Short Communication
Open Access

AdvanceSyn Toolkit: An open-source suite for model development and analysis in biological engineering

Abstract

Modelling and simulations are useful means to screen potential experimental designs for metabolic engineering. Genome-scale models of metabolism (GSM) and kinetic models (KMs) are the two main approaches for modelling, which resulted in largely disjoint computational tools for GSMs and KMs. Existing tools for GSMs require knowledge of the underlying programming languages while the development and merger of two or more KMs is difficult. In this work, AdvanceSyn Toolkit is an open-sourced high-level command-line tool to develop KMs, and to analyse GSMs and KMs; licensed under the Apache License, Version 2.0, for academic and not-for-profit use. It elevates the need to know the underlying programming language for GSM analysis. AdvanceSyn Model (ASM) specification is a simple and modular format for model development and AdvanceSyn Toolkit provides a method to merge two or more model files for simulation and sensitivity analysis.

Volume 9 Issue 4 - 2020

Maurice HT Ling
AdvanceSyn Private Limited, Republic of Singapore, Singapore

Correspondence: Maurice HT Ling, AdvanceSyn Private Limited, Republic of Singapore, Singapore, Email mauriceling@advancesyn.com

Received: September 22, 2020 | **Published:** October 08, 2020

Introduction

Metabolic engineering and synthetic biology are important complementary platforms in the current Fourth Industrial Revolution to translate research into commercially viable products,[1,2] with many researchers calling for the utilization of both fields[3–5] to access nature's diversity.[6] As engineered circuits are becoming increasingly complex, our limited knowledge in optimizing complex circuits often impedes current efforts and computational models are seen as a means of screening potential designs.[7] Several recent studies had used computational models to aid in designing engineering strategies. For example, Kim et al.[8] used genome-scale metabolic model (GSM) to improve lipid production in *Yarrowia lipolytica* and Wayman et al.[9] used GSM to improve glycan production in *Escherichia coli* while Tian et al.[10] used kinetic model (KM) to improve N-acetylneuraminic acid production in *Bacillus subtilis*.

Modelling refers to the mathematical construction of a model while simulation is the execution and solving of the model.[11] GSMs and KMs are the two main modelling approaches for metabolic engineering. GSMs are concerned with the distribution of intracellular fluxes of metabolites while KMs are concerned with the interactions of metabolites. GSMs, also known as constraint-based models, are based largely on metabolic stoichiometries and mass balance[12] while KMs are largely based on rate equation.[13] As such, GSMs cannot capture the relationship between flux, enzyme expression, metabolite levels, and regulation that is possible with KMs.[14] Hence, GSMs generally provides only steady-state distribution of metabolic fluxes while KMs provide time series of metabolite concentrations. Kerkhoven et al.[15] surmised GSMs as top-down approach while KMs as bottom-up approach. Sier et al.[16] has demonstrated that coupling both approaches can yield novel insights.

Due to the fundamental differences in modelling philosophies, computational tools for GSMs and KMs are largely disjoint. The core tool for GSM is COBRA Toolbox[17] for MATLAB and its subsequent version, COBRApy[18] for Python programming language. A large repertoire of tools had been developed for GSMs over the last 2 decades[19] since the publication of the first GSM in 1999.[20] Cameo[21] builds on top of these tools and presents a high-level interface for GSM usage. Yet, Python programming knowledge is required to use Cameo as it is a Python library. On the KM front, the most well-known tool is COPASI[22] which had been used in many studies.[23] However, despite providing a user-friendly interface to simulate models and present results, it is difficult to merge multiple existing models in COPASI as it requires finding the common metabolites between the two models and rewriting the affected equations.

To address these difficulties, AdvanceSyn Toolkit is presented as a high-level command-line tool to develop KMs, and to analyse GSMs and KMs. AdvanceSyn Toolkit wraps key operations in Cameo[21] into a unified command-line interface; thus, elevating the need to know Python programming. As a command-line interface tool, AdvanceSyn Toolkit can be incorporated into computational biology and bioinformatics pipelines.[24] AdvanceSyn Model (ASM) specification is based on Antimony language[25] used in Tellurium,[26] which is simple and modular; and initialization file structure. This makes ASM a code file format rather than a data-exchange file format; such as JSON, which is substantially more verbose, requires more structure, and data type formatting. Moreover, AdvanceSyn Toolkit provides a method to merge two or more ASM files, a feature not found in existing tools, which allows for more effective reuse of existing models.

Results

AdvanceSyn Toolkit is open source software written in Python and Python-Fire module (https://github.com/google/python-fire), which aims to simplify the implementation of command-line interface in Python 3. This combination has been used in several other tools.[27,28] AdvanceSyn Toolkit has two main sets of operations (Figure 1; Supplementary materials S2 to S21); namely, 10 operations for KMs, and 9 operations for GSMs via Cameo.[21] Here, three use cases are presented to illustrate core features of AdvanceSyn Toolkit.

Use Case #1; Development and Analysis of KM: Here, two separate KMs were developed, one for glycolysis pathway (Supplementary material S22) and another for pentose phosphate pathway (Supplementary material S23). Glycolysis pathway consists of nine reaction steps while pentose phosphate pathway consists of six reaction steps (Figure 2). However, there is one reaction linking glycolysis, from glucose-6-phosphate (g6p), to 6-phosphogluconolactone (pgl6) in pentose phosphate pathway. For simplicity, co-factor(s) and co-substrate(s) such as ATP and NADP are not modelled. Each reaction step is mathematically modelled as a rate expression, also known as

rate law,[29] which corresponds to the kinetic law in System Biology Markup Language (SBML).[30] This allows the concentration of each metabolite is modelled as a rate equation in the form of

$$\frac{d[metabolite]}{dt} = \sum_{i=1}^{N} production_i - \sum_{i=1}^{N} usage_i$$

where each production and usage term represents a reaction step and in this use case, is modelled as a product of the concentration of enzyme and substrate(s) in the general form of $[enzyme] \times \sum_{i=1}^{N} [substrate_i]$.

For example, the concentration of glucose-6-phosphate (g6p) over time is modelled as $\frac{d[g6p]}{dt} = [HK][glucose] - ([PGI][g6p] + [G6PD][g6p])$ where HK, PGI, and G6PD are hexokinase (converting glucose to glucose-6-phosphate), phosphoglucoisomerase (converting glucose-6-phosphate to fructose-6-phosphate), and glucose 6-phosphate dehydrogenase (converting glucose-6-phosphate to 6-phosphogluconolactone), respectively. AdvanceSyn Toolkit does not assume any specific type of rate expression as each reaction is defined by the user in the model, making it possible to have a different type of rate expression for each reaction.

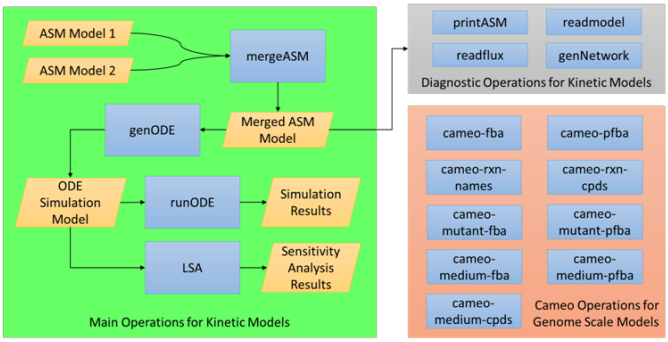

Figure 1 Overview of functionality.

