reporter (nM)')
```

1.4 RSD_sim Class

RSD_sim is a Python class that gives access to all of the simulator's built-in functions found *here*. To have access to the features of the class you must first instantiate an object of the class, directions for this can be found *here*.

This page gives info about the input of the object instantiation, and the initiaizations made for the class.

1.4.1 Class Object

__init__(*domains=5*)

Parameters:

domains: ***int*, *if NONE, default=5***
The highest input or output index of the species indicated in *molecular_species()*. For example, if G{5,9} is specified, *domains* needs to be 9 or greater. This can be higher than the highest index in the system but it cannot be lower than the highest index in the system.

1.4.2 Initializations

All DNA template concentrations and initial conditions are initialized at 0.

Rate Constants:

self.ktxnO = 0.013 (1/s)

self.ktxnG = 0.013 (1/s)

self.ktxnTG = 0.013 (1/s)

self.ktxnF = 0.013 (1/s)

self.ktxnAG = 0.013 (1/s)

self.ktxnCG = 0.013 (1/s)

self.leak = 0.03

self.krz = 0.00417 (1/s)

self.krsd = 1e3/1e9 (1/nM-s)

self.krev = 270/1e9 (1/nM-s), 0 for inputs

self.krep = 1e4/1e9 (1/nM-s)

self.krepr = 0 (1/s)

self.kth = 1e5/1e9 (1/nM-s)

self.krzTG = 0.00417 (1/s)

self.krsdF = 1e3/1e9 (1/nM-s)

self.krevF = 1e3/1e9 (1/nM-s)

self.krzA = 0.00417 (1/s)

self.krsdA = 1e3/1e9 (1/nM-s)

self.leakA = 0.06

self.krzCG = 0.00417 (1/s)

self.krsdCGa = 1e5/1e9 (1/nM-s)

self.krsdCGb = 1e5/1e9 (1/nM-s)

self.krevCG = 1 (1/s)

self.kssdO = 0 (1/s)

self.kssdF = 0 (1/s)

self.kdsduG = 0 (1/s)

self.kdsdG = 0 (1/s)

self.kdsdGO = 0 (1/s)

self.kdsdGF = 0 (1/s)

self.kdsduTG = 0 (1/s)

self.kdsdTG = 0 (1/s)

self.kdsduAG = 0 (1/s)

self.kdsdAG = 0 (1/s)

self.kdsdAGOa = 0 (1/s)

self.kdsdAGOb = 0 (1/s)

self.kdsdAGFb = 0 (1/s)

self.kdsduCG = 0 (1/s)

self.kdsdCG = 0 (1/s)

self.kdsdCGOa = 0 (1/s)

self.kdsdCGOb = 0 (1/s)

self.kdrd = 0 (1/s)

self.khybO = 1e6/1e9 (1/nM-s)

self.khybR = 1e6/1e9 (1/nM-s)

1.5 Functions

This section lists the different functions available within ctRSD-simulator-2.0.

Note:

Before all available functions can be accessed the model must be downloaded, imported, and instantiated!

More information on these steps can be found *here*.

1.5.1 Quick reference

model.global_rate_constants(kwargs**)**

Change specific rate constants for all the species involved in the reaction

model.molecular_species (*name*, *DNA_con=0*, *ic='False'*,**kwargs**)

Specify species involved in a given simulation (*name*). Species names follow the conventions in *ctRSD Reaction Schematics*. *DNA_con* specifies the DNA template concentration for transcribable species. *ic* specifies the initial concentration of the species in the simulation. The *kwargs* allow a specific rate constant for the individual species defined in *name* to be changed.

model.simulate (*t_vec*, *smethod='False'*, *iteration=1*)

Simulate the specific model instance for the time specified in *t_vec*. *smethod* can specify different numerical solver techniques. *iteration* allows discontinuous simulations. See *discontinuous simulation example*.

model.output_concentration (*name*)

Pull out the concentrations of different species as a function of time after a simulation. *name* follows the same conventions as in *molecular_species()* and as in the *ctRSD Reaction Schematics*

1.5.2 global_rate_constants()

model.global_rate_constants (**kwargs**)

with expanded keyword arguments (kwargs):

model.global_rate_constants (*krz='False'*, *krsd='False'*, *krev='False'*, *krep='False'*, *krepr='False'*, *kth='False'*, *krzTG='False'*,*krsdF='False', *krevF='False'*, *krsdA='False'*, *krzA='False'*, *krevCG='False'*, *krsdCGa='False'*, *krsdCGb='False'*, *krsdCG='False'*, *krzCG='False'*, *ktxnO='False'*, *ktxnG='False'*, *ktxnTG='False'* , *ktxnF='False'*, *ktxnAG='False'*, *ktxnCG='False'*, *ktxn='False'*, *kssdO='False'*, *kssdF='False'*, *kdsduG='False'*, *kdsdG='False'*, *kdsdGO='False'*, *kdsduAG='False'*, *kdsdAG='False'*, *kdsduCG='False'*, *kdsdCG='False'*, *kdsduTG='False'*,*kdsdTG='False', *kdsdGF='False'*, *kdsdAGOa='False'*,*kdsdAGOb='False', *kdsdAGFb='False'*,*kdsdCGOa='False', *kdsdCGOb='False'*, *kdrd='False'*, *kdeg='False'*, *kssd='False'*, *kdsd='False'*, *khybO='False'*, *khybR='False'*, *khyb='False'*, *leak='False'*, *leakA='False'*)

global_rate_constants is used to globally change rate constants for all species of the same type (single matrix), or certain groups of different types (multiple matrices), instead of changing a specific rate constant for just one individual species (index of a matrix).

The defaults for the rate constants are initialized in the RSD_sim class. These values can be found *here*.

Notes:

Rate constant values should be supplied with units of *nM* and *seconds*

More than one rate constant can be changed in a single call of *global_rate_constants()*.

The different transcription rates and degradation rates both have an input that will change all rates at once. *ktxn* changes all transcription rates in the simulator. *kdeg* changes all degradation rates in the simulator.

Parameters:

krz: ***float*, *optional***
Single ctRSD gate ribozyme cleavage rate constant.

krsd: ***float*, *optional***
Single ctRSD gate forward strand displacement rate constant.

krev: ***float*, *optional***
Output reverse strand displacement rate constant. This rate constant applies to outputs reacting with both GO and AGOb complexes. This rate constant is 0 for inputs because they only possess an output domain and cannot reverse a strand displacement reaction.

krep: ***float*, *optional***
Reporter forward strand displacement rate constant.

krepr: ***float*, *optional***
Reporter reverse strand displacement rate constant.

kth: ***float*, *optional***
Threshold gate forward strand displacement rate constant.

krzTG: ***float*, *optional***
Threshold gate ribozyme cleavage rate constant.

krsdF: ***float*, *optional***
Fuel forward forward strand displacement rate constant.

krevF: ***float*, *optional***
Fuel reverse strand displacement rate constant.

krsdA: ***float*****,** ***optional***
ctRSD AND gate forward strand displacement rate constant. Changing this rate constant in *global_rate_constants()* will change the reaction rate for both input domains of an AG. To change these rates individually see *molecular_species()* below.

krzA: ***float*****,** ***optional***
ctRSD AND gate ribozyme cleavage rate constant.

krevCG: ***float*****,** ***optional***
Reverse strand displacement rate constant for outputs on a CG. These rates follow outputs, not CG gate indices.

krsdCGa: ***float*****,** ***optional***
ctRSD comparator gate forward strand displacement rate constant for the first input domain of a CG (CG{i,_}).

krsdCGb: ***float*****,** ***optional***
ctRSD comparator gate forward strand displacement rate constant for the second input domain of a CG (CG{_,j}).

krsdCG: ***float*****,** ***optional***
ctRSD comparator gate forward strand displacement rate constant for both input domains. Changing this parameter will change both krsdCGa and krsdCGb.

krzCG: ***float*****,** ***optional***
ctRSD comparator gate ribozyme cleavage rate constant.

ktxnO: ***float*****,** ***optional***
Transcription rate constant for outputs (and inputs). This will change the transcription rates for both inputs (held along the diagonal of the ktxnO matrix) and the outputs. To change inputs individually change ktxnO in *molecular_species()* with the specific input.

ktxnG: ***float*****,** ***optional***
Transcription rate constant for gates.

ktxnTG: ***float*****,** ***optional***
Transcription rate constant for threshold gates.

ktxnF: ***float*****,** ***optional***
Transcription rate constant for fuels.

ktxnAG: ***float*****,** ***optional***
Transcription rate constants for AND gates.

ktxnCG: ***float*****,** ***optional***
Transcription rate constant for comparator gates.

kssdO: ***float*****,** ***optional***
Single stranded RNA degredation rate constant for outputs (and inputs).

kssdF: ***float*****,** ***optional***
Single stranded RNA degredation rate constant for fuel strands.

kdsduG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for uncleaved gates.

kdsdG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for uncleaved gates.

kdsdGO: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for gate:output (GO) complexes.

kdsduAG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for uncleaved AND gates.

kdsdAG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for AND gates.

kdsduCG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for uncleaved comparator gates.

kdsdCG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for comparator gates.

kdsduTG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for uncleaved threshold gates.

kdsdTG: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for threshold gates.

kdsdGF: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for gate:fuel (GF) complexes.

kdsdAGOa: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for AGOa complexes.

kdsdAGOb: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for AGOb complexes.

kdsdAGFb: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for AGFb complexes.

kdsdCGOa: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for CGOa complexes.

kdsdCGOb: ***float*****,** ***optional***
Double stranded RNA degredation rate constant for CGOb complexes.

kdrd: ***float*****,** ***optional***
Degredation rate constant for RNA in RNA:DNA hybrid duplexes.

khybO: ***float*****,** ***optional***
Hybridization rate constant for output binding to the Q strand of the reporter.

khybR: ***float*****,** ***optional***
Hybridization rate constant for the S strand to bind to the Q strand of the reporter.

leak: ***float*****,** ***optional***
The percentage of leak transcription from a single input gate (G). This should be supplied as a decimal representing a percentage, *i.e.*, 0.05 to represent 5%.

leakA: ***float*****,** ***optional***
The percentage of leak transcription from an AND gate (AG). This should be supplied as a decimal representing a percentage, *i.e.*, 0.05 to represent 5%.

Changes to multiple classes of rate constants with a single input:

kssd: ***float*****,** ***optional***
To change the degredation rate consant for all single stranded species

(kssdO, kssdF).

kdsd: ***float*****,** ***optional***
To change the degredation rate constant for all double stranded species

(kdsduG, kdsdG, kdsdGO, kdsduAG, kdsdAG, kdsduCG, kdsdCG, kdsduTG, kdsdTG, kdsdGF, kdsdAGOa, kdsdAGOb, kdsdAGFb, kdsdCGOa, kdsdCGOb).

ktxn: ***float*****, *optional***
To change change all transcription rate constants

(ktxnO, ktxnG, ktxnTG, ktxnF, ktxnAG, ktxnCG)

kdeg: ***float*****, *optional***
To change the degredation rate consant for all species

(kssdO, kssdF, kdsduG, kdsdG, kdsdGO, kdsduAG, kdsdAG, kdsduCG, kdsdCG, kdrd).

khyb: ***float*****, *optional***
To change the hybrdization rate constant for both output and reporter

(khybO, khybR).

1.5.3 molecular_species()

model.molecular_species (*name*, *DNA_con=0*, *ic='False'*,**kwargs**)

with expanded keyword arguments (kwargs):

model.molecular_species (*name*, *DNA_con=0*, *ic='False'*, *krz='False'*, *krsd='False'*, *krev='False'*, *krep='False'*, *krepr='False'*, *kth='False'*, *krzTG='False'*, *krsdF='False'*, *krevF='False'*, *krsdA='False'*, *krzA='False'*, *krevCG='False'*, *krevCGa='False'*, *krevCGb='False'*, *krsdCG='False'*, *krsdCGa='False'*, *krsdCGb='False'*, *krzCG='False'*, *ktxnO='False'*, *ktxnG='False'*, *ktxnTG='False'*, *ktxnF='False'*, *ktxnAG='False'*, *ktxnCG='False'*, *kssdO='False'*, *kssdF='False'*, *kdsduG='False'*, *kdsdG='False'*, *kdsdGO='False'*, *kdsduAG='False'*, *kdsdAG='False'*, *kdsduCG='False'*, *kdsdCG='False'*, *kdsduTG='False'*, *kdsdTG='False'*, *kdsdGF='False'*, *kdsdAGOa='False'*, *kdsdAGOb='False'*, *kdsdAGFb='False'*, *kdsdCGOa='False'*, *kdsdCGOb='False'*, *kdrd='False'*, *khybO='False'*, *khybR='False'*, *leak='False'*, *leakA='False'*):

molecular_species is used to initialize all species involved in the system being simulated.

Default DNA templates, initial conditions, and rate constants are initialized in RSD_sim. These values can be found *here*.

Notes:

Rate constant values should be supplied with units of *nM* and *seconds*

More than one rate constant can be changed in a single call of *molecular_species()*.

Specifying an optional rate constant parameter in this function will only change the value for the individual species specified in *name*. The rest of the species will have the default values or the values specified in *global_rate_constants()* if called before *molecular_species()*.

Only rate constant values relevent to the species defined with *name* can be changed. For example, it is not possible to change krz for an input or output. Likewise it is not possible to change krsdA for a single input gate (G). A warning message will be issued if the specified rate constant cannot be changed for the named species. If multiple rate constants are changed in a single call the warning message will not specify which rate constant cannot be changed.

Parameters:

name: ***string***
name inputs that show multiple options function with each of those options. All *name* inputs are also not case sensitive.

Name of species being initialized

- Input -> I{domain} / IN{domain} / INP{domain} / INPUT{domain}
- Gate -> G{domainI,domainO} / GATE{domainI,domainO}
- Reporter -> R{domain}, REP{domain}, REPORTER{domain}
- Output -> O{domainI,domainO} / OUT{domainI,domainO} / OUTPUT{domainI,domainO}
- Uncleaved Gate -> uG{domainI,domainO}
- Gate-Input Complex -> GI{domain} (not case sensitive)
- Gate-Output Complex -> GO{domainI,domainO}
- Reporter-Output Complex -> RO{domainI,domainO}
- Reporter Signal Strand -> S{domain}
- Reporter Signal Complement Strand -> Q{domain}
- Uncleaved Threshold Gate -> uTG{domain} / uT{domains} / uTH{domain}
- Threshold Gate -> TG{domain} / T{domains} / TH{domain}
- Fuel -> F{domain}
- Gate-Fuel Complex -> GF{domain}
- Uncleaved AND Gate -> uAG{domainI,domainO}
- AND Gate -> AG{domainI1.domainI2,domainO} / G{domainI1.domainI2,domainO} / GATE{domainI1.domainI2,domainO}
- AND Gate-Output Complex A -> AGOa{domainI2,domainO}
- AND Gate-Output Complex B -> AGOb{domainI,domainO}
- AND Gate Fuel Complex B -> AGFb{domain}
- Uncleaved Comparator Gate -> uCG{domainI1,domainI2}
- Comparator Gate -> CG{domainI1,domainI2}
- Comparator Gate-Output Complex A -> CGOa{domainI,domainO}
- Comparator Gate-Output Complex B -> CGOb{domainI,domainO}

DNA_con: ***float*****,** ***if NONE,default=0***
DNA concentration for inputed species. This and ic are the two ways a user can initialize a component being involved in the system. (Only applies to Input,Output,Gate,Fuel,AG,TG,CG,Reporter). Other than for Reporters this variable specifies the DNA template concentration for transcribable components in ctRSD circuits. For Reporter this is the same as a fixed initial concentration (ic). Other than for Reporter this represents the concentration of the DNA template that encodes for the transcription of the species specified in *name*.

ic: ***float*****,** ***optional*****,** ***if NONE,default=0***
Initial Concentration for inputted species. This and DNA_con are the two ways a user can initialize a component being involved in the system. Other than for the Reporter, this refers to the initial concentration of the RNA species specified in *name*

krz - leakA: ***floats*****,** ***optional***
These optional rate constant parameters are defined as in the *global_rate_constants()* function. But changing them in *molecular_species()* will only change the value for the individual species specified in *name* Below are some additional paramaters and caveats unique to *molecular_species()*

ktxnO: ***float*****,** ***optional***
Transcription rate constant for outputs (and inputs).

To change the transcription rate of an indivdual input use this optional parameter. ktxnI is not a valid input.

EX: model.molecular_species(I{3}, ktxnO=0.02)

krsdA: ***float*****,** ***optional***
ctRSD AND gate forward strand displacement rate.

If specified with AG, this will change the rate constant for the reaction with the first output in the AG. To change the rate constant for the reaction with the second output the user can specify an AGOa species anc change this rate constant.

Ex:

