atlas Documentation

Rodrigo Santibáñez

Apr 28, 2020

CONTENTS

Atlas is a small software developed to use simple text files that encode biological networks and write Rule-Based Models (RBMs). Atlas writes rules and others model components for the PySB python package PySB, PMID 23423320. The RBMs could be simulated within PySB with NFsim, PMID 26556387 (within the BioNetGen2 software, PMID 27402907), KaSim (KaSim, PMID 29950016). Models could be exported to text files in *BioNetGen* (BioNetGenLanguage) or *kappa* language (Kappa) for further calibration (BioNetFit, PMID 26556387 or pleione, PMID 31641245) and analysis (sterope for parameter sensibility and alcyone for parameter uncertainty).

CHAPTER

ONE

INSTALLATION

There are two different ways to install Atlas:

1. **Install Atlas natively (Recommended).**

   OR

2. **Clone the Github repository.** If you are familiar with git, Atlas can be cloned and the respective folder added to the python path. Further details are below.

Note: Need Help? If you run into any problems with installation, please visit our chat room: https://gitter.im/glucksfall/pleiades

1.1 Option 1: Install Atlas natively on your computer

The recommended approach is to use system tools, or install them if necessary. To install python packages, you could use pip, or download the package from the python package index.

1. **Install with system tools**

   With pip, you need to execute and Atlas will be installed on `$HOME/.local/lib/python3.6/site-packages` folder or similar.

   ```
   pip3 install atlas_rbm --user
   ```

   If you have system rights, you could install Atlas for all users with

   ```
   sudo -H pip3 install atlas_rbm
   ```

2. **Download from the python package index**

   Alternatively, you could download the package (useful when pip fails to download the package because of lack of SSL libraries) and then install with pip. For instance:

   ```
   wget https://files.pythonhosted.org/packages/ec/ed/
   ↪8b94e0a29f69a24ddb18ba4a4e6463d3ecede308576774e86baf6a84b998/atlas_rbm-1.0.2-
   ↪py3-none-any.whl
   pip3 install atlas_rbm-1.0.2-py3-none-any.whl --user
   ```

   Note: Why Python3?: Atlas is intended to be used with >=python3.4 because python2.7 won't receive further development past 2020, including security updates.

Note: pip, Python, and Anaconda: Be aware which pip you invoque. You could install pip3 with `sudo apt-get install python3-pip` if you have system rights, or install python3 from source, and adding `<python3 path>/bin/pip3` to the path, or linking it in a directory like `$HOME/bin` which is commonly added to the path at login. Also be aware that, if you installed Anaconda, pip could be linked to the Anaconda specific version of pip, which will install Atlas into Anaconda's installation folder. Type `which pip` or `which pip3` to find out the source of pip, and type `python -m site` or `python3 -m site` to find out where is more likely Atlas will be installed.

1.2 Option 2: Clone the Github repository

1. **Clone with git**

   The source code is uploaded and maintained through Github at https://github.com/networkbiolab/atlas. Therefore, you could clone the repository locally, and then add the folder to the `PYTHONPATH`. Beware that you should install the *pysb* package (pysb) and others packages by any means, specially the Jupyter Notebook project (https://jupyter.org).

```
path=/opt/atlas
git clone https://github.com/networkbiolab/atlas $path
echo export PYTHONPATH="\$PYTHONPATH:\$path" >> $HOME/.profile
```

Note: Adding the path to `$HOME/.profile` allows python to find the package installation folder after each user login. Similarly, adding the path to `$HOME/.bashrc` allows python to find the package after each terminal invocation. Other options include setting the `PYTHONPATH` environmental variable in a sh file (see the example folder) or invoke `python3 setup.py clean build install` to install Atlas as it was downloaded from the PyPI server.

CHAPTER

TWO

MODELING

Atlas is a modular software with each script centered in a specific biological network

1. *Metabolic Networks*
2. **Interaction Networks**
    1. *Protein-Protein Interaction Networks*
    2. *Protein-Small compounds Interaction Networks*
    3. **Protein-RNA Interaction Networks**
    4. **RNA-RNA Interaction Networks**
    5. *Transcription Factor-DNA Binding Site Interaction Networks*
    6. *Sigma Factor-Promoter Interaction Networks*
3. *Genome Graphs*

2.1 Metabolic Networks

Metabolic networks have four columns. The first declares a unique name for the enzyme or enzymatic complex; the second declares a unique name for the reaction; the third column lists using comma unique names for substrates; and the last row list using comma unique names for products. To declare metabolites located at the periplasm or extracellular compartments, the user should employ the prefix "PER-" and "EX-", respectively. Use *spontaneous* for non-enzymatic reactions.

Examples:

```
GENE        REACTION        SUBSTRATES        PRODUCTS
spontaneous        LACTOSE-MUTAROTATION        alpha-lactose        beta-lactose
spontaneous        GALACTOSE-MUTAROTATION        alpha-GALACTOSE        beta-GALACTOSE
spontaneous        GLUCOSE-MUTAROTATION        alpha-glucose        beta-glucose
LACY-MONOMER        TRANS-RXN-24        PER-PROTON, PER-alpha-lactose        PROTON, alpha-lactose
LACY-MONOMER        TRANS-RXN-24-beta        PER-PROTON, PER-beta-lactose        PROTON, beta-lactose
LACY-MONOMER        TRANS-RXN-94        PER-PROTON, PER-MELIBIOSE        PROTON, MELIBIOSE
LACY-MONOMER        RXN0-7215        PER-PROTON, PER-CPD-3561        PROTON, CPD-3561
LACY-MONOMER        RXN0-7217        PER-PROTON, PER-CPD-3785        PROTON, CPD-3785
LACY-MONOMER        RXN-17755        PER-PROTON, PER-CPD-3801        PROTON, CPD-3801
BETAGALACTOSID-CPLX        BETAGALACTOSID-RXN        beta-lactose, WATER        beta-GALACTOSE, beta-glucose
```

(continues on next page)

(continued from previous page)

```
BETAGALACTOSID-CPLX        BETAGALACTOSID-RXN-alpha        alpha-lactose,␣
↪WATER        alpha-GALACTOSE, alpha-glucose
BETAGALACTOSID-CPLX        RXN0-5363        alpha-lactose        alpha-ALLOLACTOSE
BETAGALACTOSID-CPLX        RXN0-5363-beta        beta-lactose        beta-ALLOLACTOSE
BETAGALACTOSID-CPLX        ALLOLACTOSE-DEG-alpha        alpha-
↪ALLOLACTOSE        alpha-GALACTOSE, alpha-glucose
BETAGALACTOSID-CPLX        ALLOLACTOSE-DEG-beta        beta-ALLOLACTOSE        beta-
↪GALACTOSE, beta-glucose
BETAGALACTOSID-CPLX        RXN-17726        CPD-3561, WATER        beta-GALACTOSE,␣
↪Fructofuranose
BETAGALACTOSID-CPLX        RXN0-7219        CPD-3785, WATER        beta-GALACTOSE, D-
↪ARABINOSE
GALACTOACETYLTRAN-CPLX        GALACTOACETYLTRAN-RXN-galactose        beta-GALACTOSE,␣
↪ACETYL-COA        6-Acetyl-beta-D-Galactose, CO-A
```

OR

```
GENE        REACTION        SUBSTRATES        PRODUCTS
spontaneous        LACTOSE-MUTAROTATION        alpha-lactose        beta-lactose
spontaneous        GALACTOSE-MUTAROTATION        alpha-GALACTOSE        beta-GALACTOSE
spontaneous        GLUCOSE-MUTAROTATION        alpha-glucose        beta-glucose
lacY        TRANS-RXN-24        PER-PROTON, PER-alpha-lactose        PROTON, alpha-
↪lactose
lacY        TRANS-RXN-24-beta        PER-PROTON, PER-beta-lactose        PROTON, beta-
↪lactose
lacY        TRANS-RXN-94        PER-PROTON, PER-MELIBIOSE        PROTON, MELIBIOSE
lacY        RXN0-7215        PER-PROTON, PER-CPD-3561        PROTON, CPD-3561
lacY        RXN0-7217        PER-PROTON, PER-CPD-3785        PROTON, CPD-3785
lacY        RXN-17755        PER-PROTON, PER-CPD-3801        PROTON, CPD-3801
lacZ        BETAGALACTOSID-RXN        beta-lactose, WATER        beta-GALACTOSE, beta-
↪glucose
lacZ        BETAGALACTOSID-RXN-alpha        alpha-lactose, WATER        alpha-
↪GALACTOSE, alpha-glucose
lacZ        RXN0-5363        alpha-lactose        alpha-ALLOLACTOSE
lacZ        RXN0-5363-beta        beta-lactose        beta-ALLOLACTOSE
lacZ        ALLOLACTOSE-DEG-alpha        alpha-ALLOLACTOSE, WATER        alpha-
↪GALACTOSE, alpha-glucose
lacZ        ALLOLACTOSE-DEG-beta        beta-ALLOLACTOSE, WATER        beta-GALACTOSE,
↪ beta-glucose
lacZ        RXN-17726        CPD-3561, WATER        beta-GALACTOSE, Fructofuranose
lacZ        RXN0-7219        CPD-3785, WATER        beta-GALACTOSE, D-ARABINOSE
lacA        GALACTOACETYLTRAN-RXN-galactose        beta-GALACTOSE, ACETYL-
↪COA        6-Acetyl-beta-D-Galactose, CO-A
```