The concentration of all metabolites and enzymes are fixed at 1 micromolar; except glucose, which is given at 1 millimolar. Glycolysis pathway model and pentose phosphate pathway model were merged (Supplementary material S24) based on common metabolites across the two models before generating a simulation script (Supplementary material S25) for simulation (Supplementary material S26). Merging is performed under the assumption that units used in both models are identical. Briefly, the models are merged using the union set of the metabolites as nodes and re-coding each reaction as edges and parameters as attributes. Our simulation results suggest that the rate of 6-phosphogluconolactone (pgl6) increases faster than both pyruvate (pyr) and xylulose-5-phosphate (x5p) given the current conditions. Sensitivity analysis (Supplementary material S27) using One-Factor-at-a-Time method[31,32] was used to evaluate the effects of varying enzyme concentrations (as listed in the Variables section of the model) on the overall metabolite distribution. Mean square error (MSE) between the metabolites in each variation of enzyme concentration and that of the original model, which is directly proportional to the impact of the enzyme concentration on the overall metabolite distribution, were calculated as

$$MSE_{Enzyme} = \sum_{i=1}^{N} \left(metabolite_{[Enzyme_{changed}],i} - metabolite_{[Enzyme_{original}],i} \right)^2 / N \ .$$

Our sensitivity analysis suggests that hexokinase concentration has the most impact followed by phosphoglucoisomerase and glucose 6-phosphate dehydrogenase. The results of both simulation and sensitivity analysis are given as comma-delimited files to enable analysis by other statistical software. This also allows the user to determine the effect of the concentration of each enzyme on a metabolite of interest, which can be useful in informing engineering strategies (Figure 2).[33]

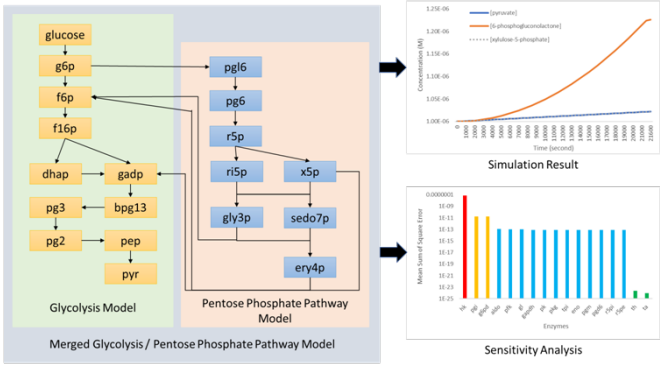

Figure 2 Merging and analysis of KMs. The acronyms for metabolites in the models are as follow (in alphabetical order): bpg13 (D-1,3-bisphosphoglycerate), dhap (Dihydroxyacetone-phosphate), ery4p (erythose-4-phosphate), g6p (D-Glucose-6-phosphate), gadp (D-glyceraldehyde-3-phosphate), glucose (D-glucose), gly3p (glyceraldehyde-4-phosphate), f16p (D-Fructose-1,6-bisphosphate), f6p (D-Fructose-6-phosphate), pep (phosphoenolpyruvate), pg2 (2-phosphoglycerate), pg3 (3-phosphoglycerate), pg6 (6-phosphogluconate), pgl6 (6-phosphogluconolactone), pyr (pyruvate), r5p (ribulose-5-phosphate), ri5p (ribose-5-phosphate), sedo7p (sedoheptulose-7-phosphate), and x5p (xylulose-5-phosphate) (Figure 3).

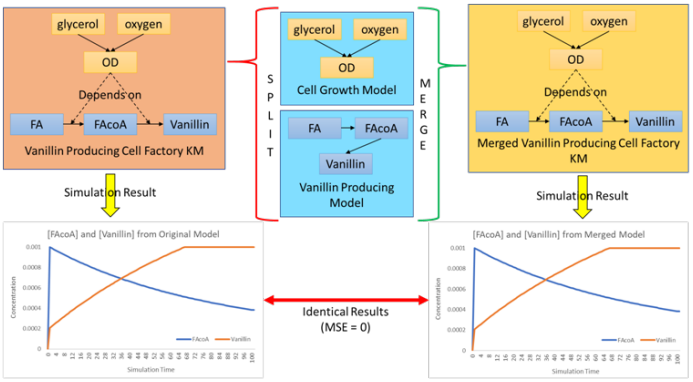

Figure 3 Splitting and merging of model gives identical simulation results.

Use Case #2; Merging of KMs: To further emphasize merging of KMs, the simplified vanillin producing cell factory KM by Yeoh et al.[34] was implemented in totality (Supplementary material S28) and simulated. The original model was split into two constituent models of cell growth model (Supplementary material S29) and vanillin producing model (Supplementary material S30). These two constituent models were then merged (Supplementary material S31) and simulated. Simulation results from the original and merged models were compared. Our result (Figure 3) shows that simulation results from the original model is identical (MSE = 0) to that of the merged model, demonstrating that the model merging algorithm is functioning correctly. In addition, this also demonstrate an important usage of AdvanceSyn Toolkit where cloned gene cassette(s), vanillin producing pathway in this case, can be merged with cell host model. Moreover, Yeoh et al.[34] use Hill equation,[35] which is different to that in Use Case #1; this further emphasizes that AdvanceSyn Toolkit does not assume any specific type of rate expression.

Use Case #3; Analysis of GSM: GSMs are commonly used for evaluating knockout strategies[36] or media components[37] to optimize yield of a metabolite of interest. AdvanceSyn Toolkit is built on top of

the functions of Cameo[21] to evaluate the biomass objective function,[38] which is a proxy for growth rate, or fluxes when one or more media component(s) or enzyme expression(s) were changed. Chang and Ling[39] had used AdvanceSyn Toolkit to explain the effects of varying glucose concentration in media, using cameo-medium-fba or cameo-medium-pfba operations, to the fluxes of *Escherichia coli* MG1655 using the GSM model, iAF1260.[40] It is essentially an attempt to explain Monod Equation in terms of metabolism and found a strong correlation (r = 0.972) between Monod's predicted growth rate and biomass objective value from GSM. Moreover, Chang and Ling[39] suggests that Monod's predicted growth rate of *E. coli* MG1655 can be predicted by the fluxes of 14 enzymes. This illustrates that AdvanceSyn Toolkit can use operations from Cameo[21] for GSM analysis without the need to learn Python programming.

Conclusion

AdvanceSyn Toolkit is an open-source, high-level command-line tool to develop KMs, and to analyse GSMs and KMs. The ability to merge two or more KMs into a unified KM is a unique feature of AdvanceSyn Toolkit as this feature allows for incremental development of more advanced models. AdvanceSyn Toolkit is a project under active development. However, it is not possible to merge GSMs with KMs or to merge multiple GSMs at this moment. Other future work includes interfacing with existing tools to accept a diverse range of model specifications in order to be an effective model translator, and to expand the repertoire of existing tools by chaining operations across various existing tools.