```
model.molecular_species(AG{3.1,2}, krsdA=1e5/1e9) # changes krsdA for first␣
↪input domain

model.molecular_species(AGOa{1,2}, krsdA=1e5/1e9) # changes krsdA for␣
↪second input domain
```

krevCG: ***float*****,** ***optional***
Reverse strand displacement rate constant for outputs on a CG. These rates follow outputs, not CG gate indices. This rate constant should be changed with specific outputs (or inputs) specified in *molecular_species()*, *e.g.*, molecular_species(O{3,1},krevCG=0.4) or molecular_species(I{3},krevCG=0.4) which changes the rate these outputs/inputs dissociate from a CG. To change the reverse rates for all outputs that correspond to either domain of a CG use the *krevCGa* and *krevCGb* paramters below.

krevCGa: ***float*****,** ***optional***
Reverse strand displacement rate constant for all outputs (or inputs) that bind to the first index of a CG, *i.e.*, CG{i,_}. This changes the entire i-th column of the *krevCG* matrix.

krevCGb: ***float*****,** ***optional***
Reverse strand displacement rate constant for all outputs (or inputs) that bind to the second index of a CG, *i.e.*, CG{_,j}. This changes the entire j-th column of the *krevCG* matrix.

If an AND gate (AG) is specified its leak transcription percentage can be changed with either *leak* or *leakA*.

1.5.4 simulate()

model.simulate (*t_vec*, *smethod='False'*, *iteration=1*)

simulate is used to run a simulation for a provided amount of time using the components previously initialized by molecular_species. simulate also includes the discontinuous feature of the simulator.

Parameters:

t_vec: ***array, type=float***
Array of time points signifying the simulation run time and interval.

smethod: ***string*****,** ***optional*****,** ***if NONE, default='LSODA'***

Solver method inputted into scipy.integrate.solve_ivp ODE integrator:

- RK45
- RK23
- DOP853

- Radau
- BDF (recommended for comparator gate simulations)
- LSODA

For simulations using comparator gates (CG) the 'BDF' method is recommended. This can speed up the computation time.

iteration: ***int*, *if NONE,default=1***
Controlling input for discontinuous feature.

Iteration signifies which step in a total simulation that the inputted simulation time and previously initialized species are tied to. For example, iteration=1 signifies first time step of simulation, iteration=2 signifies second time step of same simulation. There is no maximum in iteration, but must be positive integer.

Example of discontinuous feature can be found *here*.

1.5.5 output_concentration()

model.output_concentration (*name*)

output_concentration is used to pull out desired output concentrations created after running of the simulate function.

Note:

Concentrations are output with *nM* units

To compare to experiments with DNA reporters, the concentration of S{j} can be pulled out for plotting: *reacted_reporter = model.output_concentration('S{2}')*

Parameters:

name: ***string***
name inputs that show multiple options function with each of those options. All *name* inputs are also not case sensitive.

Name of species being initialized

- Input -> I{domain} / IN{domain} / INP{domain} / INPUT{domain}
- Gate -> G{domainI,domainO} / GATE{domainI,domainO}
- Reporter -> R{domain}, REP{domain}, REPORTER{domain}
- Output -> O{domainI,domainO} / OUT{domainI,domainO} / OUTPUT{domainI,domainO}
- Uncleaved Gate -> uG{domainI,domainO}
- Gate-Input Complex -> GI{domain} (not case sensitive)
- Gate-Output Complex -> GO{domainI,domainO}
- Reporter-Output Complex -> RO{domainI,domainO}
- Reporter Signal Strand -> S{domain}
- Reporter Signal Complement Strand -> Q{domain}
- Uncleaved Threshold Gate -> uTG{domain} / uT{domains} / uTH{domain}
- Threshold Gate -> TG{domain} / T{domains} / TH{domain}
- Fuel -> F{domain}

- Gate-Fuel Complex -> GF{domain}
- Uncleaved AND Gate -> uAG{domainI,domainO}
- AND Gate -> AG{domainI1.domainI2,domainO} / G{domainI1.domainI2,domainO} / GATE{domainI1.domainI2,domainO}
- AND Gate-Output Complex A -> AGOa{domainI2,domainO}
- AND Gate-Output Complex B -> AGOb{domainI,domainO}
- AND Gate Fuel Complex B -> AGF{domain}
- Uncleaved Comparator Gate -> uCG{domainI1,domainI2}
- Comparator Gate -> CG{domainI1,domainI2}
- Comparator Gate-Output Complex A -> CGOa{domainI,domainO}
- Comparator Gate-Output Complex B -> CGOb{domainI,domainO}

1.6 Sequence Compiler

Sequence compiler is a function avaliable in the simulator that takes in a species name as an input (same nomenclature used elsewhere in the simulator), among other function inputs, and outputs the DNA sequence that encodes for the RNA species that meets the input specifications.

Note:

Although it is a function within the simulator, and is accessed in the same way as the other functions, it does not require a system to be simulated to be used. Hence why its documentation is being kept seperate from the other functions. It just requires that the simulator package be imported.

Sequence compiler was included in the simulator so that users could have all the useful tools in designing ctRSD circuits in one place.

1.6.1 Download Domains List

Below is a link to download a provided spreadsheet that contains the list of domains for ctRSD reactions used to compile the sequences. The local file path on where this file is stored is an input to the sequence compiler function. Many of the optional inputs to the sequence compiler use the names of sequence domains contained in this Excel workbook - they can be accessed by specifying the name in the first column of the Excel workbook with the appropriate optional parameter. To add custom sequences for a domain, one can add a sequence domain with a unique name to a local copy of the Excel workbook in the correct sheet and then use the name of that sequence domain in the sequence compiler optional inputs.

ctRSD Domains List Spreadsheet can be found here

1.6.2 Function Overview

The figures below illustrate the various options for the sequence compiler. Note, the comparator gates (CG) and thresholding gates (TG) have not been exhaustively tested in experiments and their exact designs might be subject to change in future iterations.

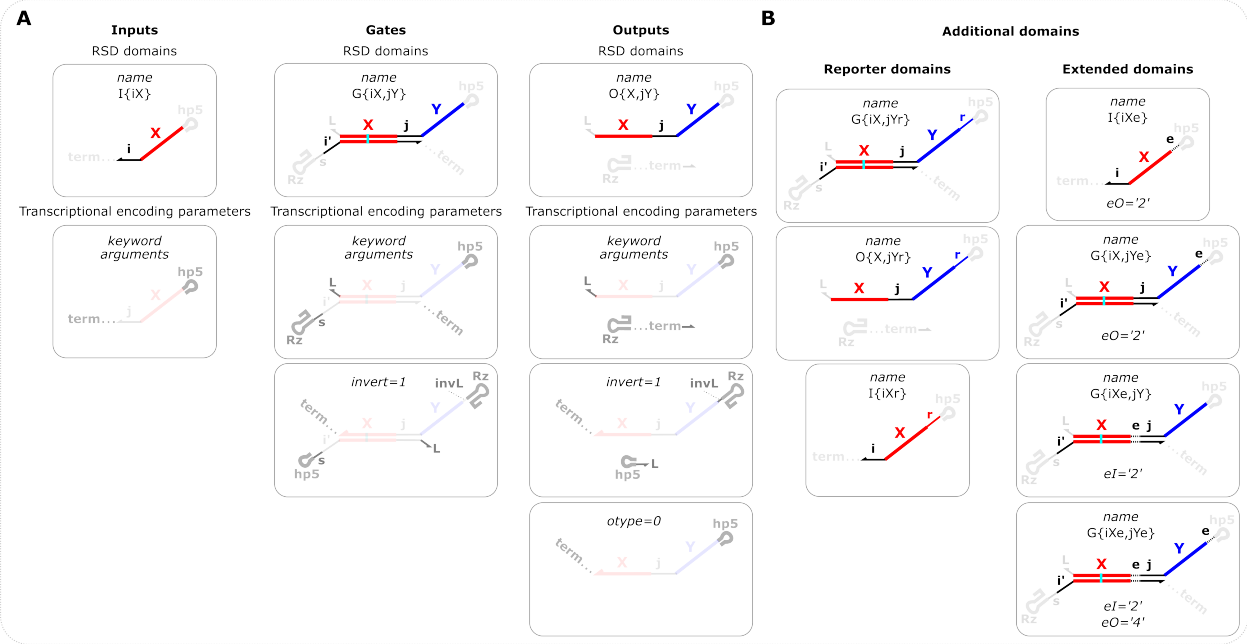

Fig. 8: **Overview of input, gate, and output definitions and options** (**A**) RSD domains are domains relevant to RNA strand displacement and are specified in the name variable. i,j,k refer to toehold domains and X and Y refer to branch migration domains which can be 1,2,3,4,5,…etc. If only the name variable is specified, the default transcriptional encoding parameters will be used. The transcriptional encoding parameters refer to domains appended to the RSD domains to facilitate transcription of the ctRSD components. The names of the optional keyword arguments are shown for each component in the second row. The third row shows the schematics if the transcription order is inverted by setting the invert keyword argument to 1. For outputs it is also possible to specify otype=0 to produce an output sequence that does not contain a ribozyme. (**B**) In addition to the options shown in panel A, there are other domains that can be specified. (Left) Reporter domains: domains necessary for irreversible reactions with a fluorescent reporter complex (r). These are specified solely in the name variable. Note there is a unique r domain for each output domain, i.e., an r for domain 1 and an r for domain 2, etc. (Right) Extended domains: domains that extend the branch migration length of components. These can be specified in either the input, output or both domains of a gate. Although not shown, extended domains are also an option for outputs. The existence of an extended domain is specified in the name variable and the identity of the extended domain is specified with the keyword arguments eI and eO. Note that inputs require the eO keyword be used to specify the identity of the output domain.

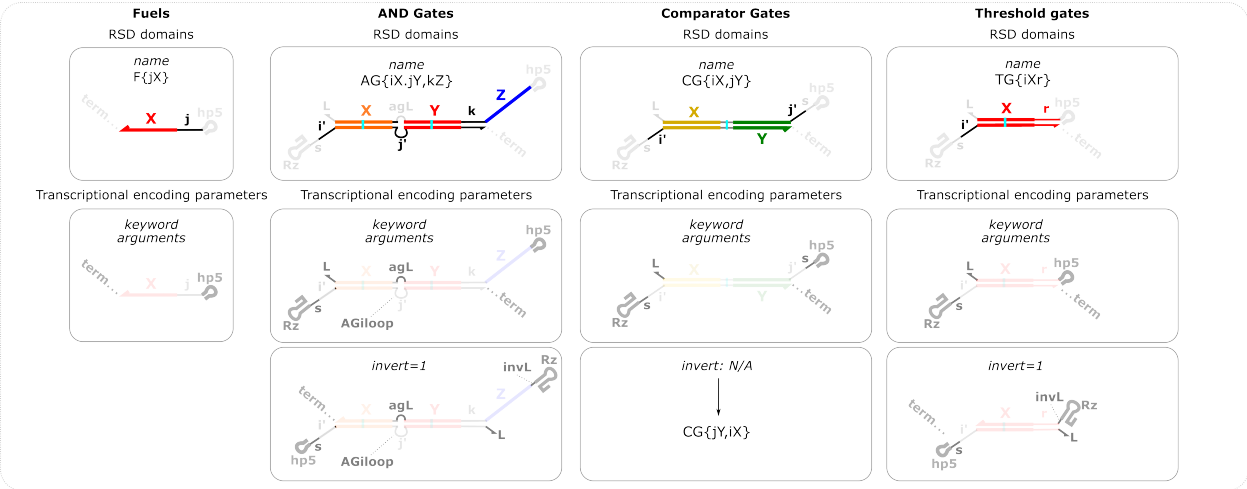

Fig. 9: **Overview of fuel, AND gate, comparator gate, and thresholding gate definitions and options** RSD domains are domains relevant to RNA strand displacement and are specified in the name variable. i,j,k refer to toehold domains and X,Y,Z refer to branch migration domains which can be 1,2,3,4,5,...etc. If only the name variable is specified, the default transcriptional encoding parameters will be used. The transcriptional encoding parameters refer to domains appended to the RSD domains to facilitate transcription of the ctRSD components. The names of the optional keyword arguments are shown for each component in the second row. The third row shows the schematics if the transcription order is inverted by setting the invert keyword argument to 1. Reporter domains can also be specified for AND gates e.g., AG{iX.jY,kZr}. Extended domains are options for fuel and AND gate. For fuel components the convention follows that of input components in the figure above. For AND gates if an e domain is specified for either input domain then that domain will be used for both input domains. But eO can be different than eI for AND gates. Extended domains are currently not an option for threshold gates.

1.6.3 Function Documentation

Warning!

Before all available functions can be accessed the model must be downloaded, imported, and instantiated!

More information on these steps can be found *here*.

model.ctRSD_seq_compile (*name*, *filepath*, *Rz='Ro'*, *L='L'*, *term='T7t'*, *hp5='5hp'*, *prom='T7p'*, *eI=''*, *eO=''*, *s=''*, *invert=0*, *invL='A'*, *agL='TA'*, *AGiloop=5*, *otype=1*, *rna=0*, *us=[]*, *ds=[]*, *temp_len=0*):

name inputs that show multiple options function with each of those options. All *name* inputs are also not case sensitive.

Parameters:

name: ***string***

Name of species sequence will be compiled for.

- Input (I) -> I{domain} / IN{domain} / INP{domain} / INPUT{domain}
- Output (O) -> O{domainI,domainO} / OUT{domainI,domainO} / OUTPUT{domain}
- Gate (G) -> G{domainI,domainO} / GATE{domainI,domainO}
- AND Gate (AG) -> AG{domainI1.domainI2,domainO} / G{domainI1.domainI2,domainO} / GATE{domainI1.domainI2,domainO}
- Fuel (F) -> F{domain}
- Threshold gate (TG) -> TG{domain} / T{domains} / TH{domain}
- Comparator Gate (CG) -> CG{domainI1,domainI2}

filepath: ***string***
Local file path for ctRSD domains list Excel sheet. Donwload *here*. This needs to include the filename of the Excel sheet: /local file path/ctRSD_domains_list.xls

Rz: ***string*, *optional*, *if NONE,default='Ro'***
Ribozyme sequence. Denoting an 'x' before the ribozyme sequence name (for example: 'xRo') will use an inactive ribozyme mutant.

L: ***string*, *optional*, *if NONE,default='L'***
Linker sequence adjacent to the 5' end of the ribozyme.

term: ***string*, *optional*, *if NONE,default='T7t'***
Terminator sequence.

hp5: ***string*, *optional*, *if NONE,default='5hp'***
The sequence of the 5' hairpin on input and output strands

prom: ***string*, *optional*, *if NONE,default='T7p'***
Promoter sequence.

eI: ***string*, *optional*, *if NONE,default=''***
An extended sequence at the 5' end of output domains on gates, AND gates, outputs, inputs, and fuels. 'e' must be specified in the output domain of the *name* of the species for this input to be valid, i.e., G{u1e,_} or AG{u3e.u1e,_} or O{u5e,_}. eI will be the same for both input domains of an AG.

eO: ***string*, *optional*, *if NONE,default=''***
An extended sequence at the 5' end of input domains on gates, AND gates, and outputs. 'e' must be specified in the input domain of the *name* of the species for this input to be valid, i.e., G{_,v1e} or AG{_,w2e} or O{_,v3e} or I{u1e} or F{w5e}. Note that eO is used to specify 'e' domains on single input

s: ***string*, *optional*, *if NONE,default=”***
Spacer sequence between the ribozyme and the input toehold of a gate. Cannot be specified for the second input toehold of an AG. ‘s4’ is the default spacer domain for TG.

invert: ***Boolean*, *optional*, *if NONE,0***
Inverted transcription order - change to 1. This will start transcription at the 5’ end of the input toehold of a gate. Not a valid input for CGs.

invL: ***string*, *optional*, *if NONE,’A’***
For inverted gates only, a linker sequence between the ribozyme domain of the gate and the output domain. Defined as a direct sequence so the default is an ‘A’ base. Any sequence can be directly specified as an input.

agL: ***string*, *optional*, *if NONE,’TA’***
For AND gates only, a linker sequence between the two input domains of an AND gate. Defined as a direct sequence so the default is an ‘A’ base. Any sequence can be directly specified as an input.

AGiloop: ***int*, *optional*, *if NONE,5***
For AND gates only, the number of bases in the internal loop toehold for the second input on an AND gate. This can be 5 bases (default) or 6 bases.

otype: ***Boolean*, *optional*, *if NONE,1***
Specifying the type of output strand to encode. 1 (default) refers to an output that has a ribozyme sequence at the 3’ end to mimic the cleaved output of a ctRSD reaction. 0 refers to an output sequence that ends in a terminator and does not use a ribozyme.

rna: ***Boolean*, *optional*, *if NONE,0***
Set rna = 1 to have the output sequence be the RNA encoded by the DNA template rather than the DNA template sequence. For RNA sequences with a ribozyme a ‘|’ will denote the cleavage site. xRz sequences (inactive ribozyme mutants) will not indicate the cleavage site.

us: ***list*, *optional*, *if NONE,[]***
List of upstream sequences to append to the DNA template. Sequences will be appended 5’ to 3’ upstream of the promoter in the order they are specified in the list.

ds: ***list*, *optional*, *if NONE,[]***
List of downstream sequences to append to the DNA template. Sequences will be appended 5’ to 3’ downstream of the terminator in the order they are specified in the list. The option ‘exc’ can be used in conjtion with *temp_length* below to create sequences of a specific length.

temp_len: ***int*, *optional*, *if NONE,0***
Specifying the total length of the DNA template. Typically, this can be used to get a template that is 125 bases or 300 bases for ordering as a gBlock or eBlock, respectively. This input should be used in conjunction with *us* and/or *ds* to specify which upstream and downstream sequences should be used to meet the length requirement. The option ‘exc’ in *ds* has a long sequence of bases for appending.

1.6.4 Examples

First Steps:

1. *Download, Import, and Initialize ctRSD-simulator-2.0*
2. *Download ctRSD Domains List*
3. Use the example below for guidance