OR

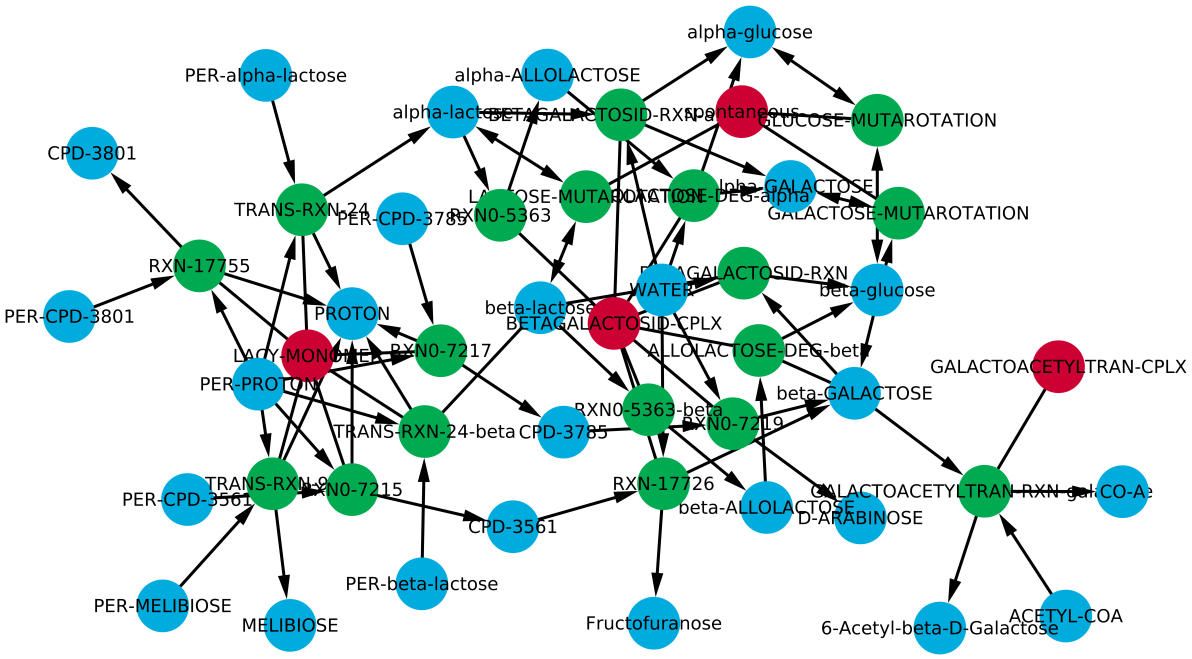
```
GENE        REACTION        SUBSTRATES        PRODUCTS
spontaneous        LACTOSE-MUTAROTATION        alpha-lactose        beta-lactose
spontaneous        GALACTOSE-MUTAROTATION        alpha-GALACTOSE        beta-GALACTOSE
spontaneous        GLUCOSE-MUTAROTATION        alpha-glucose        beta-glucose
lacY        TRANS-RXN-24        PER-PROTON, PER-alpha-lactose        PROTON, alpha-
↪lactose
lacY        TRANS-RXN-24-beta        PER-PROTON, PER-beta-lactose        PROTON, beta-
↪lactose
lacY        TRANS-RXN-94        PER-PROTON, PER-MELIBIOSE        PROTON, MELIBIOSE
lacY        RXN0-7215        PER-PROTON, PER-CPD-3561        PROTON, CPD-3561
lacY        RXN0-7217        PER-PROTON, PER-CPD-3785        PROTON, CPD-3785
```

(continues on next page)

(continued from previous page)

```
lacY        RXN-17755        PER-PROTON, PER-CPD-3801        PROTON, CPD-3801
[lacZ,lacZ,lacZ,lacZ]        BETAGALACTOSID-RXN        beta-lactose,␣
↪WATER        beta-GALACTOSE, beta-glucose
[lacZ,lacZ,lacZ,lacZ]        BETAGALACTOSID-RXN-alpha        alpha-lactose,␣
↪WATER        alpha-GALACTOSE, alpha-glucose
[lacZ,lacZ,lacZ,lacZ]        RXN0-5363        alpha-lactose        alpha-ALLOLACTOSE
[lacZ,lacZ,lacZ,lacZ]        RXN0-5363-beta        beta-lactose        beta-
↪ALLOLACTOSE
[lacZ,lacZ,lacZ,lacZ]        ALLOLACTOSE-DEG-alpha        alpha-ALLOLACTOSE,␣
↪WATER        alpha-GALACTOSE, alpha-glucose
[lacZ,lacZ,lacZ,lacZ]        ALLOLACTOSE-DEG-beta        beta-ALLOLACTOSE,␣
↪WATER        beta-GALACTOSE, beta-glucose
[lacZ,lacZ,lacZ,lacZ]        RXN-17726        CPD-3561, WATER        beta-GALACTOSE,␣
↪Fructofuranose
[lacZ,lacZ,lacZ,lacZ]        RXN0-7219        CPD-3785, WATER        beta-GALACTOSE,␣
↪D-ARABINOSE
[lacA,lacA,lacA]        GALACTOACETYLTRAN-RXN-galactose        beta-GALACTOSE, ACETYL-
↪COA        6-Acetyl-beta-D-Galactose, CO-A
```

Note: Visualization in Cytoscape. Transform the four-columns file into a two-columns file with the helper script "*Expand metabolic network.ipynb*", paste the results in a new file, and import the network into Cytoscape. Colors and arrows remains to the user for customization.



Finally, execute the "*Rules from metabolic network.ipynb*" to obtain the *Rules* to model the defined metabolic network. The complete rule-based model can be found in the lactose folder from the Network Biology Lab GitHub repository here.

```
Rule('LACTOSE_MUTAROTATION',
        met(name = 'alpha_lactose', loc = 'cyt') |
        met(name = 'beta_lactose', loc = 'cyt'),
        Parameter('fwd_LACTOSE_MUTAROTATION', 1),
```

(continues on next page)

(continued from previous page)

```
        Parameter('rvs_LACTOSE_MUTAROTATION', 1))

Rule('GALACTOSE_MUTAROTATION',
        met(name = 'alpha_GALACTOSE', loc = 'cyt') |
        met(name = 'beta_GALACTOSE', loc = 'cyt'),
        Parameter('fwd_GALACTOSE_MUTAROTATION', 1),
        Parameter('rvs_GALACTOSE_MUTAROTATION', 1))

Rule('GLUCOSE_MUTAROTATION',
        met(name = 'alpha_glucose', loc = 'cyt') |
        met(name = 'beta_glucose', loc = 'cyt'),
        Parameter('fwd_GLUCOSE_MUTAROTATION', 1),
        Parameter('rvs_GLUCOSE_MUTAROTATION', 1))

Rule('TRANS_RXN_24',
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'alpha_lactose', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'alpha_lactose', loc = 'cyt'),
        Parameter('fwd_TRANS_RXN_24', 1),
        Parameter('rvs_TRANS_RXN_24', 1))

Rule('TRANS_RXN_24_beta',
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'beta_lactose', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'beta_lactose', loc = 'cyt'),
        Parameter('fwd_TRANS_RXN_24_beta', 1),
        Parameter('rvs_TRANS_RXN_24_beta', 1))

Rule('TRANS_RXN_94',
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'MELIBIOSE', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'MELIBIOSE', loc = 'cyt'),
        Parameter('fwd_TRANS_RXN_94', 1),
        Parameter('rvs_TRANS_RXN_94', 1))

Rule('RXN0_7215', prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'CPD_3561', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'CPD_3561', loc = 'cyt'),
        Parameter('fwd_RXN0_7215', 1),
        Parameter('rvs_RXN0_7215', 1))

Rule('RXN0_7217', prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'CPD_3785', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
```

(continues on next page)