Supplementary materials and data

The supplementary materials and data set for this article are available at http://bit.ly/ADSToolkit_Supplement and http://bit.ly/ADSToolkit_DataSet respectively.

Availability and license

AdvanceSyn Toolkit is licensed under the Apache License, Version 2.0 for academic and not-for-profit use only, and is available at https://bit.ly/ADSToolkit.

Acknowledgments

None.

Conflicts of interest

The author is a co-founder of AdvanceSyn Private Limited and AdvanceSyn Toolkit is employed in consultancy services offered by the company.

References

1. Ramzi AB. Metabolic Engineering and Synthetic Biology. *Adv Exp Med Biol*. 2018;1102:81–95.
2. Palazzotto E, Tong Y, Lee SY, Weber T. Synthetic Biology and Metabolic Engineering of Actinomycetes for Natural Product Discovery. *Biotechnol Adv*. 2019;37(6):107366.
3. Choi KR, Jang WD, Yang D, et al. Systems Metabolic Engineering Strategies: Integrating Systems and Synthetic Biology with Metabolic Engineering. *Trends Biotechnol.* 2019;37(8):817–837.
4. Nguyen GT, Kim Y-G, Ahn J-W, et al. Structural Basis for Broad Substrate Selectivity of Alcohol Dehydrogenase YjgB from Escherichia coli. *Molecules*. 2020:25(10):E2404.
5. Shen Y-P, Niu F-X, Yan Z-B, et al. Recent Advances in Metabolically Engineered Microorganisms for the Production of Aromatic Chemicals Derived From Aromatic Amino Acids. *Front Bioeng Biotechnol*. 2020;8:407.
6. King JR, Edgar S, Qiao K, et al.. Accessing Nature's Diversity through Metabolic Engineering and Synthetic Biology. *F1000Research*. 2016;5:F1000 Faculty Rev-397.
7. Naseri G, Koffas MAG. Application of Combinatorial Optimization Strategies in Synthetic Biology. *Nat Commun*. 2020;11(1):2446.
8. Kim M, Park BG, Kim E-J, et al. In Silico Identification of Metabolic Engineering Strategies for Improved Lipid Production in Yarrowia lipolytica by Genome-Scale Metabolic Modeling. *Biotechnol Biofuels*. 2019;12:187.
9. Wayman JA, Glasscock C, Mansell TJ, et al. Improving Designer Glycan Production in Escherichia coli through Model-Guided Metabolic Engineering. *Metab Eng Commun*. 2019;9:e00088.
10. Tian R, Liu Y, Chen J, et al. Synthetic N-terminal Coding Sequences for Fine-Tuning Gene Expression and Metabolic Engineering in Bacillus subtilis. *Metab Eng*. 2019;55:131–141.
11. Ling M. Of (Biological) Models and Simulations. *MOJ Proteomics Bioinforma*. 2016;3:00093.
12. Gu C, Kim GB, Kim WJ, Kim HU, Lee SY. Current Status and Applications of Genome-Scale Metabolic Models. *Genome Biol*. 2019;20(1):121.
13. Resat H, Petzold L, Pettigrew MF. Kinetic Modeling of Biological Systems. *Methods Mol Biol*. 2009;541:311–335.
14. Strutz J, Martin J, Greene J, et al. Metabolic Kinetic Modeling Provides Insight into Complex Biological Questions, but Hurdles Remain. *Curr Opin Biotechnol.* 2019;59:24–30.
15. Kerkhoven EJ, Lahtvee P-J, Nielsen J. Applications of Computational Modeling in Metabolic Engineering of Yeast. *FEMS Yeast Res*. 2015;15(1):1–13.
16. Sier JH, Thumser AE, Plant NJ. Linking Physiologically-Based Pharmacokinetic and Genome-Scale Metabolic Networks to Understand Estradiol Biology. *BMC Syst Biol*. 2017;11(1):141.
17. Schellenberger J, Que R, Fleming RMT, et al. Quantitative Prediction of Cellular Metabolism with Constraint-Based Models: The COBRA Toolbox v2.0. *Nat Protoc*. 2011;6(9):1290–307.
18. Ebrahim A, Lerman JA, Palsson BO, et al. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Syst Biol*. 2013;8;7:74.
19. Machado D, Herrgård MJ. Co-Evolution of Strain Design Methods Based on Flux Balance and Elementary Mode Analysis. *Metab Eng Commun*. 2015;2:85–92.
20. Edwards JS, Palsson BO. Systems Properties of the Haemophilus influenzae Rd Metabolic Genotype. *J Biol Chem*. 1999;274(25):17410–17416.
21. Cardoso JGR, Jensen K, Lieven C, et al. Cameo: A Python Library for Computer Aided Metabolic Engineering and Optimization of Cell Factories. *ACS Synth Biol*. 2018;7(4):1163–1166.
22. Hoops S, Sahle S, Gauges R, et al. COPASI — A COmplex PAthway SImulator. *Bioinformatics*. 2006;22(24):3067–3074.
23. Bergmann FT, Hoops S, Klahn B, et al. COPASI and Its Applications in Biotechnology. *J Biotechnol*. 2017;261:215–220.
24. Leipzig J. A Review of Bioinformatic Pipeline Frameworks. *Brief Bioinform*. 2017;18(3):530–536.

25. Smith LP, Bergmann FT, Chandran D. Antimony: A Modular Model Definition Language. *Bioinformatics*. 2009;25(18):2452–2454.

26. Choi K, Medley JK, König M, et al. Tellurium: An Extensible Python-Based Modeling Environment for Systems and Synthetic Biology. *Biosystems*. 2018;171:74–79.

27. Ling MH. Island: A Simple Forward Simulation Tool for Population Genetics. *Acta Sci Comput Sci*. 2019;1(2):20–22.

28. Ling MH. RANDOMSEQ: Python Command–Line Random Sequence Generator. *MOJ Proteomics Bioinforma*. 2018;7(4):206–208.

29. Liebermeister W, Uhlendorf J, Klipp E. Modular Rate Laws for Enzymatic Reactions: Thermodynamics, Elasticities and Implementation. *Bioinformatics*. 2010;26(12):1528–1534.

30. Hucka M, Bergmann FT, Hoops S, et al. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. *J Integr Bioinforma*. 2015;12(2):266.

31. Abou-Taleb KA, Galal GF. A Comparative Study Between One-Factor-At-A-Time and Minimum Runs Resolution-IV Methods for Enhancing the Production of Polysaccharide by Stenotrophomonas daejeonensis and Pseudomonas geniculate. *Ann Agric Sci*. 2018;63(2):173–180.

32. Razavi S, Gupta HV. What Do We Mean by Sensitivity Analysis? The Need for Comprehensive Characterization of "Global" Sensitivity in Earth and Environmental Systems Models: A Critical Look at Sensitivity Analysis. *Water Resour Res*. 2015;51(5):3070–3092.

33. Tamano K. Enhancing Microbial Metabolite and Enzyme Production: Current Strategies and Challenges. *Front Microbiol*. 2014;5:718.