```
# importing simulator
import sys
```

(continues on next page)

(continued from previous page)

```
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

filepath = '//File Path//ctRSD_domains_list.xls'

# use the experimental nomenclature to specify the sequence you want
Gate_seq = model.ctRSD_seq_compile('G{u1,w2r}',filepath)

Gate_seq = model.ctRSD_seq_compile('G{u1,w2r}',filepath,Rz='R3') # specifying an␣
↪alternative ribozyme sequence

Input_seq = model.ctRSD_seq_compile('I{u1}',filepath)

Fuel_seq = model.ctRSD_seq_compile('F{w1}',filepath)

AG_seq = model.ctRSD_seq_compile('AG{u3.u1,w2r}',filepath)
```

Example Script for sequences in the 2022 Science Advances paper can be found here

Example Script for diverse gate sequences can be found here

Example Script for additional component sequences can be found here

Example script saving sequence to an Excel file can be found here

1.7 Troubleshooting

This page offers some useful tips to troubleshoot based off some of the problems we encountered when building the simulator.

1.7.1 Accessing Variables in the Simulator

In order to troubleshoot a script utilizing the simulator, you may want to access different variables that are found in the simulator, but not locally available in your script. This can help with troubleshooting issues by making sure everything is defined as you think it should be.

A detailed description of the matrix setup of the model variables can be found at *Detailed Model Description*.

Note:

Only varibles that have the "self" attachment are available to be accessed from outside the simulator (Ex: self.variable_name).

If you wish to pull out a variable that does not currently have the self attachment, simply add "self." before every occurence of that variable in the simulator.

In order to access any variable with the "self" attachment (Example Below):

1. The bulk of the simulator's code (excluding the rate equations function) exists in the *RSDs_sim()* Python class. Therefore, as you would to run a simulation, you must first instantiate an object of the RSD_sim class. Details to do this can be found *here*.

2. Assuming the model is downloaded, imported, and instantiated accessing a variable is very similar to calling one of the functions. You simply call the variable name with the name of the class object's attachment. For example, if you defined the model instantiation as "model", then the format for accessing a variable would be "model.variable_name".

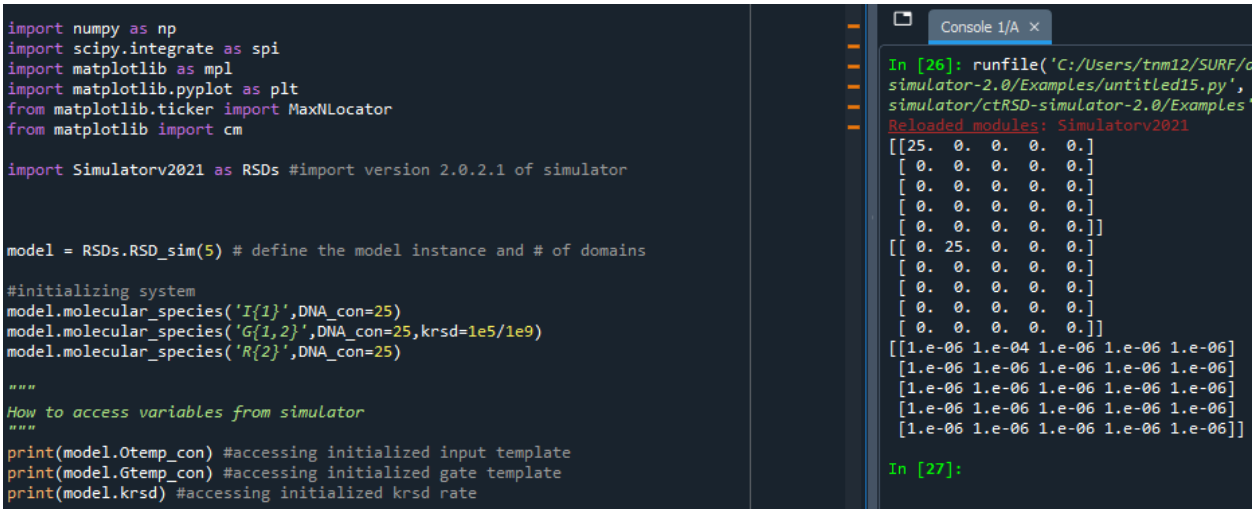

Fig. 10: **How to access variables present in simulator**

Critical values to check for customized simulations include:

The matrices holding DNA template concentrations
These are specified as *model.[name]temp_con*, where [name] can be G, O, TG, F, AG, CG

The matrices holding initial concentrations if these were changed in *molecular_species()*
These are specified as *model.[name]_ic*, where [name] can be uG, G, GO, O, TG, F, AG, CG, R, S, Q, AGOa, AGOb, AGFb, CGOa, CGOb

The matrices holding rate constants if these were changed in *global_rate_constants()* or *molecular_species()*
These are specified as *model.[name]*, where [name] is the name of the rate constant (ktxnO, krsd, krz, kdrd, kssdO, etc)

1.7.2 Not specifying enough domains in RSD_sim() initialization

When initializing the model with *RSD_sim()*, the default number of domains used to initialize the model is 5. If the highest index of any domain of a species you define in *molecular_species()* is greater than 5 then an "index out of bounds" error will appear. For example:

```
# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('G{7,2}',DNA_con=25)
```

will produce an error because index 7 of G{7,2} exceeds the default number of domains (5) specified with *RSD_sim()*:

```
line 30, in <module>
model.molecular_species('G{7,2}',DNA_con=25)

File "filepath", line 779, in molecular_species
self.Gtemp_con[gateInd1,gateInd2]=DNA_con

IndexError: index 6 is out of bounds for axis 0 with size 5
```

To fix this, the total number of domains initialized in *RSD_sim()* needs to be at least 7:

```
# create the model instance
model = RSDs.RSD_sim(7) # increasing total number of domains for initialization to 7

# specify species involved in the system
model.molecular_species('G{7,2}',DNA_con=25)
```

1.7.3 Simulations taking a long time

The simulation time increases as the number of domains in the system and the simulated time increases. So to speed up specific simulations you can try to minimize the number of domains necessary to describe the system and specify this minimum number of domains in *RSD_sim()* during initialization. You can also decrease the total simulated time (t_sim).

We primarily saw issues with simulations taking a long time to solve for systems with comparator gates (CG) and additional tips for those specific types or simulations are covered below.

Changing the *smethod* in *simulate* can also speed up simulations but numerical instabilities are also possible.

1.7.4 Speeding Up Comparator Gate Simulations

Due to comparator gate reactions utilizing very disparate time scales, simulations using comparator gates may have a much slower time efficiency than the other reactions. Below are some different options for reducing the simulation time when using comparator gates.

Troubleshooting options:

1. Utilize the BDF solving method. BDF is an implicit method meant for stiff equations, so it can help to process the CG reactions quicker. The solving method used can be changed in *simulate*.
2. If you are looping through the simulator many times, parallelizing these loops can also greatly speed time efficiency. Python offers many different ways to create parallel loops, such as the *multiprocessing* package.

1.7.5 Overriding Rate Constants

Since both *molecular_species* and *global_rate_constants* offer the ability to change the rate constants available in the simulator, you may find one function overriding changes you made in the other.

Troubleshooting Tips (Examples Below):

1. Both functions will only alter rate constants that are specified in the call of the function. If you do not specify a particular rate constant, then neither functions will change it from its most recent value.
2. The order of the changes made by the two functions depends on the order in which the functions are called in the script. Whichever function is called first will have its changes come into effect first. Therefore, if you

call both functions for the same rate constant, or either functions multiple times for the same rate constant, then the last function call will reflect the most recent value of that rate constant.