(continued from previous page)

```
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'CPD_3785', loc = 'cyt'),
        Parameter('fwd_RXN0_7217', 1),
        Parameter('rvs_RXN0_7217', 1))

Rule('RXN_17755', prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'per') +
        met(name = 'CPD_3801', loc = 'per') |
        prot(name = 'LACY_MONOMER') +
        met(name = 'PROTON', loc = 'cyt') +
        met(name = 'CPD_3801', loc = 'cyt'),
        Parameter('fwd_RXN_17755', 1),
        Parameter('rvs_RXN_17755', 1))

Rule('BETAGALACTOSID_RXN',
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'beta_lactose', loc = 'cyt') +
        met(name = 'WATER', loc = 'cyt') |
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'beta_GALACTOSE', loc = 'cyt') +
        met(name = 'beta_glucose', loc = 'cyt'),
        Parameter('fwd_BETAGALACTOSID_RXN', 1),
        Parameter('rvs_BETAGALACTOSID_RXN', 1))

Rule('BETAGALACTOSID_RXN_alpha',
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_lactose', loc = 'cyt') +
        met(name = 'WATER', loc = 'cyt') |
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_GALACTOSE', loc = 'cyt') +
        met(name = 'alpha_glucose', loc = 'cyt'),
        Parameter('fwd_BETAGALACTOSID_RXN_alpha', 1),
        Parameter('rvs_BETAGALACTOSID_RXN_alpha', 1))

Rule('RXN0_5363',
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_lactose', loc = 'cyt') |
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_ALLOLACTOSE', loc = 'cyt'),
        Parameter('fwd_RXN0_5363', 1),
        Parameter('rvs_RXN0_5363', 1))

Rule('RXN0_5363_beta',
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'beta_lactose', loc = 'cyt') |
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'beta_ALLOLACTOSE', loc = 'cyt'),
        Parameter('fwd_RXN0_5363_beta', 1),
        Parameter('rvs_RXN0_5363_beta', 1))

Rule('ALLOLACTOSE_DEG_alpha',
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_ALLOLACTOSE', loc = 'cyt') |
        cplx(name = 'BETAGALACTOSID_CPLX') +
        met(name = 'alpha_GALACTOSE', loc = 'cyt'),
        Parameter('fwd_ALLOLACTOSE_DEG_alpha', 1),
        Parameter('rvs_ALLOLACTOSE_DEG_alpha', 1))
```

(continues on next page)

(continued from previous page)

```
119
120 Rule('ALLOLACTOSE_DEG_beta',
121         cplx(name = 'BETAGALACTOSID_CPLX') +
122         met(name = 'beta_ALLOLACTOSE', loc = 'cyt') |
123         cplx(name = 'BETAGALACTOSID_CPLX') +
124         met(name = 'beta_GALACTOSE', loc = 'cyt'),
125         Parameter('fwd_ALLOLACTOSE_DEG_beta', 1),
126         Parameter('rvs_ALLOLACTOSE_DEG_beta', 1))
127
128 Rule('RXN_17726',
129         cplx(name = 'BETAGALACTOSID_CPLX') +
130         met(name = 'CPD_3561', loc = 'cyt') +
131         met(name = 'WATER', loc = 'cyt') |
132         cplx(name = 'BETAGALACTOSID_CPLX') +
133         met(name = 'beta_GALACTOSE', loc = 'cyt') +
134         met(name = 'Fructofuranose', loc = 'cyt'),
135         Parameter('fwd_RXN_17726', 1),
136         Parameter('rvs_RXN_17726', 1))
137
138 Rule('RXN0_7219',
139         cplx(name = 'BETAGALACTOSID_CPLX') +
140         met(name = 'CPD_3785', loc = 'cyt') +
141         met(name = 'WATER', loc = 'cyt') |
142         cplx(name = 'BETAGALACTOSID_CPLX') +
143         met(name = 'beta_GALACTOSE', loc = 'cyt') +
144         met(name = 'D_ARABINOSE', loc = 'cyt'),
145         Parameter('fwd_RXN0_7219', 1),
146         Parameter('rvs_RXN0_7219', 1))
147
148 Rule('GALACTOACETYLTRAN_RXN_galactose',
149         cplx(name = 'GALACTOACETYLTRAN_CPLX') +
150         met(name = 'beta_GALACTOSE', loc = 'cyt') +
151         met(name = 'ACETYL_COA', loc = 'cyt') |
152         cplx(name = 'GALACTOACETYLTRAN_CPLX') +
153         met(name = '_6_Acetyl_beta_D_Galactose', loc = 'cyt') +
154         met(name = 'CO_A', loc = 'cyt'),
155         Parameter('fwd_GALACTOACETYLTRAN_RXN_galactose', 1),
156         Parameter('rvs_GALACTOACETYLTRAN_RXN_galactose', 1))
```

Note: Reversibility of Rules. Atlas writes reversible *Rules* for each reaction declared in the network file. The `Parameter('rvs_RuleName', 1))` must be set to zero to define an irreversible reaction.

Note: Uniqueness of Rule names. Atlas will write *Rules* for unique metabolic reactions. Identical names will be reported for further curation.

Note: Simulation. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes).

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible

`Observables` for metabolites.

2.2 Protein-Protein Interaction Networks

Protein-protein interaction (PPI) interaction networks have two columns. In any order and for any number of components, each column lists using comma the interacting proteins or protein complexes. The user should employ brackets to enclose a list of proteins that are part of a complex.

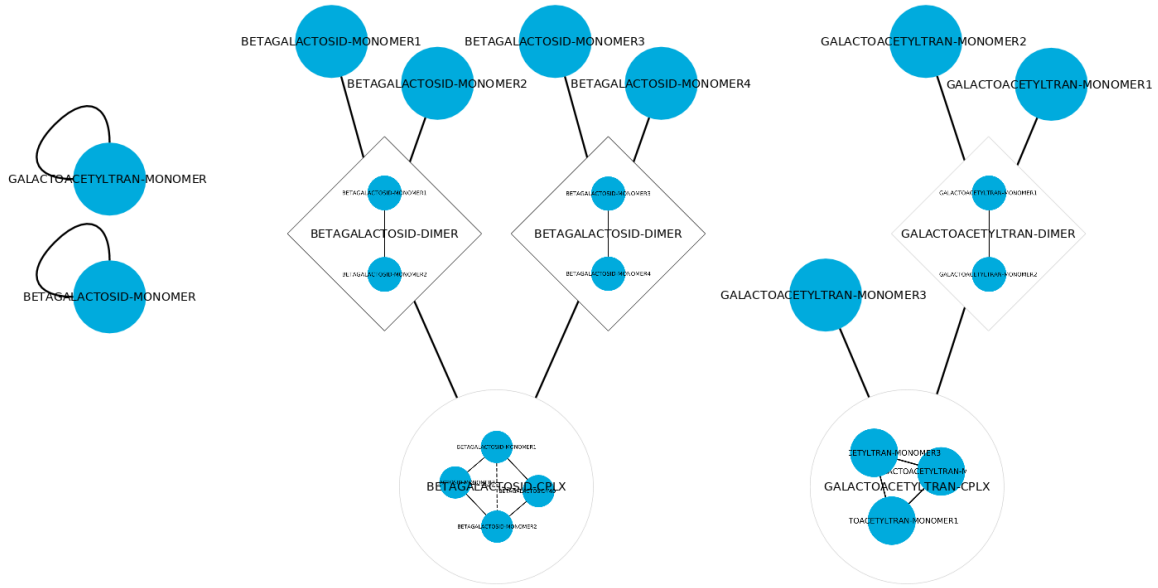
Examples:

```
SOURCE          TARGET
BETAGALACTOSID-MONOMER          BETAGALACTOSID-MONOMER
GALACTOACETYLTRAN-MONOMER          GALACTOACETYLTRAN-MONOMER
```

OR

```
SOURCE          TARGET
BETAGALACTOSID-MONOMER          BETAGALACTOSID-MONOMER
[BETAGALACTOSID-MONOMER, BETAGALACTOSID-MONOMER]          [BETAGALACTOSID-MONOMER,␣
↪BETAGALACTOSID-MONOMER]
GALACTOACETYLTRAN-MONOMER          GALACTOACETYLTRAN-MONOMER
GALACTOACETYLTRAN-MONOMER          [GALACTOACETYLTRAN-MONOMER, GALACTOACETYLTRAN-
↪MONOMER]
```

Note: Visualization in Cytoscape. Cytoscape cannot import hyper-graphs. To do so, Create simple network and right-click to embed a subnetwork in the corresponding node.