34. Yeoh JW, Jayaraman S, Tan SG-D, et al. A Model-Driven Approach towards Rational Microbial Bioprocess Optimization. *Biotechnol Bioeng*. 2020;10.1002/bit.27571.

35. Hill AV. The Combinations of Haemoglobin with Oxygen and with Carbon Monoxide. *I Biochem J*. 1913;7(5):471–480.

36. Nair G, Jungreuthmayer C, Zanghellini J. Optimal Knockout Strategies in Genome-Scale Metabolic Networks Using Particle Swarm Optimization. *BMC Bioinformatics*. 2017;18(1):78.

37. Li C-T, Yelsky J, Chen Y, et al. Utilizing Genome-Scale Models to Optimize Nutrient Supply for Sustained Algal Growth and Lipid Productivity. *Npj Syst Biol Appl*. 2019;5(1):33.

38. Feist AM, Palsson BO. The Biomass Objective Function. *Curr Opin Microbiol*. 2010;13(3):344–349.

39. Chang ED, Ling MH. Explaining Monod in terms of Escherichia coli metabolism. *Acta Sci Microbiol*. 2019;2(9):66–71.

40. Feist AM, Henry CS, Reed JL, et al. A Genome-Scale Metabolic Reconstruction for Escherichia coli K-12 MG1655 that Accounts for 1260 ORFs and Thermodynamic Information. *Mol Syst Biol*. 2007;3:121.

Citation: Ling MHT. AdvanceSyn Toolkit: An open-source suite for model development and analysis in biological engineering. *MOJ Proteomics Bioinform.* 2020;9(4):83–86. DOI: 10.15406/mojpb.2020.09.00286

AdvanceSyn Toolkit Supplementary Materials

Table of Contents

S1. Writing an AdvanceSyn Model Specification (Type 1)

AdvanceSyn Model (ASM) Specification is based on INI file, which is commonly used for software configuration.

In the simplest sense, a ASM file is a set of key-value pairs in the format of "key : value". For example, ip_address : 100.0.0.1 means that the IP address (ip_address) is set to 100.0.0.1. The main thing to note is that key must be a single string (no spaces in between). For organization purposes, key-value pairs can be grouped into sections. In this case, each key within a section must be unique.

Six sections are defined in ASM (Type 1):

1. Specification: Defining the ASM version.
2. Identifiers: Providing metadata about the model.
3. Objects: Defining the objects / entities in the model.
4. Initials: Defining the initial values of each object / entity.
5. Variables: Defining changeable variables used in the model.
6. Reactions: Defining the reactions / flows between each object / entity in the model.

Example Reaction

As an example, we will define a chemical reaction `A + B = P + Q` where the rate term can be defined as bAB. This means that the rate where substrates A and B forms products P and Q is bAB/second - b(concentration of A)(concentration of B).

Written in ordinary differential equations (ODEs),

```
dA/dt = -bAB
dB/dt = -bAB
dP/dt = bAB
dQ/dt = bAB
```

In another words, coefficient b is a rate variable.

Specification Section

Specification section is used to tell AdvanceSyn Toolkit what specification type or version this ASM file is written under. Yes, it is very likely that we will have different specification versions in future. For example,

```
[Specification]
type: 1
```

defines this file as ASM specification type 1.

Identifiers Section

Identifiers section is for the author(s) to put in descriptions and other details to describe this model. This section is mainly for description and will not be used by AdvanceSyn Toolkit for processing; thus, giving the author(s) rather free rein to describe the model. For example,

```
[Identifiers]
name: 2 substrates to 2 products
author: Maurice Ling
```

Objects Section
Objects section defines all the objects (in this case, molecules) used in this model. The key is the molecule name while the value is the corresponding molecule descriptor. For example,

```
[Objects]
A: molecule A
B: molecule B
P: molecule P
Q: molecule Q
```

Initials Section
Initials section defines the initial conditions (in this case, initial concentrations of the molecules). For example,

```
[Initials]
A: 1e-3
B: 1e-3
P: 0.0
Q: 0.0
```

defines that molecules A and B are 1 mM each while there is no molecules P and Q at the start of the reaction.

Variables Section
Variables section defines the changeable variables used in the model. These variables will be used in define the reactions. In this case, there is only 1 variable, b, as

```
[Variables]
b: 100
```

Reactions Section
Reactions section defines the flow or movement within the model. The keys are just unique identifiers in this model. It is the values that are important. For reactions, the values have 2 parts, flow and rate term (or rate law), separated by |. For example,

```
[Reactions]
r1: A + B -> P + Q | ${Variables:b} * A * B
```

defines reaction r1 as `A + B -> P + Q` (A and B to produce P and Q). The rate term is `${Variables:b} * A * B` where `${Variables:b}` takes the value of `b` in Variable section, resulting in the final rate term as `100 * A * B` in this case.

Full Resulting Model

```
[Specification]
type: 1

[Identifiers]
name: 2 substrates to 2 products
```

```
author: Maurice Ling

[Objects]
A: molecule A
B: molecule B
P: molecule P
Q: molecule Q

[Initials]
A: 1e-3
B: 1e-3
P: 0.0
Q: 0.0

[Variables]
b: 100

[Reactions]
r1: A + B -> P + Q | ${Variables:b} * A * B
```

S2. Operation: genMO

Synopsis: Read the AdvanceSyn model specification file(s) and generate a file consisting of the internal model objects.

Usage: `python astools.py genMO [option]`

where `[option]` can be

- modelfile: Relative path to the model specification file.
- outputfile: Relative path to the output model objects file.
- prefix: Prefix for new reaction IDs. This prefix cannot be any existing prefixes in any of the model specifications to be merged. Default = 'exp'.

For example:

```
python astools.py readflux \
    --mtype=ASM \
    --modelfile=models/asm/glycolysis.modelspec
```

S3. Operation: genNetwork

Synopsis: Read the AdvanceSyn model specification file(s) and generate a network / reaction visualization file.

Usage: `python astools.py genNetwork [option]`

where `[option]` can be

- modelfile: Relative path(s) to the model specification file(s), separated by semi-colon.
- outputfile: Relative path to the output file.

- outfmt: Type of network visualizatio format to generate. Allowable options are 'SIF' (Simple Interaction Format). Default = 'SIF' (Simple Interaction Format).

For example:

```
python astools.py genNetwork \
    --outfmt=SIF \
    --
modelfile=models/asm/glycolysis.modelspec;models/asm/ppp.model
spec \
    --outputfile=glycolysis_ppp.sif
```

S4. Operation: genODE

Synopsis: Generate Python ODE script from a given model specification file.