```
import numpy as np
import scipy.integrate as spi
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from matplotlib import cm

import Simulatorv2021 as RSDs #import version 2.0.2.1 of simulator



model = RSDs.RSD_sim(5) # define the model instance and # of domains


model.global_rate_constants(krsd=1e5/1e9)

model.molecular_species('G{1,2}',DNA_con=25,krsd=1e7/1e9)
```

```
Console 1/A ×
Reloaded modules: Simulatorv2021

In [25]: model.krsd
Out[25]:
array([[0.0001, 0.01  , 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001]])

In [26]:
```

Fig. 11: **global_rate_constants before molecular species**

```
import numpy as np
import scipy.integrate as spi
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from matplotlib import cm

import Simulatorv2021 as RSDs #import version 2.0.2.1 of simulator



model = RSDs.RSD_sim(5) # define the model instance and # of domains


model.molecular_species('G{1,2}',DNA_con=25,krsd=1e7/1e9)

model.global_rate_constants(krsd=1e5/1e9)
```

```
Console 1/A ×

In [23]: model.krsd
Out[23]:
array([[0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001],
       [0.0001, 0.0001, 0.0001, 0.0001, 0.0001]])

In [24]:
```

Fig. 12: **molecular_species before global_rate_constants**

1.8 Simple Examples

Individual scripts for the examples are shown below with links to the scripts on GitHub (the GitHub scripts have additional plotting features).

A Jupyter Notebook of these examples can be found here

1.8.1 Single ctRSD Gate Reaction

Simulating the system below

Useful Features:

- overveiw of setting up a basic simulation

Single ctRSD Gate Reaction Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
```

(continues on next page)

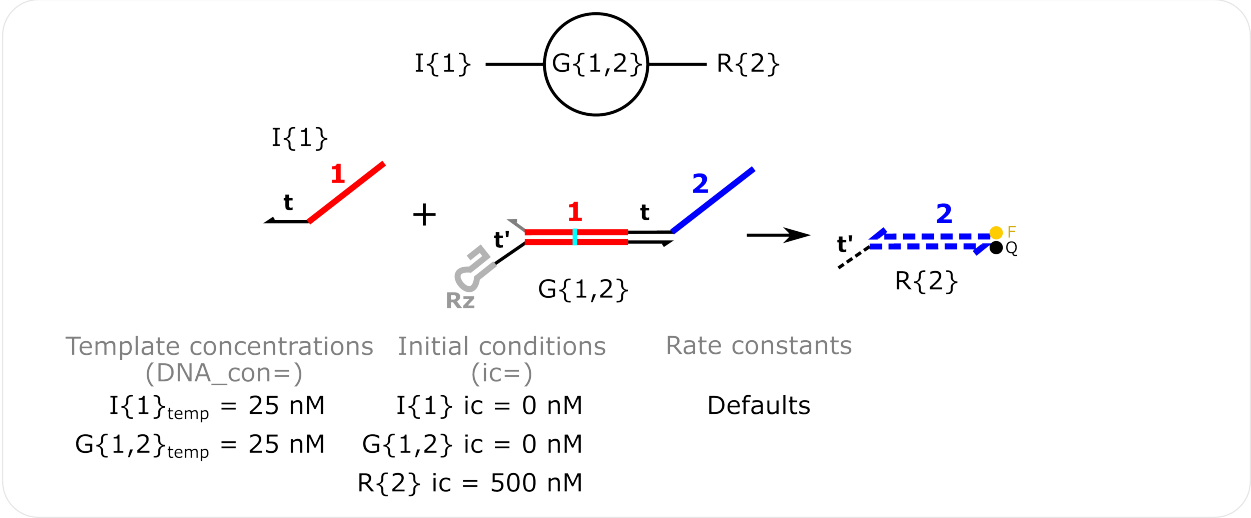

Fig. 13: **Single ctRSD Gate Reaction** Template concentrations are specified with the *DNA_con* input in *molecular_species()* and non-zero initial conditions are specified with the *ic* input in *molecular_species()*.

(continued from previous page)

```
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=25)
model.molecular_species('G{1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}') # S{2} is the output of a reacted reporter (R{2})
I1 = model.output_concentration('I{1}') # concentration of I{1} as a function of time
G12 = model.output_concentration('G{1,2}') # concentration of G{1,2} as a function of␣
↪time
# etc...

# simple plotting code
plt.plot(t_sim,S2)
plt.plot(t_sim,I1,color='red')
plt.plot(t_sim,G12,color='green')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.8.2 Two-Layer Cascade Simulation

Simulating the system below

Useful Features:

- overveiw of setting up a basic simulation
- globally changing rate constants with *global_rate_constants()*

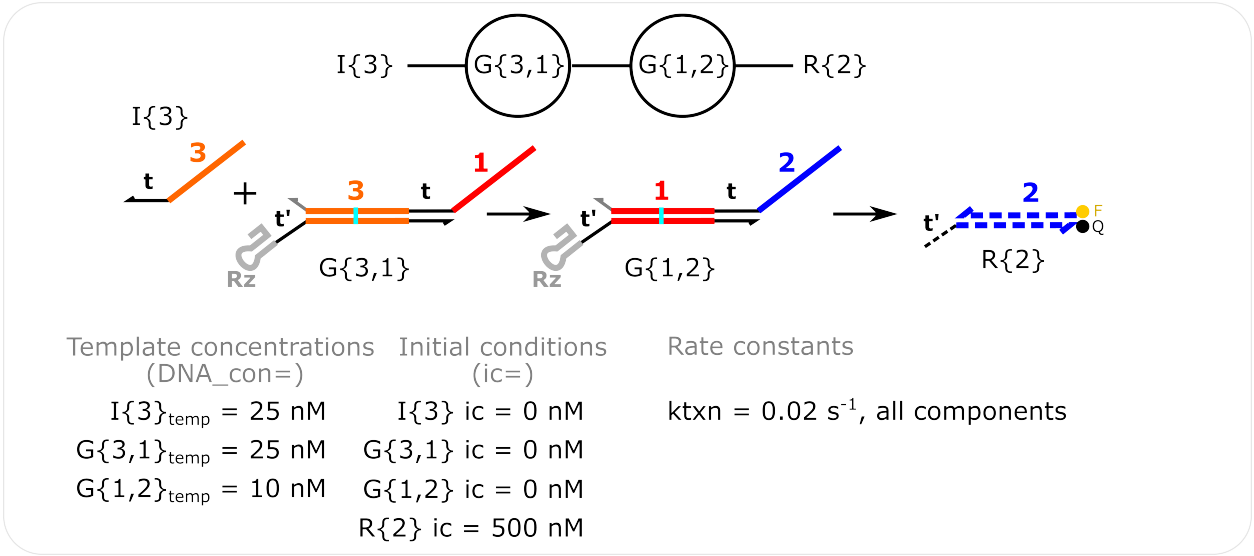

Fig. 14: **Two-Layer Cascade Simulation** Template concentrations are specified with the *DNA_con* input in *molecular_species()* and non-zero initial conditions are specified with the *ic* input in *molecular_species()*. The transcription rate constant (*ktxn*) is changed for all species in *global_rate_constants()*.

Two Layer Cascade Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

model.global_rate_constants(ktxn=0.02) # changing the global transcription rate constant

# specify species involved in the system
model.molecular_species('I{3}',DNA_con=25)
model.molecular_species('G{3,1}',DNA_con=25)
model.molecular_species('G{1,2}',DNA_con=10)
model.molecular_species('R{2}',ic=500)
```

(continues on next page)

(continued from previous page)

```
# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}') # S{2} is the output of a reacted reporter (R{2})
I3 = model.output_concentration('I{3}') # concentration of I{3} as a function of time
G12 = model.output_concentration('G{1,2}') # concentration of G{1,2} as a function of␣
↪time
# etc...

# simple plotting code
plt.plot(t_sim,S2)
plt.plot(t_sim,I3,color='red')
plt.plot(t_sim,G12,color='green')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.8.3 Fan-Out Simulation

Simulating the system below

Useful Features:

- overveiw of setting up a basic simulation
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Fan Out Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing the simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

model.global_rate_constants(ktxn=0.02) # changing the global transcription rate constant

# specify species involved in the system
model.molecular_species('O{4,3}',DNA_con=25)
model.molecular_species('G{3,1}',DNA_con=15)
model.molecular_species('G{3,5}',DNA_con=15,krsd=5e-6) # changing krsd{3,5}
model.molecular_species('R{1}',ic=500,krep=5e-5) # changing krep{1}
```

(continues on next page)

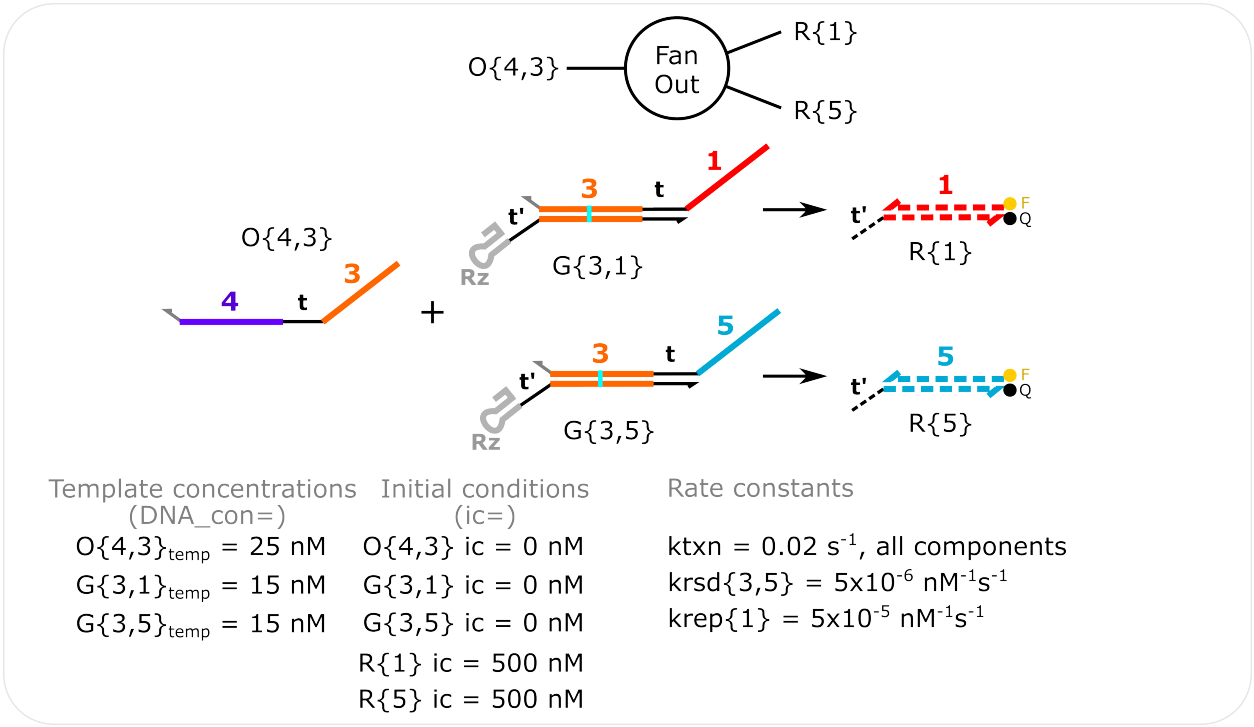

Fig. 15: **Fan-Out Simulation** Template concentrations are specified with the *DNA_con* input in *molecular_species()* and non-zero initial conditions are specified with the *ic* input in *molecular_species()*. The transcription rate constant (*ktxn*) is changed for all species in *global_rate_constants()*. *krsd{3,5}* is changed in *molecular_species()* when G{3,5} is specified. *krep{1} is changed in *molecular_species()* when R{1} is specified.

(continued from previous page)

```
model.molecular_species('R{5}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S1 = model.output_concentration('S{1}') # S{1} is the output of reacted reporter R{1}
S5 = model.output_concentration('S{5}') # S{5} is the output of reacted reporter R{5}
# etc...

# simple plotting code
plt.plot(t_sim,S1,color='red')
plt.plot(t_sim,S5,color='cyan')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.8.4 Fan-Out Fan-In Simulation

Simulating the system below

Useful Features:

- overveiw of setting up a basic simulation
- specifying more than the default number of domains within *RSD_sim()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Fan Out Fan In Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim(8) # increasing # of domains to match highest index in the system

model.global_rate_constants(ktxn=0.02) # changing the global transcription rate constant

# specify species involved in the system
model.molecular_species('O{4,3}',DNA_con=25)
model.molecular_species('G{3,8}',DNA_con=15)
model.molecular_species('G{3,5}',DNA_con=15,krsd=5e-6) # changing krsd{3,5}
model.molecular_species('G{8,2}',DNA_con=10,krev=1e-8) # changing krev{8,2}
```

(continues on next page)

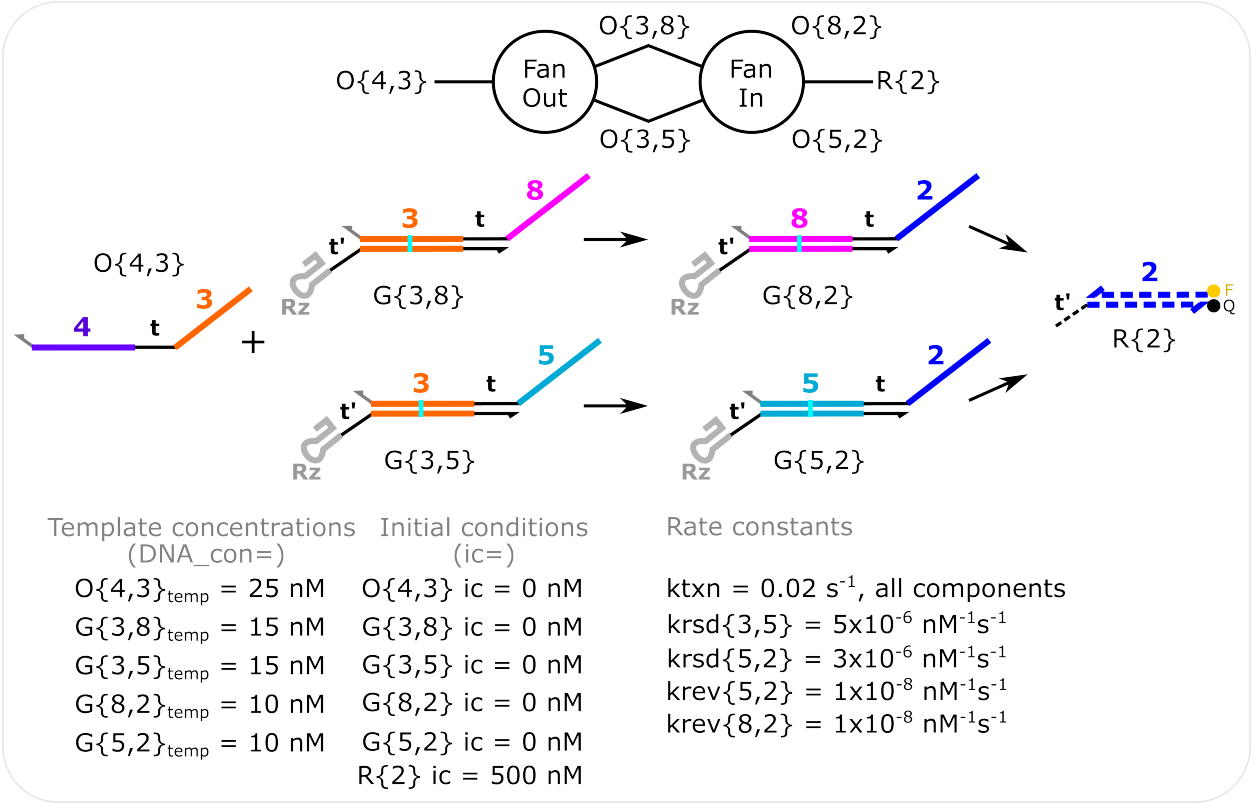

Fig. 16: **Fan-Out Fan-In Simulation** Template concentrations are specified with the *DNA_con* input in *molecular_species()* and non-zero initial conditions are specified with the *ic* input in *molecular_species()*. The transcription rate constant (*ktxn*) is changed for all species in *global_rate_constants()*. Rate constants for individual species are changed in *molecular_species()*. The total domains initialized in the model is increased to 8 when the model is initialized in *RSD_sim()*.