Finally, execute the "*Rules from protein-protein.ipynb*" to obtain the *Rules* to model the defined interaction network. The complete rule-based model can be found in the lactose folder from the Network Biology Lab GitHub repository here.

```
Rule('complex_assembly_rule_0',
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = None) +
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = None) |
        prot(name = 'BETAGALACTOSID_MONOMER', up = 1, dw = None) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = 1),
        Parameter('fwd_complex_assembly_rule_0', 1),
        Parameter('rvs_complex_assembly_rule_0', 0))

Rule('complex_assembly_rule_1',
        prot(name = 'BETAGALACTOSID_MONOMER', up = 1, dw = None) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = 1) +
        prot(name = 'BETAGALACTOSID_MONOMER', up = 1, dw = None) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = 1) |
        prot(name = 'BETAGALACTOSID_MONOMER', up = 1, dw = None) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = 2, dw = 1) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = 3, dw = 2) %
        prot(name = 'BETAGALACTOSID_MONOMER', up = None, dw = 3),
        Parameter('fwd_complex_assembly_rule_1', 1),
        Parameter('rvs_complex_assembly_rule_1', 0))

Rule('complex_assembly_rule_2',
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = None) +
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = None) |
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = 1, dw = None) %
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = 1),
        Parameter('fwd_complex_assembly_rule_2', 1),
        Parameter('rvs_complex_assembly_rule_2', 0))

Rule('complex_assembly_rule_3',
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = None) +
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = 1, dw = None) %
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = 1) |
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = 1, dw = None) %
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = 2, dw = 1) %
        prot(name = 'GALACTOACETYLTRAN_MONOMER', up = None, dw = 2),
        Parameter('fwd_complex_assembly_rule_3', 1),
        Parameter('rvs_complex_assembly_rule_3', 0))
```

Note: Reversibility of Rules. Atlas writes irreversible *Rules* for each reaction declared in the network file. The `Parameter('rvs_RuleName', 0))` must be set to non-zero to define an reversible reaction.

Note: Uniqueness of Rule names. Atlas will write *Rules* with numbered names. Use only one file to model the many interactions the system has.

Note: Simulation. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes). Manually add the necessary `Monomers` and `Initials` for non-enzymatic proteins.

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible `Observables` for enzymatic proteins. Other observables for proteins should be added manually.

2.3 Protein-Small compounds Interaction Networks

Protein-small compound interaction networks have two columns. Similar to a PPI network, but the user should add the prefix "SMALL-" to encode a small compound that interacts with the protein or protein complex.

Examples:

```
SOURCE        TARGET
PER-araF        SMALL-PER-alpha-L-arabinofuranose
PER-araF        SMALL-PER-beta-L-arabinofuranose
PER-araF        SMALL-PER-alpha-L-arabinopyranose
PER-araF        SMALL-PER-beta-L-arabinopyranose
[crp, crp]        SMALL-CAMP
[crp, SMALL-CAMP, crp]        SMALL-CAMP
[lacI, lacI]        SMALL-ALLOLACTOSE
[lacI, SMALL-ALLOLACTOSE, lacI]        SMALL-ALLOLACTOSE
[araG, araG]        SMALL-ATP
araC        SMALL-alpha-L-arabinopyranose
[araC, araC]        SMALL-alpha-L-arabinopyranose
[araC, SMALL-alpha-L-arabinopyranose, araC]        SMALL-alpha-L-arabinopyranose
```

Finally, execute the "*Rules from protein-small compounds.ipynb*" to obtain the *Rules* to model the defined interaction network. The complete rule-based model can be found in the arabinose folder from the Network Biology Lab GitHub repository here.

```
Rule('ProtMet_RuleAssembly_1',
        prot(name = 'araF', loc = 'per', met = None, up = None, dw = None) +
        met(name = 'alpha_L_arabinofuranose', loc = 'per', prot = None) |
        prot(name = 'araF', loc = 'per', met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinofuranose', loc = 'per', prot = 1),
        Parameter('fwd_ProtMet_RuleAssembly_1', 1),
        Parameter('rvs_ProtMet_RuleAssembly_1', 1))

Rule('ProtMet_RuleAssembly_2',
        prot(name = 'araF', loc = 'per', met = None, up = None, dw = None) +
        met(name = 'beta_L_arabinofuranose', loc = 'per', prot = None) |
        prot(name = 'araF', loc = 'per', met = 1, up = None, dw = None) %
        met(name = 'beta_L_arabinofuranose', loc = 'per', prot = 1),
        Parameter('fwd_ProtMet_RuleAssembly_2', 1),
        Parameter('rvs_ProtMet_RuleAssembly_2', 1))

Rule('ProtMet_RuleAssembly_3',
        prot(name = 'araF', loc = 'per', met = None, up = None, dw = None) +
        met(name = 'alpha_L_arabinopyranose', loc = 'per', prot = None) |
        prot(name = 'araF', loc = 'per', met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', loc = 'per', prot = 1),
        Parameter('fwd_ProtMet_RuleAssembly_3', 1),
        Parameter('rvs_ProtMet_RuleAssembly_3', 1))

Rule('ProtMet_RuleAssembly_4',
        prot(name = 'araF', loc = 'per', met = None, up = None, dw = None) +
        met(name = 'beta_L_arabinopyranose', loc = 'per', prot = None) |
        prot(name = 'araF', loc = 'per', met = 1, up = None, dw = None) %
        met(name = 'beta_L_arabinopyranose', loc = 'per', prot = 1),
        Parameter('fwd_ProtMet_RuleAssembly_4', 1),
        Parameter('rvs_ProtMet_RuleAssembly_4', 1))

```

(continues on next page)

(continued from previous page)

```
Rule('ProtMet_RuleAssembly_5',
        prot(name = 'crp', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'crp', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'CAMP', loc = 'cyt', prot = None) |
        prot(name = 'crp', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'crp', loc = 'cyt', met = 2, up = 1, dw = None) %
        met(name = 'CAMP', loc = 'cyt', prot = 2),
        Parameter('fwd_ProtMet_RuleAssembly_5', 1),
        Parameter('rvs_ProtMet_RuleAssembly_5', 1))

Rule('ProtMet_RuleAssembly_6',
        prot(name = 'crp', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'CAMP', loc = 'cyt', prot = 2) %
        prot(name = 'crp', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'CAMP', loc = 'cyt', prot = None) |
        prot(name = 'crp', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'CAMP', loc = 'cyt', prot = 2) %
        prot(name = 'crp', loc = 'cyt', met = 3, up = 1, dw = None) %
        met(name = 'CAMP', loc = 'cyt', prot = 3),
        Parameter('fwd_ProtMet_RuleAssembly_6', 1),
        Parameter('rvs_ProtMet_RuleAssembly_6', 1))

Rule('ProtMet_RuleAssembly_7',
        prot(name = 'lacI', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'lacI', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = None) |
        prot(name = 'lacI', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'lacI', loc = 'cyt', met = 2, up = 1, dw = None) %
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = 2),
        Parameter('fwd_ProtMet_RuleAssembly_7', 1),
        Parameter('rvs_ProtMet_RuleAssembly_7', 1))

Rule('ProtMet_RuleAssembly_8',
        prot(name = 'lacI', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = 2) %
        prot(name = 'lacI', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = None) |
        prot(name = 'lacI', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = 2) %
        prot(name = 'lacI', loc = 'cyt', met = 3, up = 1, dw = None) %
        met(name = 'ALLOLACTOSE', loc = 'cyt', prot = 3),
        Parameter('fwd_ProtMet_RuleAssembly_8', 1),
        Parameter('rvs_ProtMet_RuleAssembly_8', 1))

Rule('ProtMet_RuleAssembly_9',
        prot(name = 'araG', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'araG', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'ATP', loc = 'cyt', prot = None) |
        prot(name = 'araG', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'araG', loc = 'cyt', met = 2, up = 1, dw = None) %
        met(name = 'ATP', loc = 'cyt', prot = 2),
        Parameter('fwd_ProtMet_RuleAssembly_9', 1),
        Parameter('rvs_ProtMet_RuleAssembly_9', 1))

Rule('ProtMet_RuleAssembly_10',
        prot(name = 'araC', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'araC', loc = 'cyt', met = None, up = 1, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = None) |
        prot(name = 'araC', loc = 'cyt', met = None, up = None, dw = 1) %
        prot(name = 'araC', loc = 'cyt', met = 2, up = 1, dw = None) %
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = 2),
        Parameter('fwd_ProtMet_RuleAssembly_10', 1),
        Parameter('rvs_ProtMet_RuleAssembly_10', 1))

Rule('ProtMet_RuleAssembly_11',
        prot(name = 'araC', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = 2) %
        prot(name = 'araC', loc = 'cyt', met = None, up = 1, dw = None) +
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = None) |
        prot(name = 'araC', loc = 'cyt', met = 2, up = None, dw = 1) %
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = 2) %
        prot(name = 'araC', loc = 'cyt', met = 3, up = 1, dw = None) %
        met(name = 'alpha_L_arabinopyranose', loc = 'cyt', prot = 3),
        Parameter('fwd_ProtMet_RuleAssembly_11', 1),
        Parameter('rvs_ProtMet_RuleAssembly_11', 1))
```

Note: Reversibility of Rules. Atlas writes reversible *Rules* for each reaction declared in the network file. The `Parameter('rvs_RuleName', 1))` must be set to zero to define an irreversible reaction.

Note: Uniqueness of Rule names. Atlas will write *Rules* with numbered names. Use only one file to model the many interactions the system has.

Note: Simulation. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes). Manually add the necessary `Monomers` and `Initials` for non-enzymatic proteins.

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible `Observables` for enzymatic proteins. Other observables for proteins should be added manually.