Usage: `python astools.py genODE [option]`

where `[option]` can be

- modelfile: Name of model specification file in models folder. This assumes that the model file is not in models folder.
- type: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.
- solver: Type of solver to use. Allowable types are 'Euler', 'Heun' (Runge-Kutta 2nd method or Trapezoidal), 'RK3' (third order Runge-Kutta), 'RK4' (fourth order Runge-Kutta), 'RK38' (fourth order Runge-Kutta method, 3/8 rule), 'CK4' (fourth order Cash-Karp), 'CK5' (fifth order Cash-Karp), 'RKF4' (fourth order Runge-Kutta-Fehlberg), 'RKF5' (fifth order Runge-Kutta-Fehlberg), 'DP4' (fourth order Dormand-Prince), and 'DP5' (fifth order Dormand-Prince). Default = 'RK4'.
- timestep: Time step interval for simulation. Default = 1.0.
- endtime: Time to end simulation - the simulation will run from 0 to end time. Default = 21600.
- lowerbound: Define lower boundary of objects. For example, "1;2" means that when the value of the object hits 1, it will be bounced back to 2. Default = 0;0; that is, when the value of the object goes to negative, it will be bounced back to zero.
- upperbound: Define upper boundary of objects. For example, "10;9" means that when the value of the object hits 1, it will be pushed down to 9. Default = 1e-3;1e-3; that is, when the value of the object above 1e-3, it will be pushed back to 1e-3.
- odefile: Python ODE script file to write out. This file will be written into odescript folder. Default = odescript.py.

For example:

```
python genODE \
    --modelfile=models/asm/glycolysis.modelspec \
    --mtype=ASM \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --lowerbound=0;0 \
    --upperbound=1e-3;1e-3 \
```

```
--odefile=glycolysis.py
```

S5. Operation: installdep

Synopsis: Install external tools and dependencies. List of external tools and dependencies that will be installed:

- bokeh (https://bokeh.pydata.org), BSD 3-Clause "New" or "Revised" License
- cameo (https://github.com/biosustain/cameo), Apache Licence 2.0

Usage: `python astools.py installdep`

S6. Operation: LSA

Synopsis: Perform local sensitivity analysis using OFAT/OAT (one factor at a time) method where the last data time (end time) simulation results are recorded into results file.

Usage: `python astools.py LSA [option]`

where `[option]` can be

- modelfile: Name of model specification file in models folder. This assumes that the model file is not in models folder.
- multiple: Multiples to change each variable value. Default = 100 (which will multiple the original parameter value by 100).
- prefix: A prefixing string for the set of new model specification for identification purposes. Default = ''.
- type: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.
- solver: Type of solver to use. Allowable types are 'Euler', 'Heun' (Runge-Kutta 2nd method or Trapezoidal), 'RK3' (third order Runge-Kutta), 'RK4' (fourth order Runge-Kutta), 'RK38' (fourth order Runge-Kutta method, 3/8 rule), 'CK4' (fourth order Cash-Karp), 'CK5' (fifth order Cash-Karp), 'RKF4' (fourth order Runge-Kutta-Fehlberg), 'RKF5' (fifth order Runge-Kutta-Fehlberg), 'DP4' (fourth order Dormand-Prince), and 'DP5' (fifth order Dormand-Prince). Default = 'RK4'.
- timestep: Time step interval for simulation. Default = 1.0.
- endtime: Time to end simulation - the simulation will run from 0 to end time. Default = 21600.
- lowerbound: Define lower boundary of objects. For example, "1;2" means that when the value of the object hits 1, it will be bounced back to 2. Default = 0;0; that is, when the value of the object goes to negative, it will be bounced back to zero.
- upperbound: Define upper boundary of objects. For example, "10;9" means that when the value of the object hits 1, it will be pushed down to 9. Default = 1e-3;1e-3; that is, when the value of the object above 1e-3, it will be pushed back to 1e-3.
- cleanup: Flag to determine whether to remove all generated temporary models and ODE code files. Default = True.
- outfmt: Output format. Allowable types are 'reduced' (only the final result will be saved into resultfile) and 'full' (all data, depending on sampling, will be saved into resultfile).

- sampling: Sampling frequency. If 100, means only every 100th simulation result will be written out. The first (start) and last (end) result will always be written out. Default = 100.
- resultfile: Relative or absolute file path to write out sensitivity results. Default = 'sensitivity_analysis.csv'

For example:

```
python genODE \
    --modelfile=models/asm/glycolysis.modelspec \
    --prefix=sen01 \
    --mtype=ASM \
    --multiple=100 \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --cleanup=True \
    --outfmt=reduced \
    --resultfile=sensitivity_analysis.csv
```

S7. Operation: mergeASM

Synopsis: Read the AdvanceSyn model specification file(s) and merge them into a single AdvanceSyn model specification file.

Usage: `python astools.py mergeASM [option]`

where `[option]` can be

- modelfile: Relative path(s) to the model specification file(s), separated by semi-colon.
- outputfile: Relative path to the output ASM model file.
- prefix: Prefix for new reaction IDs. This prefix cannot be any existing prefixes in any of the model specifications to be merged. Default = 'exp'.

For example:

```
python astools.py mergeASM \
    --prefix=exp \
    --
modelfile=models/asm/glycolysis.modelspec;models/asm/ppp.model
spec \
    --outputfile=models/asm/glycolysis_ppp.modelspec
```

S8. Operation: printASM

Synopsis: Read the AdvanceSyn model specification file and print out its details before processing into model objects.

Usage: `python astools.py printASM [option]`

where `[option]` can be

- modelfile: Relative path to the model specification file.

- readertype: Reader type for AdvanceSyn Model specification. Allowable types are 'basic' and 'extended'.

For example:
```
python astools.py printASM \
    --modelfile=models/asm/glycolysis.modelspec \
    --readertype=extended
```

S9. Operation: readmodel

Synopsis: Read a model file and print out its details after processing into model objects.

Usage: `python astools.py readmodel [option]`

where `[option]` can be

- modelfile: Relative path to the model specification file.
- mtype: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification), 'MO' (AdvanceSyn Model Objects). Default = 'ASM'.

For example:
```
python astools.py readmodel \
    --mtype=ASM \
    --modelfile=models/asm/glycolysis.modelspec
```

S10. Operation: readflux

Synopsis: Read the AdvanceSyn model objects file and print out fluxes (productions and usages) of model objects.

Usage: `python astools.py readflux [option]`

where `[option]` can be

- modelfile: Relative path to the model specification file.
- mtype: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification), 'MO' (AdvanceSyn Model Objects). Default = 'ASM'.

For example:
```
python astools.py readflux \
    --mtype=ASM \
    --modelfile=models/asm/glycolysis.modelspec
```

S11. Operation: runODE

Synopsis: Run / execute the ODE model and write out the simulation results.

Usage: `python astools.py runODE [option]`

where `[option]` can be

- odefile: Python ODE script file (in odescript folder) to run / execute.
- sampling: Sampling frequency. If 100, means only every 100th simulation result will be written out. The first (start) and last (end) result will always be written out. Default = 100.
- resultfile: Relative or absolute file path to write out simulation results. Default = 'oderesult.csv'

For example:

```
python astools.py runODE \
    --odefile=glycolysis.py \
    --sampling=500 \
    --resultfile=oderesult.csv
```

S12. Operation: senGen

Synopsis: Generate a series of AdvanceSyn Model Specifications from an existing model by multiplying a multiple to the variable in preparation for sensitivity analyses.

Usage: `python astools.py senGen [option]`

where `[option]` can be

- modelfile: Name of model specification file in models folder. This assumes that the model file is not in models folder.
- multiple: Multiples to change each variable value. Default = 100 (which will multiple the original parameter value by 100).
- prefix: A prefixing string for the set of new model specification for identification purposes. Default = ''.
- mtype: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.