(continued from previous page)

```
model.molecular_species('G{5,2}',DNA_con=10,krsd=3e-6,krev=1e-8) # changing krsd{5,2}␣
↪and krev{5,2}
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')  # S{2} is the output of a reacted reporter (R{2}
↪)
# etc...

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.8.5 Experimental Nomenclature

Simulating the systems below

Useful Features:

- overveiw of setting up a basic simulation
- using expanded experimental nomenclature when specifying components within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*

Note using the expanded nomenclature below does not change anything about the simulation. The simulator ignores the letters before or after the indices. This is merely a way to keep track of the experimental components in a simulation. If the different toeholds have different rate constants, these can be changed when each component is defined in *molecular_species()*, see *Two toehold cascade simulation*

Single ctRSD Gate Reaction with Experimental Nomenclature Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{u1}',DNA_con=25)
model.molecular_species('G{u1,w2r}',DNA_con=25)
```

(continues on next page)

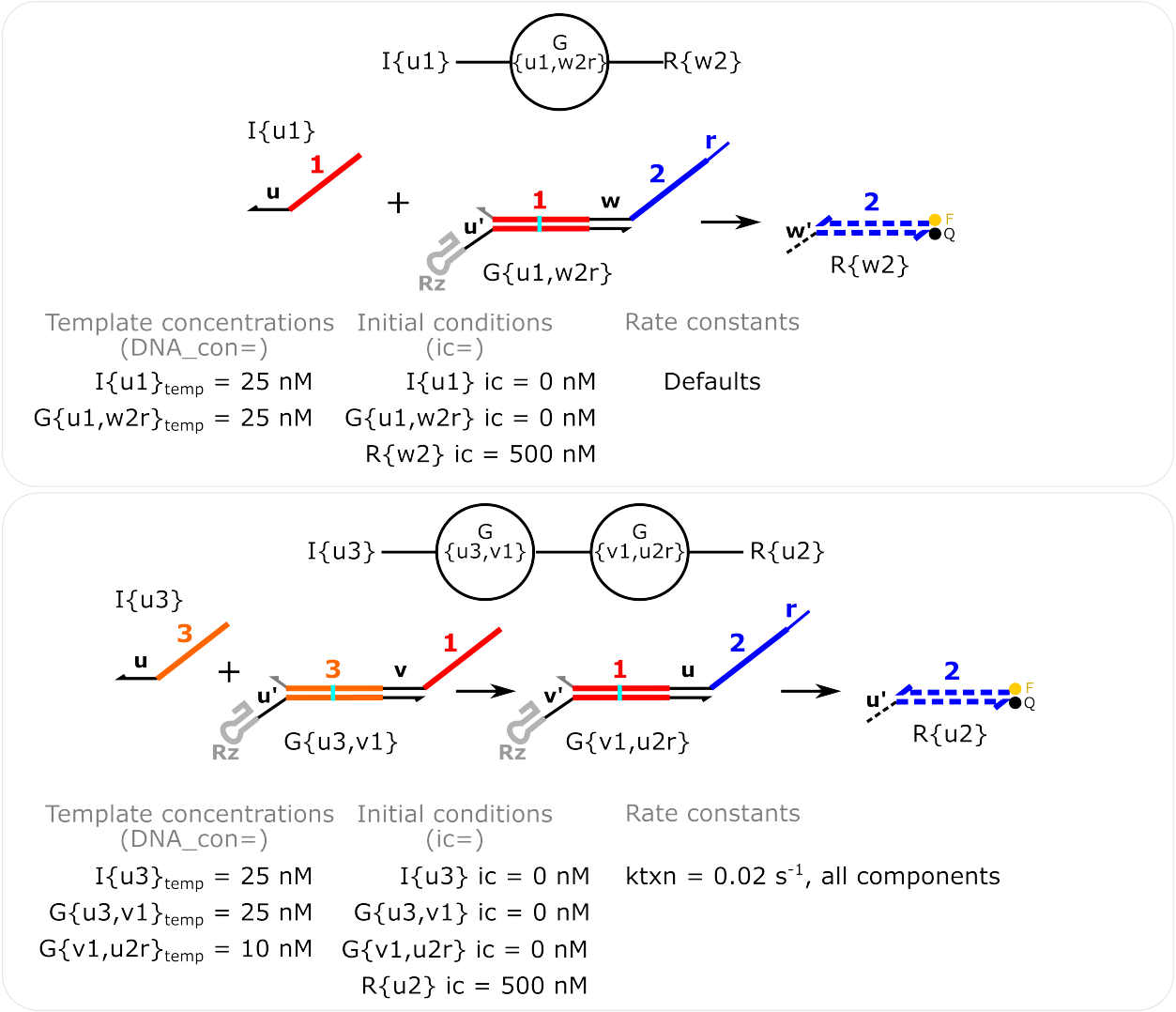

Fig. 17: **Single gate reaction and two layer cascade with experimental nomenclature** In ctRSD circuit experiments different input and output toeholds are often used. This, and the inclusion of other additional domains, leads to an expanded nomenclature compared to what is used in the simulator. Experimentalists may want to use the expanded nomenclature in their simulations to keep track of how circuits are linked together. The simulator allows for this expanded nomenclature by ignoring any letters before or after the input-output indices of a component. The code below highlights this feature.

(continued from previous page)

```
model.molecular_species('R{w2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{w2}')  # S{w2} is the output of a reacted reporter (R
↪{w2})
# etc...

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

Two Layer Simulation with Experimental Nomenclature Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

model.global_rate_constants(ktxn=0.02) # changing the global transcription rate constant

# specify species involved in the system
model.molecular_species('I{u3}',DNA_con=25)
model.molecular_species('G{u3,v1}',DNA_con=25)
model.molecular_species('G{v1,u2r}',DNA_con=10)
model.molecular_species('R{u2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # simulating from 0 to 3 hours with 1000 increments␣
↪(*3600 converts to seconds)
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{u2}')
# etc...

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('Concentration (nM)')
```

1.9 Advanced Simulator Features

Note:

All following examples are important features of the ctRSD-simulator-2.0.

Note the code blocks illustrate how to set examples up for a single set of initial conditions. The Python scripts on GitHub work through testing many different initial conditions, *i.e.*, changing which inputs are present.

1.9.1 Discontinuous Simulation

Discontinuous Simulation provides guidance for the usage of discontinuous feature of ctRSD-simulator-2.0. This feature allows the user to run a simulation for a specific amount of time using any desired conditions, and then apply a different set of conditions for the next time span, until the end of the simulation. The user has the ability to change the conditions for a given amount of time across the entire length of the simulation as many times as needed. Also, full function of the different features of the simulator are available to be changed for different time spans of the total simulaton time. In the example script, G{1,2} is transcribed alongside a R{2} for 60 min and then the template for I{1} is added.

Note:

The discontinuous feature is a part of the simulate function. More information on using simulate for discontinuous simulations can be found *here*

Useful Features:

- discontinuous simulation feature in *simulate()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species*

Discontinuous Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
```

(continues on next page)

(continued from previous page)

```
model.molecular_species('G{1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

# simulating the model for 1 hour
t_sim = np.linspace(0,1,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# adding the I{1} template to the model
model.molecular_species('I{1}',DNA_con=25)

# continue simulation with I{1} template added for 3 more hours
t_sim2 = np.linspace(t_sim[-1]/3600,4,1001)*3600 # seconds
model.simulate(t_sim2,iteration=2) #must specify it is second iteration

# pulling out the reporter concentration for plotting
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

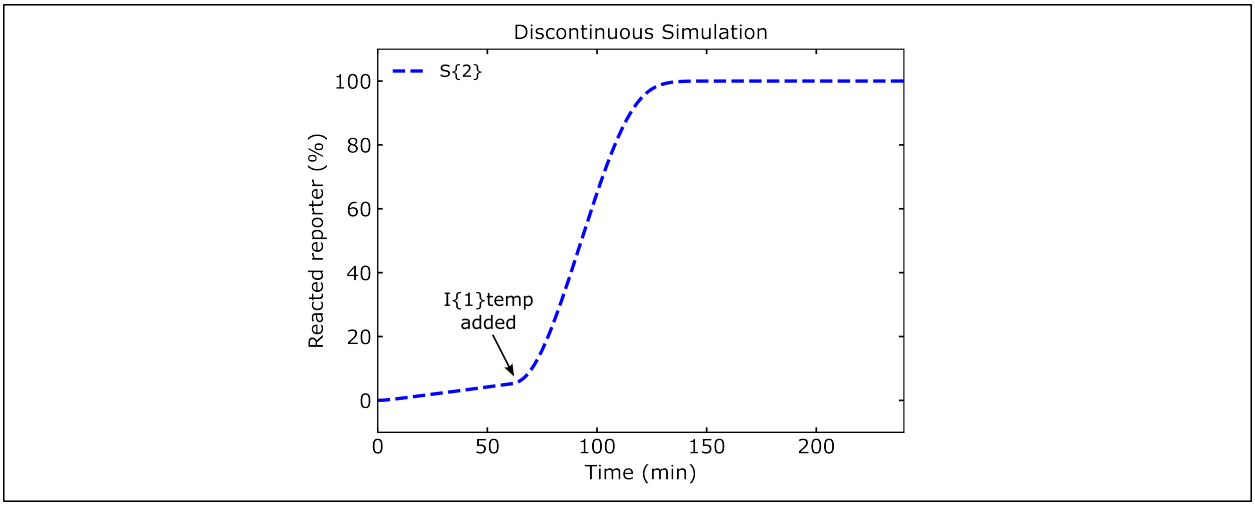

Fig. 18: **Discontinuous Simulation**

1.9.2 Degradation Simulation

Degradation Simulation shows the ability to use *global_rate_constants* to raise degradation rates from their 0 default to initialize degradation reactions in a system.

global_rate_constants() gives the user the ability to change all degradation rates at once using "kdeg" as an argument, to just change degradation rates for single stranded species,"kssd," double stranded species,"kdsd," or RNA:DNA hyrbids,"kdrd," and finally to change the degrdation rates for any given individual species.

The following two figures change all degradation rates simultaneously.

Useful Features:

- incorporating degradation reactions
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Degredation Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

#globally changes all degradation rates
model.global_rate_constants(kdeg=0.001)

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=25)
model.molecular_species('G{1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

The following degredation example simulates a system with degradation rates where single stranded species, double stranded species, and RNA in RNA:DNA hyrbids are all independently changed.

Degradation Simulation with changing groups of rates Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator
```

(continues on next page)

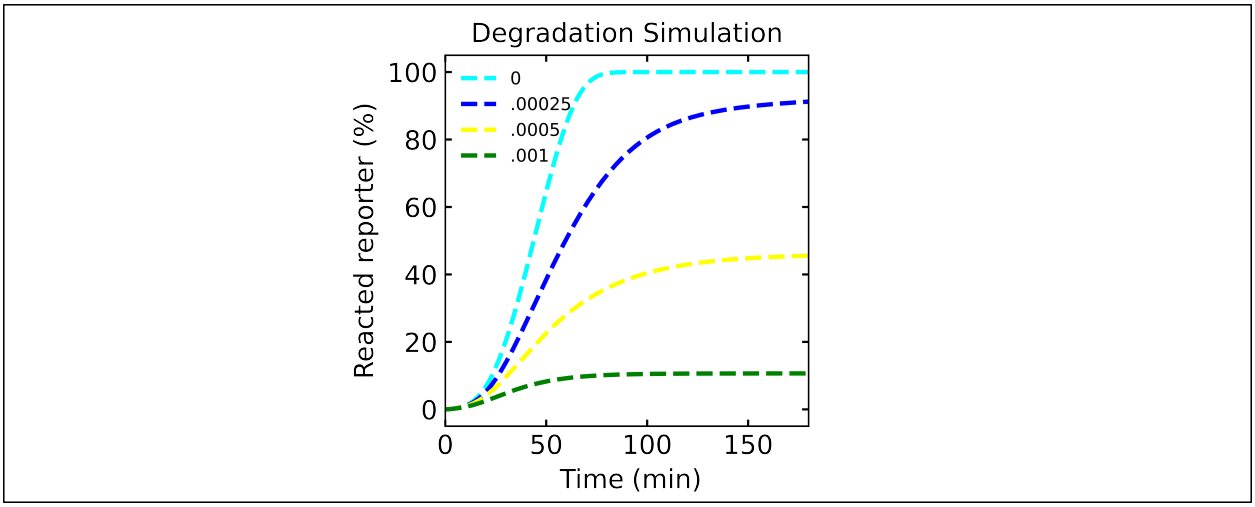

Fig. 19: **Degradation Simulation**

(continued from previous page)

```
# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# globally changes specific types of degradation rates (below are some example commands)
model.global_rate_constants(kssd=0.001) # to change only ssRNA degradation rates
# model.global_rate_constants(kdsd=0.001) # to change only dsRNA degradation rates
# model.global_rate_constants(kdrd=0.001) # to change only RNA in RNA:DNA complex␣
↪degradation rates
# model.global_rate_constants(kssd=0.001,kdsd=0.001,kdrd=0.001) # to change all three␣
↪above together

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=25)
model.molecular_species('G{1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')
I1 = model.output_concentration('I{1}')
G12 = model.output_concentration('G{1,2}')
O12 = model.output_concentration('O{1,2}')

# simple plotting code
plt.subplot(2,4,1)
plt.plot(t_sim,I1,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')

plt.subplot(2,4,2)
```

(continues on next page)

(continued from previous page)

```
plt.plot(t_sim,G12,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')

plt.subplot(2,4,3)
plt.plot(t_sim,O12,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')

plt.subplot(2,4,4)
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

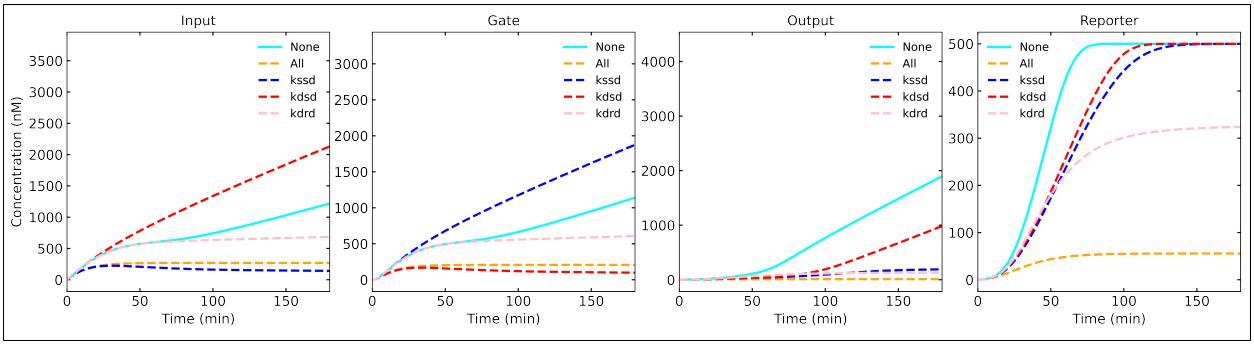

Fig. 20: **Degradation Simulation (Changing Groups of Degradation Rates)**

1.9.3 Two-Toehold Cascade Simulation

This simulation demonstrates a 4-layer cascade in which gates of every other layer have different strand displacement rate constants. This mimics a system with two input-output toeholds that alternate between layers and have different rate constants.