2.4 Transcription Factor-DNA Binding Site Interaction Networks

The transcription factor-DNA binding site network represents the physical interaction bewteen proteins and DNA. The network have two columns and for the former network, the first column lists using comma all components of a TF enclosed in brackets (optionally with small compounds) and in the second column declares the DNA binding site. Users should use the prefix "SMALL-" for small compounds and the prefix "BS-" to encode DNA binding sites using unique names. The second type of GRN shows in the first column the RNA polymerase holoenzyme complex (components in brackets) and in the second the promoter. Users should name promoters with the gene name followed by the suffix "-pro#" where # is an integer.

Examples:

```
SOURCE          TARGET
# araBAD and araC
```

(continues on next page)

(continued from previous page)

```
[crp, SMALL-CAMP, crp, SMALL-CAMP]        BS-83-104
[crp, SMALL-CAMP, crp, SMALL-CAMP]        [BS-83-104, BS-araB-pro1]

araC        BS-35-51
araC        BS-56-72
araC        BS-109-125
araC        BS-130-146
araC        BS-267-283

[araC, BS-56-72]        [araC, BS-267-283, BS-araC-pro1]

[araC, SMALL-alpha-L-arabinopyranose]        BS-35-51
[araC, BS-56-72]        SMALL-alpha-L-arabinopyranose
[araC, BS-267-283]        SMALL-alpha-L-arabinopyranose

[araC, SMALL-alpha-L-arabinopyranose, BS-56-72]        [araC, SMALL-alpha-L-
↪arabinopyranose, BS-35-51, BS-araB-pro1]

# araFGH
[araC, SMALL-alpha-L-arabinopyranose]        BS-158-174
[araC, SMALL-alpha-L-arabinopyranose]        BS-137-153
[araC, SMALL-alpha-L-arabinopyranose]        BS-83-99
[araC, SMALL-alpha-L-arabinopyranose]        BS-62-78

[araC, SMALL-alpha-L-arabinopyranose, BS-83-99]        [araC, SMALL-alpha-L-
↪arabinopyranose, BS-62-78, BS-araF-pro1]

# araE
[araC, SMALL-alpha-L-arabinopyranose]        BS-57-73
[araC, SMALL-alpha-L-arabinopyranose]        BS-36-52

[araC, SMALL-alpha-L-arabinopyranose, BS-57-73]        [araC, SMALL-alpha-L-
↪arabinopyranose, BS-36-52, BS-araE-pro1]
```

Finally, execute the "*Rules from tf-tfbs.ipynb*" to obtain the *Rules* to model the defined interaction network. The complete rule-based model can be found in the arabinose folder from the Network Biology Lab GitHub repository here.

```
# [crp, SMALL_CAMP, crp, SMALL_CAMP] interacts with BS_83_104
Rule('TranscriptionFactorMet_AssemblyRule_1',
        prot(name = 'crp', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'CAMP', prot = 3) %
        prot(name = 'crp', dna = None, met = 3, up = 1, dw = None) %
        met(name = 'CAMP', prot = 2) +
        dna(name = 'BS_83_104', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'crp', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'CAMP', prot = 3) %
        prot(name = 'crp', dna = 4, met = 3, up = 1, dw = None) %
        met(name = 'CAMP', prot = 2) %
        dna(name = 'BS_83_104', prot = 4, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_1', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_1', 0))

# [crp, SMALL_CAMP, crp, SMALL_CAMP] interacts with [BS_83_104, BS_araB_pro1]
Rule('TranscriptionFactorMet_AssemblyRule_2',
        prot(name = 'crp', dna = None, met = 2, up = None, dw = 1) %
```

(continues on next page)

(continued from previous page)

```
        met(name = 'CAMP', prot = 3) %
        prot(name = 'crp', dna = None, met = 3, up = 1, dw = None) %
        met(name = 'CAMP', prot = 2) +
        dna(name = 'BS_83_104', prot = None, free = 'True', up = WILD, dw = WILD) %
        dna(name = 'araB', type = 'pro1', prot = None, free = 'True', up = WILD, dw =
↪WILD) |
        prot(name = 'crp', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'CAMP', prot = 3) %
        prot(name = 'crp', dna = 4, met = 3, up = 1, dw = None) %
        met(name = 'CAMP', prot = 2) %
        dna(name = 'BS_83_104', prot = 4, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araB', type = 'pro1', prot = None, free = 'False', up = WILD, dw
↪= WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_2', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_2', 0))

# araC interacts with BS_35_51
Rule('TranscriptionFactorMet_AssemblyRule_3',
        prot(name = 'araC', dna = None, met = None, up = None, dw = None) +
        dna(name = 'BS_35_51', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_35_51', prot = 1, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_3', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_3', 0))

# araC interacts with BS_56_72
Rule('TranscriptionFactorMet_AssemblyRule_4',
        prot(name = 'araC', dna = None, met = None, up = None, dw = None) +
        dna(name = 'BS_56_72', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_56_72', prot = 1, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_4', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_4', 0))

# araC interacts with BS_109_125
Rule('TranscriptionFactorMet_AssemblyRule_5',
        prot(name = 'araC', dna = None, met = None, up = None, dw = None) +
        dna(name = 'BS_109_125', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_109_125', prot = 1, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_5', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_5', 0))

# araC interacts with BS_130_146
Rule('TranscriptionFactorMet_AssemblyRule_6',
        prot(name = 'araC', dna = None, met = None, up = None, dw = None) +
        dna(name = 'BS_130_146', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_130_146', prot = 1, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_6', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_6', 0))

# araC interacts with BS_267_283
Rule('TranscriptionFactorMet_AssemblyRule_7',
        prot(name = 'araC', dna = None, met = None, up = None, dw = None) +
        dna(name = 'BS_267_283', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'BS_267_283', prot = 1, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_7', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_7', 0))

# [araC, BS_56_72] interacts with [araC, BS_267_283, BS_araC_pro1]
Rule('TranscriptionFactorMet_AssemblyRule_8',
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_56_72', prot = 1, free = 'False', up = WILD, dw = WILD) +
        prot(name = 'araC', dna = 2, met = None, up = None, dw = None) %
        dna(name = 'BS_267_283', prot = 2, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araC', type = 'pro1', prot = None, free = 'False', up = WILD, dw
 ↪= WILD) |
        prot(name = 'araC', dna = None, met = None, up = None, dw = 1) %
        dna(name = 'BS_56_72', prot = 2, free = 'False', up = WILD, dw = WILD) %
        prot(name = 'araC', dna = 2, met = None, up = 1, dw = None) %
        dna(name = 'BS_267_283', prot = None, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araC', type = 'pro1', prot = None, free = 'False', up = WILD, dw
 ↪= WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_8', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_8', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_35_51
Rule('TranscriptionFactorMet_AssemblyRule_9',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_35_51', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_35_51', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_9', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_9', 0))

# [araC, BS_56_72] interacts with SMALL_alpha_L_arabinopyranose
Rule('TranscriptionFactorMet_AssemblyRule_10',
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_56_72', prot = 1, free = 'False', up = WILD, dw = WILD) +
        met(name = 'alpha_L_arabinopyranose', prot = None) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        dna(name = 'BS_56_72', prot = 2, free = 'False', up = WILD, dw = WILD) %
        met(name = 'alpha_L_arabinopyranose', prot = 1),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_10', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_10', 0))

# [araC, BS_267_283] interacts with SMALL_alpha_L_arabinopyranose
Rule('TranscriptionFactorMet_AssemblyRule_11',
        prot(name = 'araC', dna = 1, met = None, up = None, dw = None) %
        dna(name = 'BS_267_283', prot = 1, free = 'False', up = WILD, dw = WILD) +
        met(name = 'alpha_L_arabinopyranose', prot = None) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        dna(name = 'BS_267_283', prot = 2, free = 'False', up = WILD, dw = WILD) %
        met(name = 'alpha_L_arabinopyranose', prot = 1),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_11', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_11', 0))

# [araC, SMALL_alpha_L_arabinopyranose, BS_56_72] interacts with [araC, SMALL_alpha_L_
 ↪arabinopyranose, BS_35_51, BS_araB_pro1]
Rule('TranscriptionFactorMet_AssemblyRule_12',
```

(continues on next page)

(continued from previous page)

```
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_56_72', prot = 2, free = 'False', up = WILD, dw = WILD) +
        prot(name = 'araC', dna = 4, met = 3, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_35_51', prot = 4, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araB', type = 'pro1', prot = None, free = 'False', up = WILD, dw
↪= WILD) |
        prot(name = 'araC', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_56_72', prot = 4, free = 'False', up = WILD, dw = WILD) %
        prot(name = 'araC', dna = 4, met = 3, up = 1, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 2) %
        dna(name = 'BS_35_51', prot = None, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araB', type = 'pro1', prot = None, free = 'False', up = WILD, dw
↪= WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_12', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_12', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_158_174
Rule('TranscriptionFactorMet_AssemblyRule_13',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_158_174', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_158_174', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_13', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_13', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_137_153
Rule('TranscriptionFactorMet_AssemblyRule_14',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_137_153', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_137_153', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_14', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_14', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_83_99
Rule('TranscriptionFactorMet_AssemblyRule_15',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_83_99', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_83_99', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_15', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_15', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_62_78
Rule('TranscriptionFactorMet_AssemblyRule_16',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_62_78', prot = None, free = 'True', up = WILD, dw = WILD) |
```