For example:

```
python astools.py senGen \
    --modelfile=models/asm/glycolysis.modelspec \
    --prefix=sen01 \
    --mtype=ASM \
    --multiple=100
```

S13. Operation: cameo-fba

Synopsis: Simulate a model using Flux Balance Analysis (FBA), with Cameo.

Usage: `python astools.py cameo-fba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-fba \
    --model=iJO1366 \
    --result_type=objective
```

S14. Operation: cameo-medium-cpds

Synopsis: List the medium in a model, with Cameo.

Usage: `python astools.py cameo-medium-cpds [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).

For example:

```
python astools.py cameo-medium-cpds --model=iAF1260
```

S15. Operation: cameo-medium-fba

Synopsis: Simulate a model after adding media change(s) using Flux Balance Analysis (pFBA), with Cameo.

Usage: `python astools.py cameo-medium-fba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- change: String to define medium change(s). Each change is defined as <compound ID>:<new value>. For example, EX_o2_e,0 will represent anaerobic condition. Multiple changes are delimited using semicolon.
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-medium-fba \
    --model=iAF1260 \
    --change=EX_o2_e,0;EX_glc__D_e,5.0 \
    --result_type=objective
```

S16. Operation: cameo-medium-pfba

Synopsis: Simulate a model after adding media change(s) using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. Molecular Systems Biology, 6(1):390.
http://www.ncbi.nlm.nih.gov/pubmed/20664636

Usage: `python astools.py cameo-medium-pfba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- change: String to define medium change(s). Each change is defined as <compound ID>:<new value>. For example, EX_o2_e,0 will represent anaerobic condition. Multiple changes are delimited using semicolon.
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-medium-pfba \
    --model=iAF1260 \
    --change=EX_o2_e,0;EX_glc__D_e,5.0 \
    --result_type=objective
```

S17. Operation: cameo-mutant-fba

Synopsis: Simulate a model after adding mutation(s) using Flux Balance Analysis (FBA), with Cameo.

Usage: `python astools.py cameo-mutant-fba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- mutation: String to define mutation(s). Each mutation is defined as <reaction ID>:<upper bound>:<lower bound>. For example, RBFK,0,0 will represent a knock out. Multiple mutations are delimited using semicolon.
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-mutant-fba \
    --model=iJO1366 \
    --mutation=NNAM,100,0;RBFK,0,0 \
    --result_type=objective
```

S18. Operation: cameo-mutant-pfba

Synopsis: Simulate a model after adding mutation(s) using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. Molecular Systems Biology, 6(1):390.
http://www.ncbi.nlm.nih.gov/pubmed/20664636

Usage: `python astools.py cameo-mutant-pfba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- mutation: String to define mutation(s). Each mutation is defined as <reaction ID>:<upper bound>:<lower bound>. For example, RBFK,0,0 will represent a knock out. Multiple mutations are delimited using semicolon.
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-mutant-pfba \
    --model=iJO1366 \
    --mutation=NNAM,100,0;RBFK,0,0 \
    --result_type=objective
```

S19. Operation: cameo-pfba

Synopsis: Simulate a model using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. Molecular Systems Biology, 6(1):390.
http://www.ncbi.nlm.nih.gov/pubmed/20664636

Usage: `python astools.py cameo-pfba [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).
- result_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

For example:

```
python astools.py cameo-pfba \
    --model=iJO1366 \
    --result_type=objective
```

S20. Operation: cameo-rxn-cpds

Synopsis: List the reactants and products for each reaction in a model, with Cameo.

Usage: `python astools.py cameo-rxn-cpds [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).

For example:

```
python astools.py cameo-rxn-cpds --model=iJO1366
```

S21. Operation: cameo-rxn-names

Synopsis: List the reaction names in a model, with Cameo.

Usage: `python astools.py cameo-rxn-names [option]`

where `[option]` can be

- model: Model acceptable by Cameo (see http://cameo.bio/02-import-models.html).

For example:

```
python astools.py cameo-rxn-names --model=iJO1366
```

S22. Example: Glycolysis Model

```
Filename: glycolysis_manuscript.modelspec

[Specification]
type: 1

[Identifiers]
name: glycolysis
author: Maurice Ling

[Objects]
glucose: D-glucose
g6p: a-D-Glucose-6-phosphate
f6p: b-D-Fructose-6-phosphate
f16p: b-D-Fructose-1,6-phosphate
gadp: D-glyceraldehyde 3-phosphate
dhap: Dihydroxyacetone phosphate
bpg13: D-1,3-bisphosphoglycerate
pg3: 3-phosphoglycerate
pg2: 2-phosphoglycerate
pep: phosphoenolpyruvate
pyr: pyruvate

[Initials]
glucose: 1e-3
g6p: 1e-6
f6p: 1e-6
f16p: 1e-6
gadp: 1e-6
dhap: 1e-6
bpg13: 1e-6
pg3: 1e-6
pg2: 1e-6
pep: 1e-6
pyr: 1e-6
```

```
[Variables]
hk: 1e-6
pgi: 1e-6
pfk: 1e-6
aldo: 1e-6
tpi: 1e-6
gapdh: 1e-6
pkg: 1e-6
pgm: 1e-6
eno: 1e-6
pk: 1e-6

[Reactions]
r1: glucose -> g6p | ${Variables:hk} * glucose
r2: g6p -> f6p | ${Variables:pgi} * g6p
r3: f6p -> f16p | ${Variables:pfk} * f6p
r4: f16p -> gadp + dhap | ${Variables:aldo} * f16p
r5: dhap -> gadp | ${Variables:tpi} * dhap
r6: gadp -> bpg13 | ${Variables:gapdh} * gadp
r7: bpg13 -> pg3 | ${Variables:pkg} * bpg13
r8: pg3 -> pg2 | ${Variables:pgm} * pg3
r9: pg2 -> pep | ${Variables:eno} * pg2
r10: pep -> pyr | ${Variables:pk} * pep
```

S23. Example: Pentose Phosphate Pathway Model

```
Filename: ppp_manuscript.modelspec

[Specification]
type: 1

[Identifiers]
name: pentose phosphate
author: Maurice Ling

[Objects]
g6p: a-D-Glucose-6-phosphate
pgl6: 6-phosphogluconolactone
pg6: 6-phosphogluconate
r5p: ribulose-5-phosphate
ri5p: ribose-5-phosphate
x5p: xylulose-5-phosphate
gly3p: glyceraldehyde-4-phosphate
sedo7p: sedoheptulose-7-phosphate
f6p: b-D-Fructose-6-phosphate
ery4p: erythose-4-phosphate
gadp: D-glyceraldehyde 3-phosphate

[Initials]
g6p: 1e-6
pgl6: 1e-6
```

```
pg6: 1e-6
r5p: 1e-6
ri5p: 1e-6
x5p: 1e-6
gly3p: 1e-6
sedo7p: 1e-6
f6p: 1e-6
ery4p: 1e-6
gadp: 1e-6