Useful Features:

- using expanded experimental nomenclature when specifying components within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Two Toehold Sim Python Script can be found here

```
'''
This example shows just the 4-layer cascade
'''

# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
```

(continues on next page)

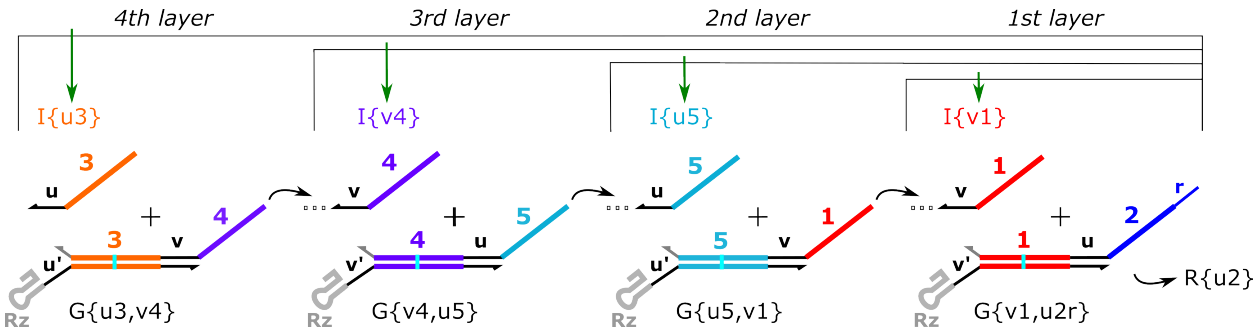

Fig. 21: **Two-Toehold Simulation**

(continued from previous page)

```
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

#globally changing transcription rate constant
model.global_rate_constants(ktxn=0.0075)

# specify species involved in the system
model.molecular_species('I{u3}',DNA_con=25)

kvth=5e3/1e9 # specifying a faster rate constant for v toeholds
model.molecular_species('G{u3,v4}',DNA_con=25)
model.molecular_species('G{v4,u5}',DNA_con=25,krsd=kvth)
model.molecular_species('G{u5,v1}',DNA_con=25)
model.molecular_species('G{v1,u2r}',DNA_con=25,krsd=kvth)

model.molecular_species('R{u2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.9.4 AND Gate with Fuel Simulation

This simulation shows a basic ctRSD AND gate system, but with fuel added to one of the inputs using *molecular_species*

Useful Features:

- specifying AND gates in *molecular_species()*
- specifying fuel strands in *molecular_species()*
- changing indiviudal rate constants within *molecular_species()*
- changing initial conditions within *molecular_species()*

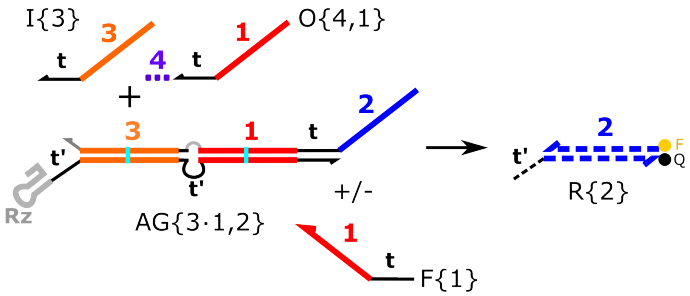

Fig. 22: **AG Fuel Simulation**

AG Fuel Sim Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{3}',DNA_con=25)
model.molecular_species('O{4,1}',DNA_con=1.25) # limiting second input to the AG

model.molecular_species('AG{3.1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

model.molecular_species('F{1}',DNA_con=25)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model
```

(continues on next page)

(continued from previous page)

```
# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

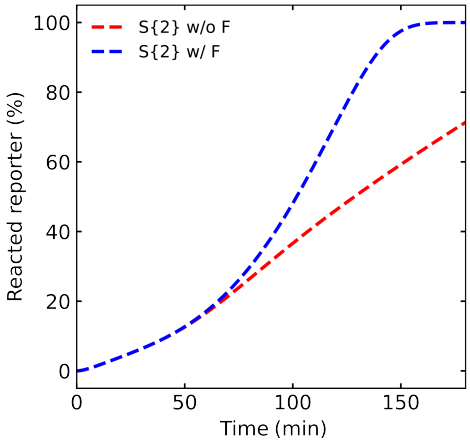

Fig. 23: **AG + Fuel Simulation Results**

1.9.5 Two AND Gate OR Simulation

This simulation shows a 4-input circuit composed of 2 ctRSD AND gates that both produce the same output (ex. (A AND B) OR (C AND D)).

Useful Features:

- specifying AND gates in *molecular_species()*
- changing indiviudal rate constants within *molecular_species()*
- changing initial conditions within *molecular_species()*

Two AND gate OR Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
```

(continues on next page)

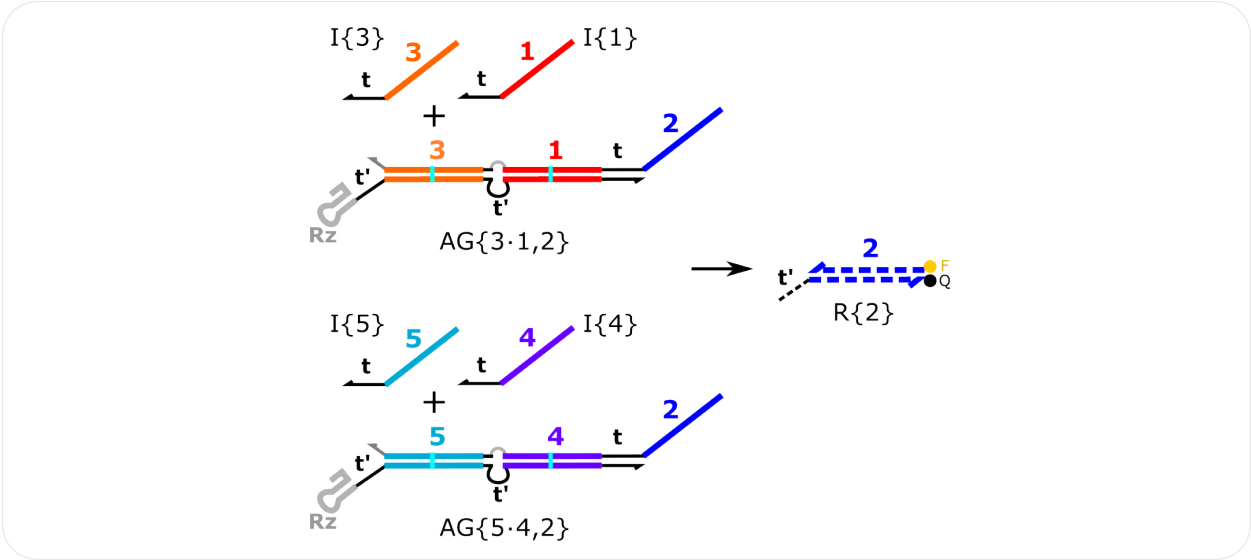

Fig. 24: **Two AND Gate OR Simulation**

(continued from previous page)

```
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=0)
model.molecular_species('I{3}',DNA_con=0)
model.molecular_species('I{4}',DNA_con=25)
model.molecular_species('I{5}',DNA_con=25)

model.molecular_species('AG{5.4,2}',DNA_con=25)
model.molecular_species('AG{3.1,2}',DNA_con=25)
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

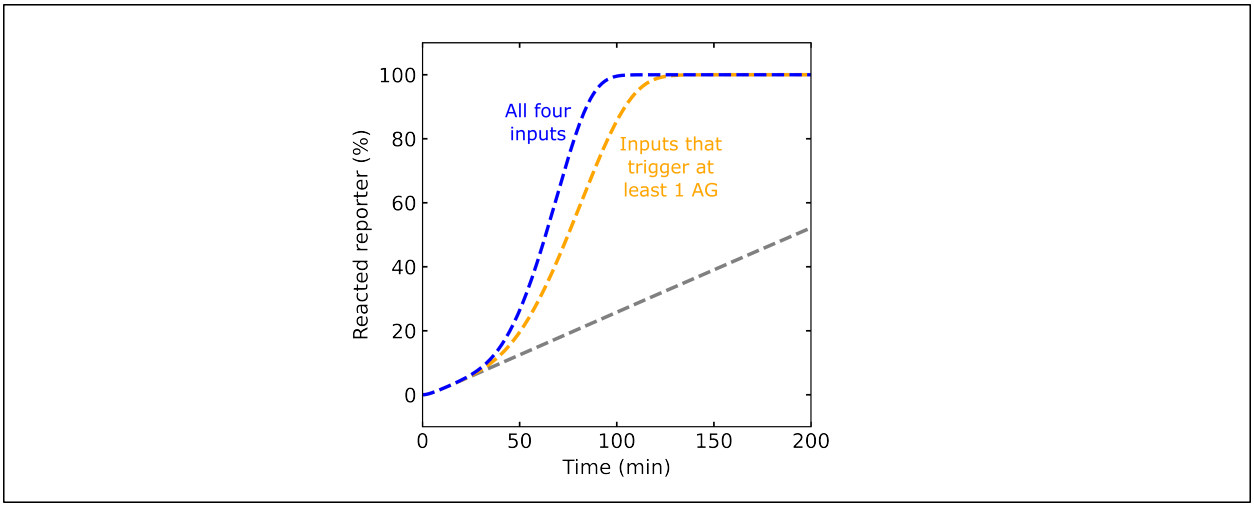

Fig. 25: **Two AND Gate OR Simulation Results**

1.9.6 Thresholding Simulation

Thresholding Simulation showcases a simple thresholding reaction, where a threshold gate is produced to effectively annihlate input produced at a rate below the threshold gate.

Useful Features:

- specifying threshold gates in *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

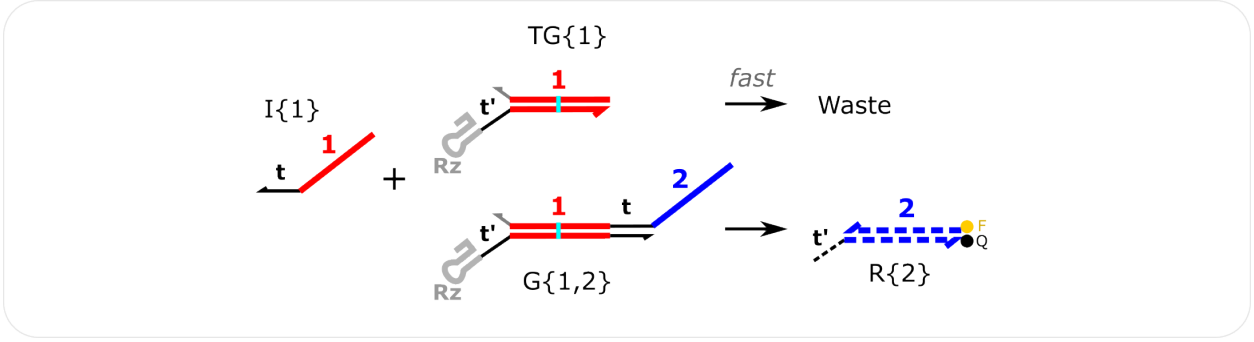

Fig. 26: **Thresholding Simulation**

Thresholding Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator
```

(continues on next page)

(continued from previous page)

```
# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=25)

model.molecular_species('G{1,2}',DNA_con=25)
model.molecular_species('TG{1}',DNA_con=25)

model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,3,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

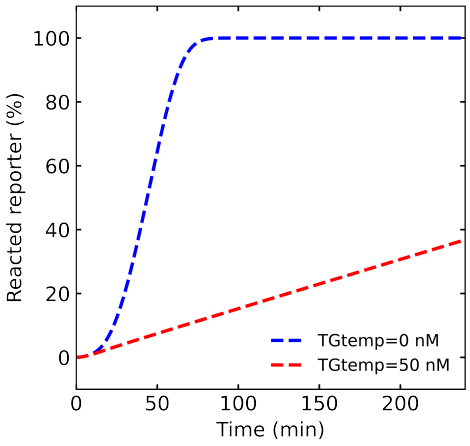

Fig. 27: **Threshold Simulation Results**

1.9.7 Seesaw AND Element Simulations

A simulation of an AND gate using the seesaw gate design from Scaling Up Digital Circuit Computation with DNA Strand Displacement Cascades (Qian and Winfree *Science* 2011).

Useful Features:

- specifying threshold gates in *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*
- changing initial conditions within *molecular_species()*

ctRSD seesaw element simulation

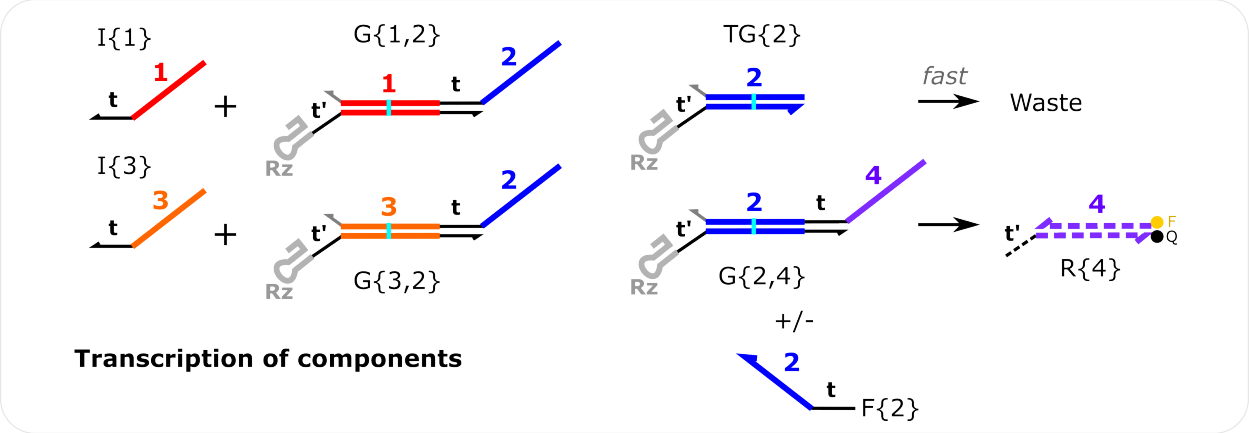

Fig. 28: **ctRSD Seesaw Simulation**

ctRSD Seesaw Simulation Comparison Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# specify species involved in the system
model.molecular_species('I{1}',DNA_con=25)
model.molecular_species('I{3}',DNA_con=25)

model.molecular_species('G{1,2}',DNA_con=16)
model.molecular_species('G{3,2}',DNA_con=16)
```