(continues on next page)

(continued from previous page)

```
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_62_78', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_16', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_16', 0))

# [araC, SMALL_alpha_L_arabinopyranose, BS_83_99] interacts with [araC, SMALL_alpha_L_
↪arabinopyranose, BS_62_78, BS_araF_pro1]
Rule('TranscriptionFactorMet_AssemblyRule_17',
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_83_99', prot = 2, free = 'False', up = WILD, dw = WILD) +
        prot(name = 'araC', dna = 4, met = 3, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_62_78', prot = 4, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araF', type = 'pro1', prot = None, free = 'False', up = WILD, dw_
↪= WILD) |
        prot(name = 'araC', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_83_99', prot = 4, free = 'False', up = WILD, dw = WILD) %
        prot(name = 'araC', dna = 4, met = 3, up = 1, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 2) %
        dna(name = 'BS_62_78', prot = None, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araF', type = 'pro1', prot = None, free = 'False', up = WILD, dw_
↪= WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_17', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_17', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_57_73
Rule('TranscriptionFactorMet_AssemblyRule_18',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_57_73', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_57_73', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_18', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_18', 0))

# [araC, SMALL_alpha_L_arabinopyranose] interacts with BS_36_52
Rule('TranscriptionFactorMet_AssemblyRule_19',
        prot(name = 'araC', dna = None, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) +
        dna(name = 'BS_36_52', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_36_52', prot = 2, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_19', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_19', 0))

# [araC, SMALL_alpha_L_arabinopyranose, BS_57_73] interacts with [araC, SMALL_alpha_L_
↪arabinopyranose, BS_36_52, BS_araE_pro1]
Rule('TranscriptionFactorMet_AssemblyRule_20',
        prot(name = 'araC', dna = 2, met = 1, up = None, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 1) %
        dna(name = 'BS_57_73', prot = 2, free = 'False', up = WILD, dw = WILD) +
        prot(name = 'araC', dna = 4, met = 3, up = None, dw = None) %
```

(continues on next page)

(continued from previous page)

```
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_36_52', prot = 4, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araE', type = 'pro1', prot = None, free = 'False', up = WILD, dw
↪= WILD) |
        prot(name = 'araC', dna = None, met = 2, up = None, dw = 1) %
        met(name = 'alpha_L_arabinopyranose', prot = 3) %
        dna(name = 'BS_57_73', prot = 4, free = 'False', up = WILD, dw = WILD) %
        prot(name = 'araC', dna = 4, met = 3, up = 1, dw = None) %
        met(name = 'alpha_L_arabinopyranose', prot = 2) %
        dna(name = 'BS_36_52', prot = None, free = 'False', up = WILD, dw = WILD) %
        dna(name = 'araE', type = 'pro1', prot = None, free = 'False', up = WILD, dw
↪= WILD),
        Parameter('fwd_TranscriptionFactorMet_AssemblyRule_20', 0),
        Parameter('rvs_TranscriptionFactorMet_AssemblyRule_20', 0))
```

Note: Reversibility of reactions. Atlas writes dead *Rules* for each reaction declared in the network file. The `Parameter('fwd_ReactionName', 0))` must be set to non-zero to activate the rule and `Parameter('rvs_ReactionName', 0))` must be set to non-zero to define a reversible reaction.

Note: Simulation. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes).

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible `Observables` for metabolites.

2.5 Sigma Factor-Promoter Interaction Networks

The Sigma Factor-Promoter network have two columns and for the former network, the first column lists using comma all components of a TF enclosed in brackets (optionally with small compounds) and in the second column declares the DNA binding site. Users should use the prefix "SMALL-" for small compounds and the prefix "BS-" to encode DNA binding sites using unique names. The second type of GRN shows in the first column the RNA polymerase holoenzyme complex (components in brackets) and in the second the promoter. Users should name promoters with the gene name followed by the suffix "-pro#" where # is an integer.

Examples:

```
SOURCE          TARGET
# Docking to promoters
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoA-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoB-pro1
# [rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoC-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoD-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoE-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoH-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoN-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-rpoS-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-fliA-pro1
[rpoA, rpoA, rpoB, rpoC, rpoD]          BS-fecI-pro1
```

(continues on next page)

(continued from previous page)

```
[rpoA, rpoA, rpoB, rpoC, rpoE]        BS-rpoD-pro1
[rpoA, rpoA, rpoB, rpoC, rpoE]        BS-rpoE-pro1
[rpoA, rpoA, rpoB, rpoC, rpoE]        BS-rpoH-pro1
[rpoA, rpoA, rpoB, rpoC, rpoE]        BS-rpoN-pro1

[rpoA, rpoA, rpoB, rpoC, rpoH]        BS-rpoA-pro1
[rpoA, rpoA, rpoB, rpoC, rpoH]        BS-rpoD-pro1

[rpoA, rpoA, rpoB, rpoC, rpoN]        BS-rpoA-pro1
[rpoA, rpoA, rpoB, rpoC, rpoN]        BS-rpoD-pro1
[rpoA, rpoA, rpoB, rpoC, rpoN]        BS-rpoH-pro1

[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-fecI-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoA-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoB-pro1
# [rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoC-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoD-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoE-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoH-pro1
[rpoA, rpoA, rpoB, rpoC, rpoS]        BS-rpoN-pro1

[rpoA, rpoA, rpoB, rpoC, fliA]        BS-rpoD-pro1
[rpoA, rpoA, rpoB, rpoC, fliA]        BS-rpoN-pro1
[rpoA, rpoA, rpoB, rpoC, fliA]        BS-fliA-pro1
```

Finally, execute the "*Rules from SigmaFactors x Architecture.ipynb*" to obtain the *Rules* to model the defined interaction network. The complete rule-based model can be found in the sigma folder from the Network Biology Lab GitHub repository here.

```
# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoA_pro1
Rule('docking_1_rpoA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoA', type = 'pro1', prot = None, free = 'True', up = WILD, dw =␣
↪WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw =␣
↪WILD),
        Parameter('fwd_docking_1_rpoA_pro1', 0),
        Parameter('rvs_docking_1_rpoA_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoB_pro1
Rule('docking_2_rpoB_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoB', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoB', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_2_rpoB_pro1', 0),
        Parameter('rvs_docking_2_rpoB_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoD_pro1
Rule('docking_3_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_3_rpoD_pro1', 0),
        Parameter('rvs_docking_3_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoE_pro1
Rule('docking_4_rpoE_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoE', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoE', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_4_rpoE_pro1', 0),
        Parameter('rvs_docking_4_rpoE_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoH_pro1
Rule('docking_5_rpoH_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoH', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoH', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_5_rpoH_pro1', 0),
        Parameter('rvs_docking_5_rpoH_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoN_pro1
Rule('docking_6_rpoN_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoN', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoN', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_6_rpoN_pro1', 0),
        Parameter('rvs_docking_6_rpoN_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoS_pro1
Rule('docking_7_rpoS_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoS', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoS', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_7_rpoS_pro1', 0),
        Parameter('rvs_docking_7_rpoS_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_fliA_pro1
Rule('docking_8_fliA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'fliA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'fliA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_8_fliA_pro1', 0),
        Parameter('rvs_docking_8_fliA_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_fecI_pro1
Rule('docking_9_fecI_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'fecI', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'fecI', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_9_fecI_pro1', 0),
        Parameter('rvs_docking_9_fecI_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoE] interacts with BS_rpoD_pro1
Rule('docking_10_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_10_rpoD_pro1', 0),
        Parameter('rvs_docking_10_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoE] interacts with BS_rpoE_pro1
Rule('docking_11_rpoE_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoE', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoE', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_11_rpoE_pro1', 0),
        Parameter('rvs_docking_11_rpoE_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoE] interacts with BS_rpoH_pro1
Rule('docking_12_rpoH_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoH', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoH', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_12_rpoH_pro1', 0),
        Parameter('rvs_docking_12_rpoH_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoE] interacts with BS_rpoN_pro1
Rule('docking_13_rpoN_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoN', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoE', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoN', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_13_rpoN_pro1', 0),
        Parameter('rvs_docking_13_rpoN_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoH] interacts with BS_rpoA_pro1
Rule('docking_14_rpoA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoH', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoH', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_14_rpoA_pro1', 0),
        Parameter('rvs_docking_14_rpoA_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoH] interacts with BS_rpoD_pro1
Rule('docking_15_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoH', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoH', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_15_rpoD_pro1', 0),
        Parameter('rvs_docking_15_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoN] interacts with BS_rpoA_pro1
Rule('docking_16_rpoA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_16_rpoA_pro1', 0),
        Parameter('rvs_docking_16_rpoA_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoN] interacts with BS_rpoD_pro1
Rule('docking_17_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_17_rpoD_pro1', 0),
        Parameter('rvs_docking_17_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoN] interacts with BS_rpoH_pro1
Rule('docking_18_rpoH_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoH', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoN', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoH', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_18_rpoH_pro1', 0),
        Parameter('rvs_docking_18_rpoH_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_fecI_pro1
Rule('docking_19_fecI_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'fecI', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'fecI', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_19_fecI_pro1', 0),
        Parameter('rvs_docking_19_fecI_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoA_pro1
Rule('docking_20_rpoA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_20_rpoA_pro1', 0),
        Parameter('rvs_docking_20_rpoA_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoB_pro1
Rule('docking_21_rpoB_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoB', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoB', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_21_rpoB_pro1', 0),
        Parameter('rvs_docking_21_rpoB_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoD_pro1
Rule('docking_22_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_22_rpoD_pro1', 0),
        Parameter('rvs_docking_22_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoE_pro1
Rule('docking_23_rpoE_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoE', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoE', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_23_rpoE_pro1', 0),
        Parameter('rvs_docking_23_rpoE_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoH_pro1
Rule('docking_24_rpoH_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoH', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoH', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_24_rpoH_pro1', 0),
        Parameter('rvs_docking_24_rpoH_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, rpoS] interacts with BS_rpoN_pro1
Rule('docking_25_rpoN_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoN', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoS', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoN', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_25_rpoN_pro1', 0),
        Parameter('rvs_docking_25_rpoN_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, fliA] interacts with BS_rpoD_pro1
Rule('docking_26_rpoD_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = None, met = None, up = 4, dw = None) +
```