[Variables]
g6pd: 1e-6
gl: 1e-6
pgd6: 1e-6
r5pi: 1e-6
r5pe: 1e-6
th: 1e-6
ta: 1e-6

[Reactions]
r1: g6p -> pgl6 | ${Variables:g6pd} * g6p
r2: pgl6 -> pg6 | ${Variables:gl} * pgl6
r3: pg6 -> r5p | ${Variables:pgd6} * pg6
r4: r5p -> ri5p | ${Variables:r5pi} * r5p
r5: r5p -> x5p | ${Variables:r5pe} * r5p
r6: ri5p + x5p -> gly3p + sedo7p | ${Variables:th} * ri5p * x5p
r7: gly3p + sedo7p -> f6p + ery4p | ${Variables:ta} * gly3p *
sedo7p
r8: ery4p + x5p -> gadp + f6p | ${Variables:th} * ery4p * x5p
```

S24. Example: Merged Glycolysis / Pentose Phosphate Pathway

Glycolysis model and pentose phosphate pathway model were merged using the following command to yield `glycolysis_ppp.modelspec` as the merged model file:

```
python astools.py mergeASM \
    --prefix=exp \
    --
modelfile=models/asm/glycolysis_manuscript.modelspec;models/as
m/ppp_manuscript.modelspec \
    --outputfile=models/asm/glycolysis_ppp.modelspec

Filename: glycolysis_ppp.modelspec

[Specification]
type = 1

[Identifiers]
name = glycolysis
author = Maurice Ling
name_1 = pentose phosphate
```

```
author_1 = Maurice Ling

[Objects]
glucose = D-glucose
g6p = a-D-Glucose-6-phosphate
f6p = b-D-Fructose-6-phosphate
f16p = b-D-Fructose-1,6-phosphate
gadp = D-glyceraldehyde 3-phosphate
dhap = Dihydroxyacetone phosphate
bpg13 = D-1,3-bisphosphoglycerate
pg3 = 3-phosphoglycerate
pg2 = 2-phosphoglycerate
pep = phosphoenolpyruvate
pyr = pyruvate
pgl6 = 6-phosphogluconolactone
pg6 = 6-phosphogluconate
r5p = ribulose-5-phosphate
ri5p = ribose-5-phosphate
x5p = xylulose-5-phosphate
gly3p = glyceraldehyde-4-phosphate
sedo7p = sedoheptulose-7-phosphate
ery4p = erythose-4-phosphate

[Initials]
glucose = 1e-3
g6p = 1e-6
f6p = 1e-6
f16p = 1e-6
gadp = 1e-6
dhap = 1e-6
bpg13 = 1e-6
pg3 = 1e-6
pg2 = 1e-6
pep = 1e-6
pyr = 1e-6
pgl6 = 1e-6
pg6 = 1e-6
r5p = 1e-6
ri5p = 1e-6
x5p = 1e-6
gly3p = 1e-6
sedo7p = 1e-6
ery4p = 1e-6

[Variables]
hk = 1e-6
pgi = 1e-6
pfk = 1e-6
aldo = 1e-6
tpi = 1e-6
gapdh = 1e-6
```

```
pkg = 1e-6
pgm = 1e-6
eno = 1e-6
pk = 1e-6
g6pd = 1e-6
gl = 1e-6
pgd6 = 1e-6
r5pi = 1e-6
r5pe = 1e-6
th = 1e-6
ta = 1e-6

[Reactions]
exp1 = glucose -> g6p | ${Variables:hk} * glucose
exp2 = g6p -> f6p | ${Variables:pgi} * g6p
exp3 = f6p -> f16p | ${Variables:pfk} * f6p
exp4 = f16p -> gadp + dhap | ${Variables:aldo} * f16p
exp5 = dhap -> gadp | ${Variables:tpi} * dhap
exp6 = gadp -> bpg13 | ${Variables:gapdh} * gadp
exp7 = bpg13 -> pg3 | ${Variables:pkg} * bpg13
exp8 = pg3 -> pg2 | ${Variables:pgm} * pg3
exp9 = pg2 -> pep | ${Variables:eno} * pg2
exp10 = pep -> pyr | ${Variables:pk} * pep
exp11 = g6p -> pgl6 | ${Variables:g6pd} * g6p
exp12 = pgl6 -> pg6 | ${Variables:gl} * pgl6
exp13 = pg6 -> r5p | ${Variables:pgd6} * pg6
exp14 = r5p -> ri5p | ${Variables:r5pi} * r5p
exp15 = r5p -> x5p | ${Variables:r5pe} * r5p
exp16 = ri5p + x5p -> gly3p + sedo7p | ${Variables:th} * ri5p *
x5p
exp17 = gly3p + sedo7p -> f6p + ery4p | ${Variables:ta} * gly3p
* sedo7p
exp18 = ery4p + x5p -> gadp + f6p | ${Variables:th} * ery4p *
x5p
```

S25. Example: Glycolysis / Pentose Phosphate Pathway Simulation Script

The simulation script, `glycolysis_ppp.py`, is generated from the merged model specification, `glycolysis_ppp.modelspec`, using the following command:

```
python astools.py genODE \
    --modelfile=models/asm/glycolysis_ppp.modelspec \
    --mtype=ASM \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --lowerbound=0;0 \
    --upperbound=1e-3;1e-3 \
    --odefile=glycolysis_ppp.py
```

```
Filename: glycolysis_ppp.py

''' ------------------------------------
Python ODE script generated by ASModeller
(a package in AdvanceSynToolKit)

Date Time: 2020-6-7 13:15:15:218630

name: glycolysis
author: Maurice Ling
name_1: pentose phosphate
author_1: Maurice Ling
------------------------------------ '''

def glucose(t, y):
    exp1 = 1e-6 * y[0]
    return (0) - (exp1)

def g6p(t, y):
    exp1 = 1e-6 * y[0]
    exp2 = 1e-6 * y[1]
    exp11 = 1e-6 * y[1]
    return (exp1) - (exp2 + exp11)

def f6p(t, y):
    exp2 = 1e-6 * y[1]
    exp17 = 1e-6 * y[16] * y[17]
    exp18 = 1e-6 * y[18] * y[15]
    exp3 = 1e-6 * y[2]
    return (exp2 + exp17 + exp18) - (exp3)

def f16p(t, y):
    exp3 = 1e-6 * y[2]
    exp4 = 1e-6 * y[3]
    return (exp3) - (exp4)

def gadp(t, y):
    exp4 = 1e-6 * y[3]
    exp5 = 1e-6 * y[5]
    exp18 = 1e-6 * y[18] * y[15]
    exp6 = 1e-6 * y[4]
    return (exp4 + exp5 + exp18) - (exp6)

def dhap(t, y):
    exp4 = 1e-6 * y[3]
    exp5 = 1e-6 * y[5]
    return (exp4) - (exp5)

def bpg13(t, y):
    exp6 = 1e-6 * y[4]
    exp7 = 1e-6 * y[6]
```

```
    return (exp6) - (exp7)

def pg3(t, y):
    exp7 = 1e-6 * y[6]
    exp8 = 1e-6 * y[7]
    return (exp7) - (exp8)