(continues on next page)

(continued from previous page)

```
model.molecular_species('TG{2}',DNA_con=30)

model.molecular_species('G{2,4}',DNA_con=25)
model.molecular_species('F{2}',DNA_con=25)

model.molecular_species('R{4}',ic=500)

# simulating the model
t_sim = np.linspace(0,4,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S4 = model.output_concentration('S{4}')

# simple plotting code
plt.plot(t_sim,S4,color='purple')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

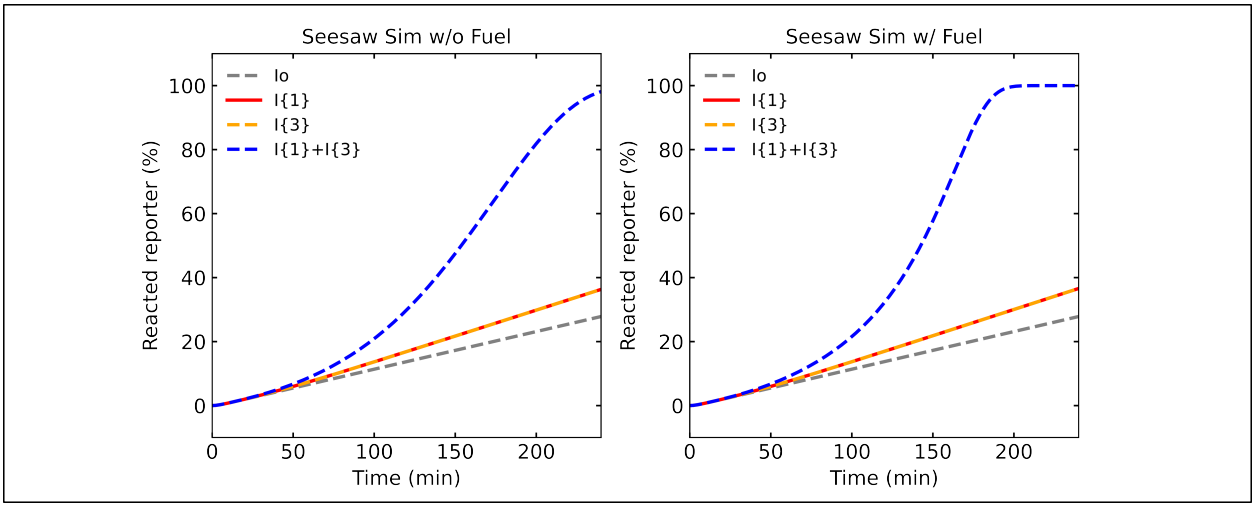

Fig. 29: **ctRSD Seesaw Simulation Results**

DNA seesaw AND element simulation

This simulation mimics a DNA strand displacment reaction by setting DNA_con to 0 for all species and specifying only initial concentrations. Thus, only a fixed amount of each species is present and no transcription occurs.

DNA Seesaw Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
```

(continues on next page)

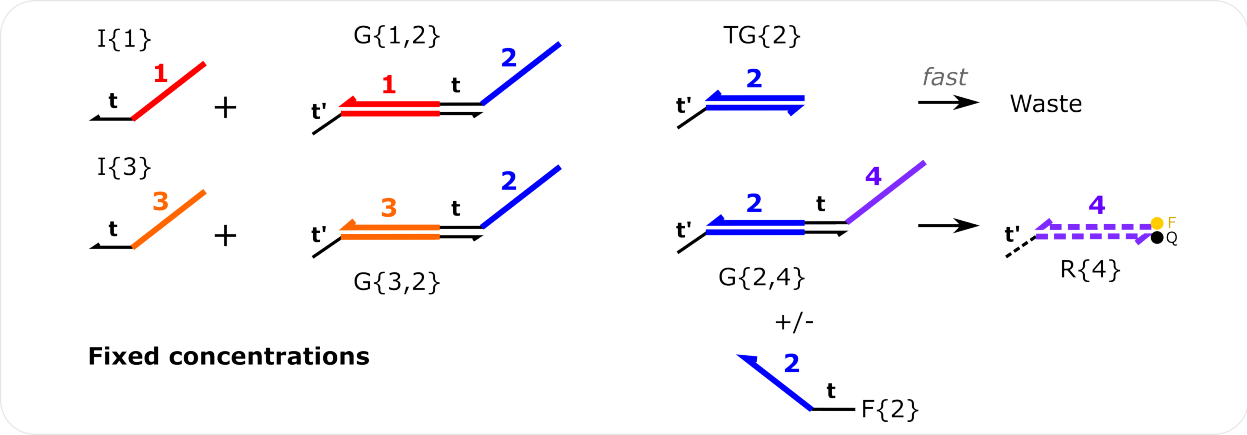

Fig. 30: **DNA Seesaw Simulation**

(continued from previous page)

```
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim() # default # of domains (5 domains)

# changing rate constants to match those used in 2011 DNA computing paper
model.global_rate_constants(krev=5e4/1e9,krsd=5e4/1e9,krsdF=5e4/1e9,kth=2e6/1e9,krep=5e4/
↪1e9)

# specify species involved in the system
# here only initial conditions are used so there is only a fixed concentration of each␣
↪component added
model.molecular_species('I{1}',ic=90)
model.molecular_species('I{3}',ic=90)

model.molecular_species('G{1,2}',ic=100)
model.molecular_species('G{3,2}',ic=100)

model.molecular_species('TG{2}',ic=120)

model.molecular_species('G{2,4}',ic=200)
model.molecular_species('F{2}',ic=200)

model.molecular_species('R{4}',ic=150)

# simulating the model
t_sim = np.linspace(0,4,1001)*3600 # seconds
model.simulate(t_sim) # simulate the model

# pull out the species from the model solution to plot
S4 = model.output_concentration('S{4}')
```

(continues on next page)

(continued from previous page)

```
# simple plotting code
plt.plot(t_sim,S4,color='purple')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

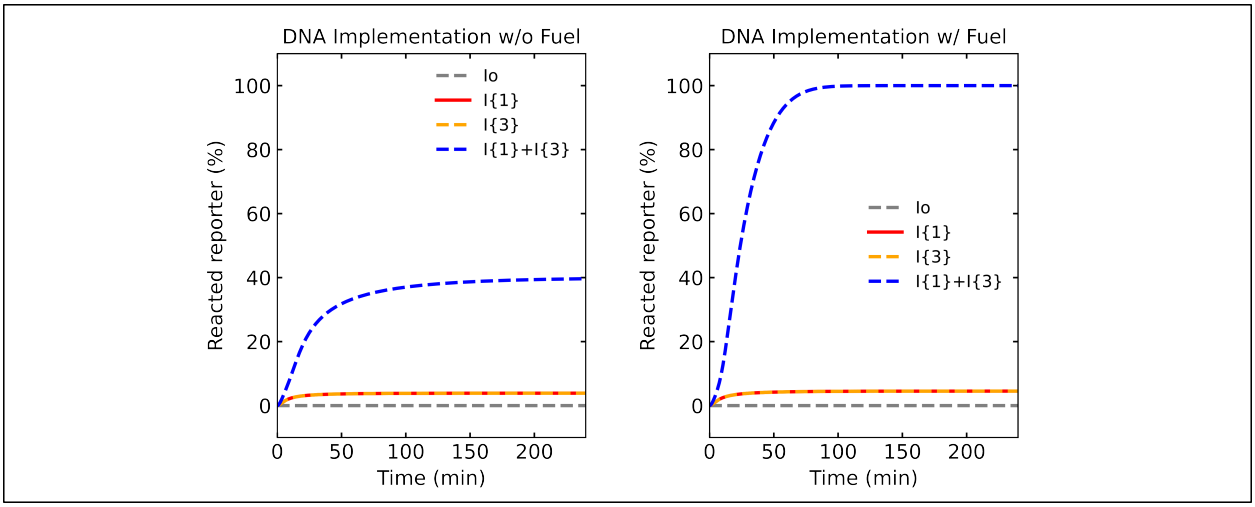

Fig. 31: **DNA Seesaw Simulation Results**

1.9.8 Comparator Gate Simulation

Comparator Gate simulation shows a basic comparator gate reaction. This features shows the ability for a ctRSD circuit that compares the production rate of two inputs and only lets the input with the higher rate of product to the next layer of the circuit. Comparator gates are designed to function like the annihilator gates from Scaling Up Molecular Pattern Recognition with DNA-Based Winner-Take-All Neural Networks (Cherry and Qian *Nature* 2018).

Useful Features:

- specifying more than the default number of domains within *RSD_sim()*
- specifying comparator gates in *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

CG Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
```

(continues on next page)