(continues on next page)

(continued from previous page)

```
        dna(name = 'rpoD', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoD', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_26_rpoD_pro1', 0),
        Parameter('rvs_docking_26_rpoD_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, fliA] interacts with BS_rpoN_pro1
Rule('docking_27_rpoN_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoN', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoN', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_27_rpoN_pro1', 0),
        Parameter('rvs_docking_27_rpoN_pro1', 0))

# [rpoA, rpoA, rpoB, rpoC, fliA] interacts with BS_fliA_pro1
Rule('docking_28_fliA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'fliA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'fliA', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'fliA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_28_fliA_pro1', 0),
        Parameter('rvs_docking_28_fliA_pro1', 0))
```

Note: **Reversibility of reactions**. Atlas writes dead *Rules* for each reaction declared in the network file. The `Parameter('fwd_ReactionName', 0))` must be set to non-zero to activate the rule and `Parameter('rvs_ReactionName', 0))` must be set to non-zero to define a reversible reaction.

Note: **Simulation**. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes).

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible `Observables` for metabolites.

2.6 Genome Graphs

Metabolic networks have four columns. The first declares a unique name for the enzyme or enzymatic complex; the second declares a unique name for the reaction; the third column lists using comma unique names for substrates; and the last row list using comma unique names for products. To declare metabolites located at the periplasm or extracellular compartments, the user should employ the prefix "PER-" and "EX-", respectively. Use *spontaneous* for non-enzymatic reactions.

Examples:

```
SOURCE         TARGET
araB-pro1         araB-rbs
araB-rbs        araB-cds
araB-cds        araA-rbs
araA-rbs        araA-cds
araA-cds        araD-rbs
araD-rbs        araD-cds
araD-cds        araD-ter1

araC-pro1         araC-BS-56-72
araC-BS-56-72         araC-rbs
araC-rbs        araC-cds
araC-cds        araC-ter1

araE-pro1         araE-rbs
araE-rbs        araE-cds
araE-cds        araE-ter1

araF-pro1         araF-rbs
araF-rbs        araF-cds
araF-cds        araG-rbs
araG-rbs        araG-cds
araG-cds        araH-rbs
araH-rbs        araH-cds
araH-cds        araH-ter1
```

OR

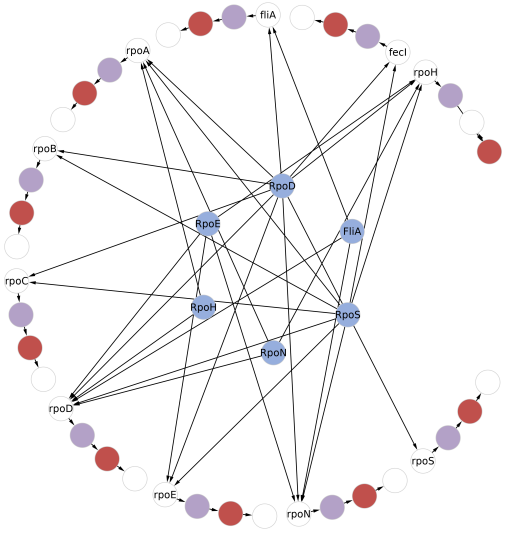
```
SOURCE         TARGET
rpoA-pro1         rpoA-rbs
rpoA-rbs        rpoA-cds
rpoA-cds        rpoA-ter1

rpoB-pro1         rpoB-rbs
rpoB-rbs        rpoB-cds
rpoB-cds        rpoC-rbs
```

(continues on next page)

(continued from previous page)

```
rpoC-rbs        rpoC-cds
rpoC-cds        rpoC-ter1

rpoD-pro1        rpoD-rbs
rpoD-rbs        rpoD-cds
rpoD-cds        rpoD-ter1

rpoE-pro1        rpoE-rbs
rpoE-rbs        rpoE-cds
rpoE-cds        rpoE-ter1

rpoH-pro1        rpoH-rbs
rpoH-rbs        rpoH-cds
rpoH-cds        rpoH-ter1

rpoN-pro1        rpoN-rbs
rpoN-rbs        rpoN-cds
rpoN-cds        rpoN-ter1

rpoS-pro1        rpoS-rbs
rpoS-rbs        rpoS-cds
rpoS-cds        rpoS-ter1

fliA-pro1        fliA-rbs
fliA-rbs        fliA-cds
fliA-cds        fliA-ter1

fecI-pro1        fecI-rbs
fecI-rbs        fecI-cds
fecI-cds        fecI-ter1
```

Note: Visualization in Cytoscape. Colors and arrows remains to the user for customization. The network could be complemented with a description of sigma factor specifity for promoter, as the following network



Finally, execute the "*Rules from metabolic network.ipynb*" to obtain the *Rules* to model the defined network. If using a Sigma Factor-Promoter Interaction Network, the user could use "*Rules from SigmaFactors x Architecture*" to obtain

the *Rules* to model both network at once. The complete rule-based model can be found in the arabinose folder (1st example) and in the sigma folder (2nd example) from the Network Biology Lab GitHub repository here.

Note: Kappa BioBrick Framework. The *Rules* for transcription and translation come from the work of Stewart and Wilson-Kanamori (See more here). A "pure" genome graph uses the originally defined rules, while a genome graph + sigma factor specifity uses a modified *rules* to model the release of the sigma factor from the RNA Polymerase at the transcription initiation. Please note the explicit modeling of the RNA Polymerase complex in the second example.

```
Rule('docking_araB_pro1',
        cplx(name = 'RNAP', dna = None) + dna(name = 'araB', type = 'pro1', prot = None, free = 'True') |
        cplx(name = 'RNAP', dna = 1) % dna(name = 'araB', type = 'pro1', prot = 1, free = 'False'),
        Parameter('fwd_docking_araB_pro1', 1), Parameter('rvs_docking_araB_pro1', 1))
```

OR

```
# [rpoA, rpoA, rpoB, rpoC, rpoD] interacts with BS_rpoA_pro1
Rule('docking_1_rpoA_pro1',
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = None, met = None, up = 4, dw = None) +
        dna(name = 'rpoA', type = 'pro1', prot = None, free = 'True', up = WILD, dw = WILD) |
        prot(name = 'rpoA', dna = None, met = None, up = None, dw = 1) %
        prot(name = 'rpoA', dna = None, met = None, up = 1, dw = 2) %
        prot(name = 'rpoB', dna = None, met = None, up = 2, dw = 3) %
        prot(name = 'rpoC', dna = None, met = None, up = 3, dw = 4) %
        prot(name = 'rpoD', dna = 5, met = None, up = 4, dw = None) %
        dna(name = 'rpoA', type = 'pro1', prot = 5, free = 'False', up = WILD, dw = WILD),
        Parameter('fwd_docking_1_rpoA_rbs', 1),
        Parameter('rvs_docking_1_rpoA_pro1', 0))
```

Note: Reversibility of reactions. Atlas writes irreversible *Rules* for each interaction between the RNA Polymerase and a promoter. The `Parameter('rvs_ReactionName', 0))` must be set to non-zero to define a reversible reaction. The remaining *Rules* are irreversible without a way to define reversible reactions.