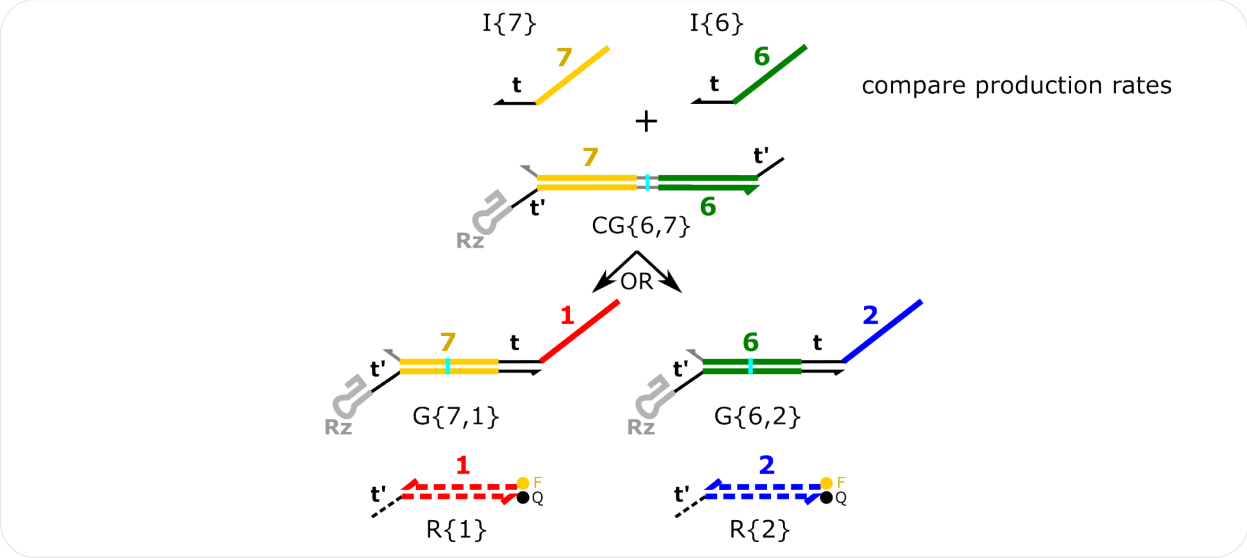

Fig. 32: **CG Simulation**

(continued from previous page)

```
model = RSDs.RSD_sim(7) # specifying the domains as the highest index in the simulated␣
↪system below

# specify species involved in the system
model.molecular_species('I{6}',DNA_con=25)
model.molecular_species('I{7}',DNA_con=10)

model.molecular_species('CG{6,7}',DNA_con=45)

model.molecular_species('G{6,2}',DNA_con=15)
model.molecular_species('G{7,1}',DNA_con=15)

model.molecular_species('R{1}',ic=500)
model.molecular_species('R{2}',ic=500)

# simulating the model
t_sim = np.linspace(0,4,1001)*3600 # seconds
model.simulate(t_sim,smethod='BDF') # simulate the model ('BDF' method can speed up CG␣
↪simulations)

# pull out the species from the model solution to plot
S1 = model.output_concentration('S{1}')
S2 = model.output_concentration('S{2}')

# simple plotting code
plt.plot(t_sim,S1,color='red')
plt.plot(t_sim,S2,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

The CG grid simulation is another example using a basic CG system that shows many more input template concentration

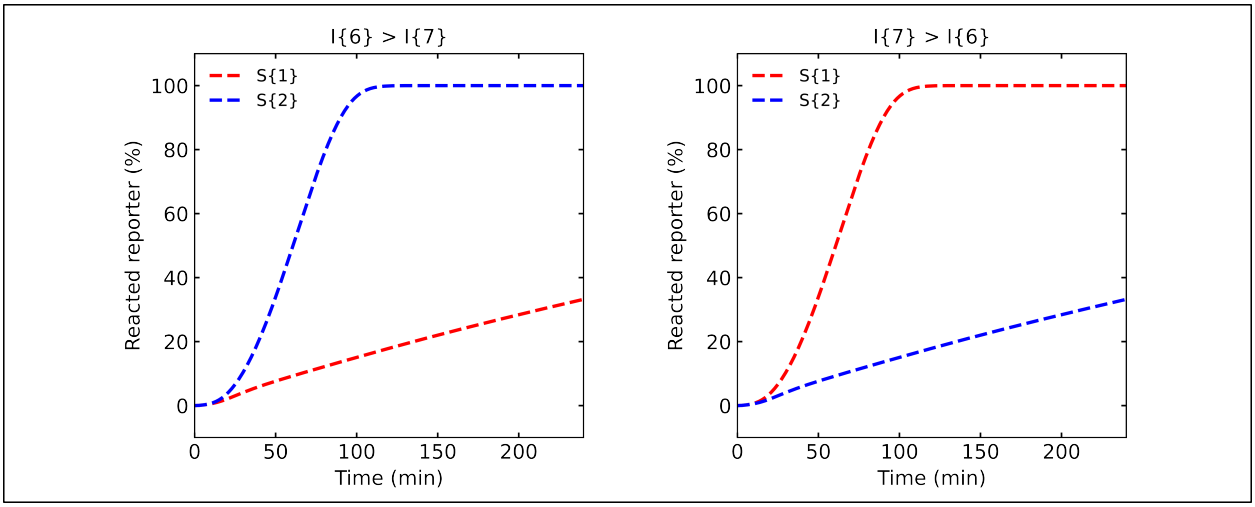

Fig. 33: **CG Simulation Results**

combinations for the two inputs in the system.

CG Grid Simulation Python Script can be found here

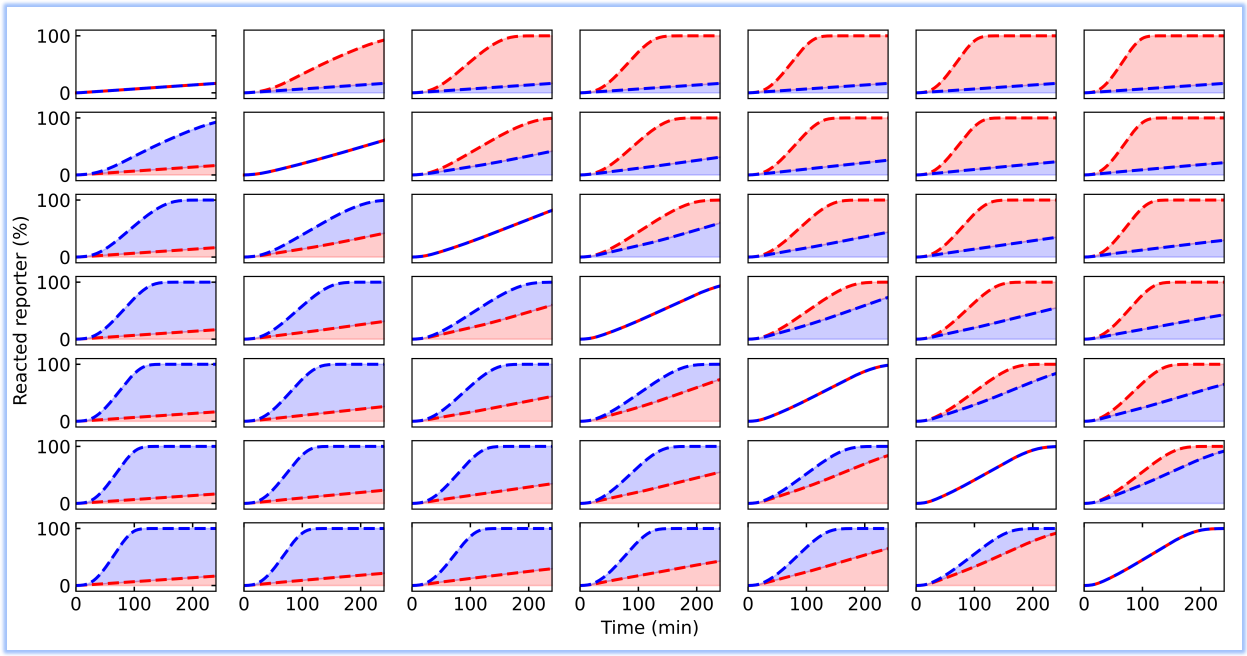

Fig. 34: **CG Grid Simulation Results**

1.9.9 Three Comparator Gate Simulation

Three Comparator Gate Simulation is an extension of the comparator gate simulation that compares the rate of production of three inputs by using multiple comparator gates that encompass the pairwise comparisons of the inputs.

Useful Features:

- simulating relatively large circuits
- specifying more than the default number of domains within *RSD_sim()*
- specifying comparator gates in *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

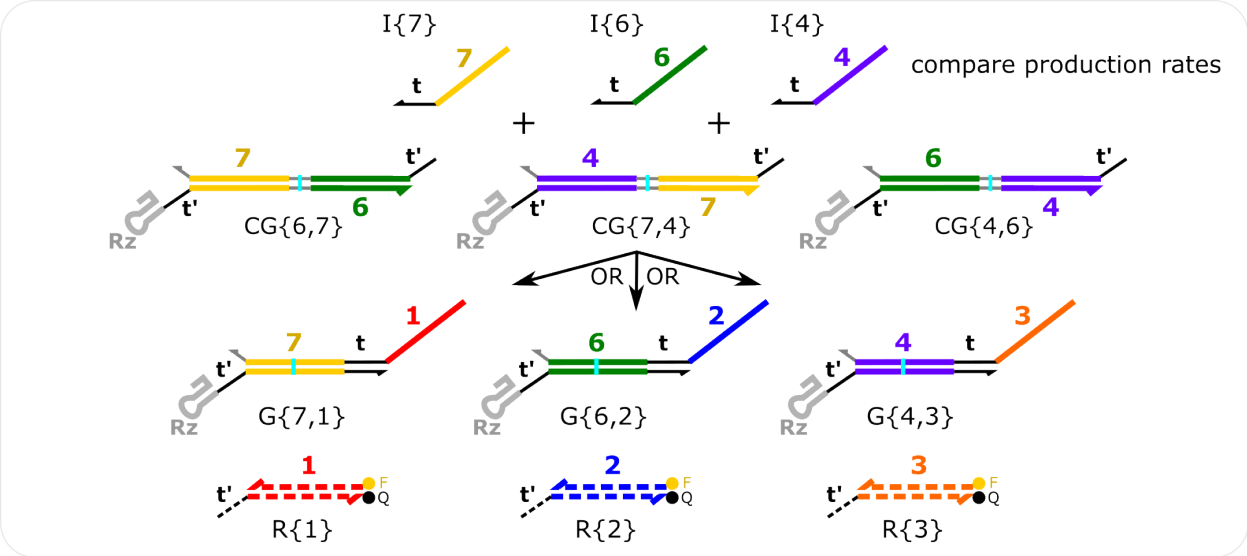

Fig. 35: **Three CG Simulation** The model is set up such that the same input domain cannot be repeated in the same index for two gates. For example, CG{6,7} and CG{4,7} will result in an incorrect result because both gates have domain 7 in the second index. This should be changed to CG{6,7} and CG{7,4} so that domain 7 is in a different index for the two gates.

Three CG Simulation Python Script can be found here

```
# auxiliary packages needed in the script below, e.g., plotting
import numpy as np
import matplotlib.pyplot as plt

# importing simulator
import sys
sys.path.insert(1,'filepath to simulator location on local computer')
import ctRSD_simulator_200 as RSDs # import latest version of the simulator


# create the model instance
model = RSDs.RSD_sim(7) # specifying the domains as the highest index in the simulated␣
↪system below
```

(continues on next page)

(continued from previous page)

```
# increasing the forward strand displacement rate constant for all CG
model.global_rate_constants(krsdCG=5e5/1e9)

# specify species involved in the system
model.molecular_species('I{7}',DNA_con=50)
model.molecular_species('I{6}',DNA_con=30)
model.molecular_species('I{4}',DNA_con=20)

model.molecular_species('CG{6,7}',DNA_con=45)
model.molecular_species('CG{7,4}',DNA_con=45)
model.molecular_species('CG{4,6}',DNA_con=45)

model.molecular_species('G{6,2}',DNA_con=15)
model.molecular_species('G{7,1}',DNA_con=15)
model.molecular_species('G{4,3}',DNA_con=15)

model.molecular_species('R{1}',ic=500)
model.molecular_species('R{2}',ic=500)
model.molecular_species('R{3}',ic=500)

# simulating the model
t_sim = np.linspace(0,6,1001)*3600 # seconds
model.simulate(t_sim,smethod='BDF') # simulate the model ('BDF' method can speed up CG␣
↪simulations)

# pull out the species from the model solution to plot
S1 = model.output_concentration('S{1}')
S2 = model.output_concentration('S{2}')
S3 = model.output_concentration('S{3}')

# simple plotting code
plt.plot(t_sim,S1,color='red')
plt.plot(t_sim,S2,color='blue')
plt.plot(t_sim,S3,color='orange')
plt.xlabel('Time (s)')
plt.ylabel('Concentration (nM)')
```

1.10 2022 Science Advances Paper

Note:

The following examples simulate figures from the 2022 Science Advances paper (Schaffter and Strychalski), Cotranscriptionally encoded RNA strand displacement circuits. See the paper for a detailed description of each experiment and/or simulation.

In that paper a relatively weak output toehold was used with domain 2 for reportering. So in the following scripts, gates with output domain 2 have their reverse strand displacement rate constant decreased compared to the default in *molecular_species()*.

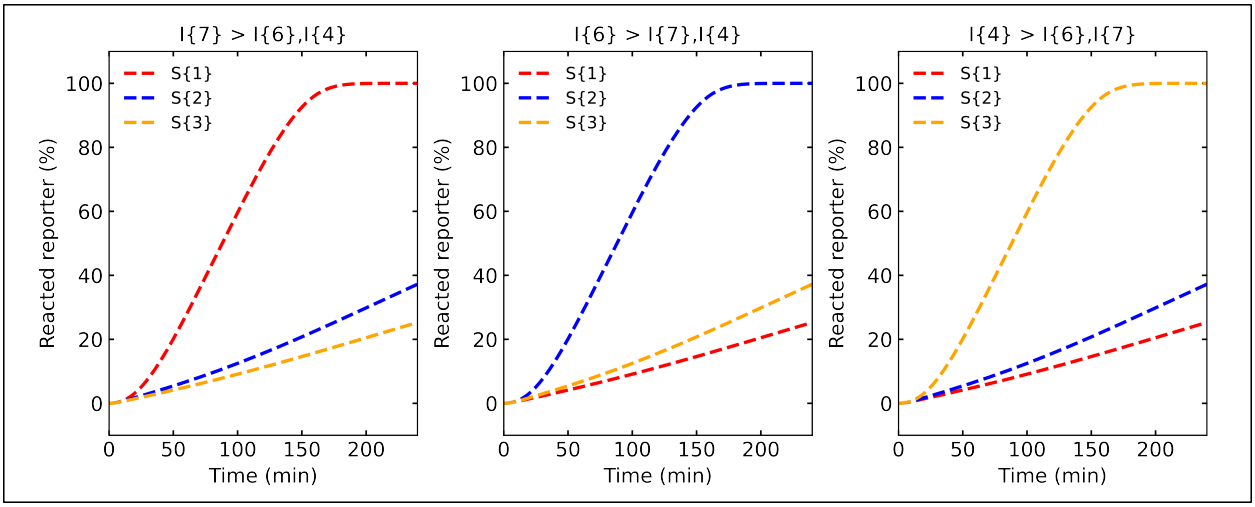

Fig. 36: **Three CG Simulation Results**

1.10.1 Figure 2D

Figure 2D simulates a basic, one-layer ctRSD circuit with different input template concentrations.

Useful Features:

- changing indiviudal rate constants within *molecular_species()*

Figure 2D Python Script can be found here

1.10.2 Figure 4C

Figure 4C simulates a ctRSD OR element.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 4C Python Script can be found here

1.10.3 Figure 4F

Figure 4F simulates a ctRSD AND gate.

Useful Features:

- Specifying AND gates within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 4F Python Script can be found here

1.10.4 Figure 4H

Figure 4H uses fuel reactions to simulate a signal amplification element.

Useful Features:

- Specifying fuel strands within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 4H Python Script can be found here

1.10.5 Figure 5B

Figure 5B simulates a one, two, three, and four layer ctRSD cascades.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 5B Python Script can be found here

1.10.6 Figure 5C

Figure 5C simulates a 4-input ctRSD OR element.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 5C Python Script can be found here

1.10.7 Figure 5D

Figure 5D simulates a two layer cascade of ctRSD AND gates.

Useful Features:

- Specifying AND gates within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 5D Python Script can be found here

1.10.8 Figure 5E

Figure 5E simulates a two layer cascade of a ctRSD OR gate leading to a ctRSD AND gate.

Useful Features:

- Specifying AND gates within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Warning!

The original implementation of the model in ctRSD-simulator-1.0.1 overestimated the reverse rates for FAN-IN circuts. The new model implementation in ctRSD-simulator-2.0 corrected this issue. Therefore, the results can be slightly different between simulators.

More information on the imporved model implementation can be found *here*.

Figure 5E Python Script can be found here

1.10.9 Figure 5F

Figure 5F simulates a two layer cascade of a ctRSD AND gate leading to a ctRSD OR element.

Useful Features:

- Specifying AND gates within *molecular_species()*
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

Figure 5F Python Script can be found here

1.10.10 SI Figure 12

SI Figure 12 simulates an experiment conducted to estimate ribozyme cleavage rate. In the experiment, G{1,2} is initially transcribed for 15 min by itself and given different krz values. After 15 min, the G{1,2} template is degraded (concentration set to 0) and the fraction of the cleaved gate as a function of time is observed in the absence of transcription.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*
- Plotting species other than S{}
- *Discontinuous simulation* feature in simulate()

SA22_SI_Figure12 Python Script can be found here

1.10.11 SI Figure 16

SI Figure 16 simulates mixing fixed concentations of I{1} and G{1,2} with a DNA reporter.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*
- changing indiviudal initial conditions within *molecular_species()*
- *Discontinuous simulation* feature in simulate()

SA22_SI_Figure16 Python Script can be found here

1.10.12 SI Figure 18

SI Figure 18 simulates potential mechanims for uncleaved gates reacting slowly with inputs.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure18 Python Script can be found here

1.10.13 SI Figure 19

Figure 19 simulates the effect of increase reversing rates on one, two, three, and four layer cascades.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure19 Python Script can be found here

1.10.14 SI Figure 26

Figure 26 simulates steric hindrance between the leak products and ctRSD gates could reduce overall leak observed in longer cascades.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure26 Python Script can be found here

1.10.15 SI Figure 27B

Figure 27B simulates lowering the forward strand displacement rate constant for I{4}.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure27B Python Script can be found here

1.10.16 SI Figure 30C

Figure 30C simulates how ctRSD circuit kinetics depend on toehold length and the length of a single-stranded spacer after the self-cleaving ribozyme.

Useful Features:

- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure30C Python Script can be found here

1.10.17 SI Figure 31B

Figure 31B in the SI shows the influence of total template concentration and T7 RNAP concentration on transcriptional load. In terms, of the simulator, this example presents the need to be able to test different transcription rates to find the best rate for a set of data. The simulator uses transcription_calibration for this purpose.

Note:

Click here for full features of the *transcription_calibration()* function.

Useful Features:

- calibration transcription rates using transcription_calibration()
- globally changing rate constants with *global_rate_constants()*
- changing indiviudal rate constants within *molecular_species()*

SA22_SI_Figure31B Python Script can be found here

1.11 2023 ACS Synthetic Biology Paper

Note:

The scripts to simulate the results from the 2023 ACS Synthetic Biology paper (Schaffter et. al.), Design approaches to expand the toolkit for building cotranscriptionally encoded RNA strand displacement circuits can be found here . See the paper for a detailed description of each experiment and/or simulation.

In the paper a relatively weak output toehold (w) was used in some contexts. Gates with w output toeholds had their reverse strand displacement rate constant decreased compared to the default in *molecular_species()*.