Note: Simulation. The model can be simulated only with the instantiation of `Monomers` and `Initials` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the necessary `Monomers` and `Initials` (including proteins and enzymatic complexes). For initial genes, please refer to the following example:

Plotting. The model can be observed only with the instantation of `Observables` (More here). Run *Monomer+Initials+Observables from metabolic network.ipynb* to obtain automatically the all possible `Observables` for metabolites.

CHAPTER

THREE

SIMULATION

Simulation could be done within the PySB python package (See more at PySB documentation) . Here is the relevant code that able the simulation of any PySB model, albeit PySB exports the model, calls the simulator, and imports the results under the hood. See *Plotting* for a simple example on how to plot simulation results.

```
from numpy import linspace
from pysb.bng import generate_network, generate_equations
from pysb.simulator import ScipyOdeSimulator, BngSimulator, KappaSimulator

# modify accordingly
from pysb.pathfinder import set_path
set_path('bng', '/opt/git-repositories/bionetgen.RuleWorld/bng2/')
set_path('kasim', '/opt/git-repositories/KaSim4.Kappa-Dev/')

## for network-based simulations:
## ScipyOdeSimulator and BngSimulator ode and ssa methods
# generate_network(model)
# generate_equations(model)

## set the number of stochastic simulations
runs = 100
# data1 = ScipyOdeSimulator(model, linspace(0, 100, 200)).run().dataframe
# data1 = BngSimulator(model, linspace(0, 200, 201)).run(method = 'ode').dataframe
# data2 = BngSimulator(model, linspace(0, 200, 201)).run(method = 'ssa', n_runs =
↪runs).dataframe
# data2 = BngSimulator(model, linspace(0, 200, 201)).run(method = 'nf', n_runs =
↪runs).dataframe
data2 = KappaSimulator(model, linspace(0, 100, 101)).run(n_runs = runs).dataframe

## process simulations
data = []
for i in range(0, runs):
        data.append(data2.xs(i))

avrg = 0
for i in range(0, runs):
        avrg += data[i]
avrg = avrg / runs

stdv = 0
for i in range(0, runs):
        stdv += (data[i] - avrg)**2
stdv = (stdv / (runs-1))**0.5
```

CHAPTER

FOUR

PLOTTING

PySB could inform the results of a simulation to dataframes (See *Simulation*) and visualization of results could be done with matplotlib or seaborn even (See more here). To access the data, the dataframes columns reproduce the names of the `Observables`. The following example could be adapted to show the dynamics of any `Observable`.

Note: Importantly, PySB allows the inspection of the model to find which `Monomers` (and complexes of monomers) exists in the model, but as the simulation is network-free, the possible formed complexes are up to the user concern.

Note: Atlas produces automatically `Observables` for metabolites, and other components and complexes could also be observed and plotted, but their declaration in the model is entirely up to the user.

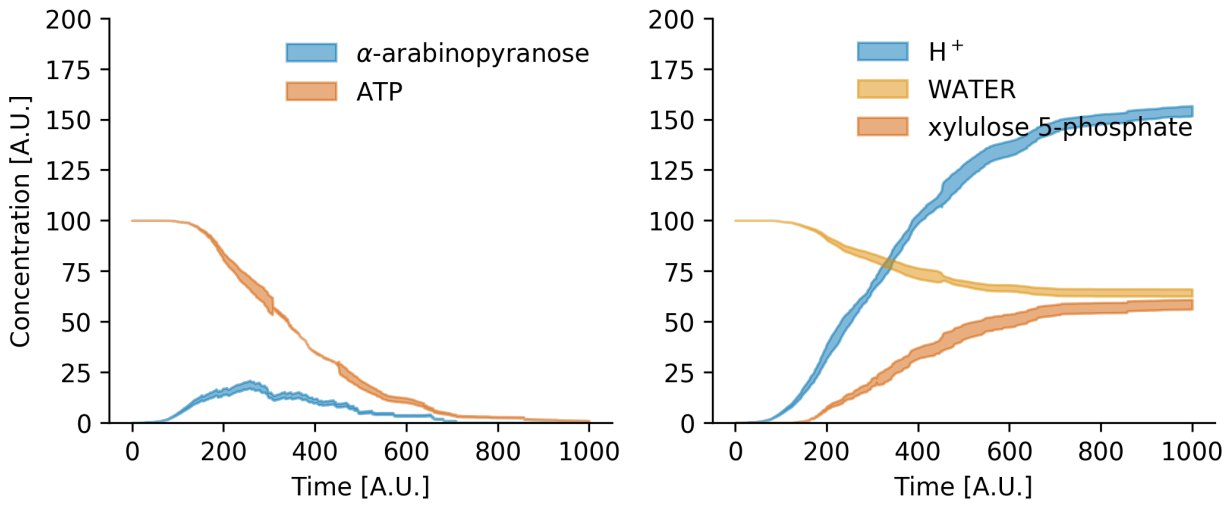
```
fig, ax = plt.subplots(1, 2, figsize = (4*2, 3*1), dpi = 100)

# ax[0].plot(data1.index, data1['obs_alpha_L_arabinopyranose_cyt'], label = '__NOLABEL__', color = palette[0])
# ax[0].plot(data1.index, data1['obs_ATP_cyt'], label = '__NOLABEL__', color = palette[3])
ax[0].fill_between(avrg.index,
        avrg['obs_alpha_L_arabinopyranose_cyt'] + stdv['obs_alpha_L_arabinopyranose_cyt'],
        avrg['obs_alpha_L_arabinopyranose_cyt'] - stdv['obs_alpha_L_arabinopyranose_cyt'],
        label = r'$\alpha$-arabinopyranose', **{'color' : palette[0], 'alpha' : 0.5})
ax[0].fill_between(avrg.index,
        avrg['obs_ATP_cyt'] + stdv['obs_ATP_cyt'],
        avrg['obs_ATP_cyt'] - stdv['obs_ATP_cyt'],
        label = r'ATP', **{'color' : palette[3], 'alpha' : 0.5})

# ax[1].plot(data1.index, data1['obs_PROTON_cyt'], label = '__NOLABEL__', color = palette[0])
# ax[1].plot(data1.index, data1['obs_XYLULOSE_5_PHOSPHATE_cyt'], label = '__NOLABEL__', color = palette[3])
ax[1].fill_between(avrg.index,
        avrg['obs_PROTON_cyt'] + stdv['obs_PROTON_cyt'],
        avrg['obs_PROTON_cyt'] - stdv['obs_PROTON_cyt'],
        label = r'H$^+$', **{'color' : palette[0], 'alpha' : 0.5})
ax[1].fill_between(avrg.index,
        avrg['obs_WATER_cyt'] + stdv['obs_WATER_cyt'],
        avrg['obs_WATER_cyt'] - stdv['obs_WATER_cyt'],
        label = 'WATER', **{'color' : palette[1], 'alpha' : 0.5})
ax[1].fill_between(avrg.index,
```

(continues on next page)

(continued from previous page)

```
        avrg['obs_XYLULOSE_5_PHOSPHATE_cyt'] + stdv['obs_XYLULOSE_5_PHOSPHATE_cyt'],
        avrg['obs_XYLULOSE_5_PHOSPHATE_cyt'] - stdv['obs_XYLULOSE_5_PHOSPHATE_cyt'],
        label = r'xylulose 5-phosphate', **{'color' : palette[3], 'alpha' : 0.5})

ax[0].set_xlabel('Time [A.U.]')
ax[0].set_ylabel('Concentration [A.U.]')
# ax[0].set_xlim(left = 0, right = 100)
ax[0].set_ylim(bottom = 0, top = 200)

ax[1].set_xlabel('Time [A.U.]')
# ax[1].set_xlim(left = 0, right = 100)
ax[1].set_ylim(bottom = 0, top = 200)

ax[0].legend(frameon = False)
ax[1].legend(frameon = False)

seaborn.despine()
plt.savefig('Fig_Arabinose.png', format = 'png', bbox_inches = 'tight', dpi = 350)
# for publication
# plt.savefig('Fig_Arabinose.pdf', format = 'pdf', bbox_inches = 'tight', dpi = 350)

plt.show()
```

And the results is



Note: See the Arabinose Model to inspect the rules and reproduce (at some extent because of stochasticity) the plot showed in this Manual.

CHAPTER

FIVE

EXPORT TO

The PySB python package could export to different languages (See more here). Use the following code to export to BioNetGen and *kappa* languages, putting the code at the end of the model.

```
from pysb.export import export
with open('model.kappa', 'w') as outfile:
        outfile.write(export(model, 'kappa'))
with open('model.bngl', 'w') as outfile:
        outfile.write(export(model, 'bngl'))
```

Note: In the case of matlab, mathematica, and stochkit, PySB requires to expand the rules to determine all mass-balances to write ODE-based models, a proccess call network generation and could take excessive time to finish.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search