def pg2(t, y):
    exp8 = 1e-6 * y[7]
    exp9 = 1e-6 * y[8]
    return (exp8) - (exp9)

def pep(t, y):
    exp9 = 1e-6 * y[8]
    exp10 = 1e-6 * y[9]
    return (exp9) - (exp10)

def pyr(t, y):
    exp10 = 1e-6 * y[9]
    return (exp10) - (0)

def pgl6(t, y):
    exp11 = 1e-6 * y[1]
    exp12 = 1e-6 * y[11]
    return (exp11) - (exp12)

def pg6(t, y):
    exp12 = 1e-6 * y[11]
    exp13 = 1e-6 * y[12]
    return (exp12) - (exp13)

def r5p(t, y):
    exp13 = 1e-6 * y[12]
    exp14 = 1e-6 * y[13]
    exp15 = 1e-6 * y[13]
    return (exp13) - (exp14 + exp15)

def ri5p(t, y):
    exp14 = 1e-6 * y[13]
    exp16 = 1e-6 * y[14] * y[15]
    return (exp14) - (exp16)

def x5p(t, y):
    exp15 = 1e-6 * y[13]
    exp16 = 1e-6 * y[14] * y[15]
    exp18 = 1e-6 * y[18] * y[15]
    return (exp15) - (exp16 + exp18)

def gly3p(t, y):
    exp16 = 1e-6 * y[14] * y[15]
    exp17 = 1e-6 * y[16] * y[17]
```

```
    return (exp16) - (exp17)

def sedo7p(t, y):
    exp16 = 1e-6 * y[14] * y[15]
    exp17 = 1e-6 * y[16] * y[17]
    return (exp16) - (exp17)

def ery4p(t, y):
    exp17 = 1e-6 * y[16] * y[17]
    exp18 = 1e-6 * y[18] * y[15]
    return (exp17) - (exp18)

def boundary_checker(y, boundary, type):
    '''
    Private function - called by ODE solvers to perform boundary
checking
    of variable values and reset them to specific values if the
original
    values fall out of the boundary values.

    Boundary parameter takes the form of a dictionary with
variable number
    as key and a list of [<boundary value>, <value to set if
boundary is
    exceeded>]. For example, the following dictionary for lower
boundary
    (type = 'lower') {'1': [0.0, 0.0], '5': [2.0, 2.0]} will set
the lower
    boundary of variable y[0] and [5] to 0.0 and 2.0 respectively.
This
    also allows for setting to a different value - for example,
{'1': [0.0,
    1.0]} will set variable y[0] to 2.0 if the original y[0]
value is
    negative.

    @param y: values for variables
    @type y: list
    @param boundary: set of values for boundary of variables
    @type boundary: dictionary
    @param type: the type of boundary to be checked, either
'upper' (upper
    boundary) or 'lower' (lower boundary)
    '''
    for k in list(boundary.keys()):
        if y[int(k)] < boundary[k][0] and type == 'lower':
            y[int(k)] = boundary[k][1]
        if y[int(k)] > boundary[k][0] and type == 'upper':
            y[int(k)] = boundary[k][1]
    return y
```

```
def RK4(funcs, x0, y0, step, xmax, nonODEfunc=None,
        lower_bound=None, upper_bound=None,
        overflow=1e100, zerodivision=1e100):
    '''
    Generator to integrate a system of ODEs, y' = f(x, y), using
fourth
    order Runge-Kutta method.

    A function (as nonODEfunc parameter) can be included to
modify one or
    more variables (y0 list). This function will not be an ODE
(not a
    dy/dt). This can be used to consolidate the modification of
one or
    more variables at each ODE solving step. For example, y[0]
= y[1] / y[2]
    can be written as

    >>> def modifying_function(y, step):
            y[0] = y[1] / y[2]
            return y

    This function must take 'y' (variable list) and 'step' (time
step) as
    parameters and must return 'y' (the modified variable list).
This
    function will execute before boundary checking at each time
step.

    Upper and lower boundaries of one or more variable can be
set using
    upper_bound and lower_bound parameters respectively. These
parameters
    takes the form of a dictionary with variable number as key
and a list
    of [<boundary value>, <value to set if boundary is
exceeded>]. For
    example, the following dictionary for lower boundary {'1':
[0.0, 0.0],
    '5': [2.0, 2.0]} will set the lower boundary of variable
y[0] and y[5]
    to 0.0 and 2.0 respectively. This also allows for setting
to a different
    value - for example, {'1': [0.0, 1.0]} will set variable
y[0] to 2.0 if
    the original y[0] value is negative.

    @param funcs: system of differential equations
    @type funcs: list
    @param x0: initial value of x-axis, which is usually
starting time
```

```
    @type x0: float
    @param y0: initial values for variables
    @type y0: list
    @param step: step size on the x-axis (also known as step in
calculus)
    @type step: float
    @param xmax: maximum value of x-axis, which is usually
ending time
    @type xmax: float
    @param nonODEfunc: a function to modify the variable list
(y0)
    @type nonODEfunc: function
    @param lower_bound: set of values for lower boundary of
variables
    @type lower_bound: dictionary
    @param upper_bound: set of values for upper boundary of
variables
    @type upper_bound: dictionary
    @param overflow: value (usually a large value) to assign in
event of
    over flow error (usually caused by a large number) during
integration.
    Default = 1e100.
    @type overflow: float
    @param zerodivision: value (usually a large value) to assign
in event
    of zero division error, which results in positive infinity,
during
    integration. Default = 1e100.
    @type zerodivision: float
    '''
    yield [x0] + y0
    def solver(funcs, x0, y0, step):
        n = len(funcs)
        f1, f2, f3, f4 = [0]*n, [0]*n, [0]*n, [0]*n
        y1 = [0]*n
        for i in range(n):
            try: f1[i] = funcs[i](x0, y0)
            except TypeError: pass
            except ZeroDivisionError: f1[i] = zerodivision
            except OverflowError: f1[i] = overflow
        for j in range(n):
            y1[j] = y0[j] + (0.5*step*f1[j])
        for i in range(n):
            try: f2[i] = funcs[i]((x0+(0.5*step)), y1)
            except TypeError: pass
            except ZeroDivisionError: f2[i] = zerodivision
            except OverflowError: f2[i] = overflow
        for j in range(n):
            y1[j] = y0[j] + (0.5*step*f2[j])
        for i in range(n):
```

```
            try: f3[i] = funcs[i]((x0+(0.5*step)), y1)
            except TypeError: pass
            except ZeroDivisionError: f3[i] = zerodivision
            except OverflowError: f3[i] = overflow
        for j in range(n):
            y1[j] = y0[j] + (step*f3[j])
        for i in range(n):
            try: f4[i] = funcs[i]((x0+step), y1)
            except TypeError: pass
            except ZeroDivisionError: f4[i] = zerodivision
            except OverflowError: f4[i] = overflow
        for i in range(n):
            try: y1[i] = y0[i] + (step * \
                    (f1[i] + (2.0*f2[i]) + (2.0*f3[i]) + f4[i])
/ 6.0)
            except TypeError: pass
            except ZeroDivisionError: y1[i] = zerodivision
            except OverflowError: y1[i] = overflow
        return y1
    while x0 < xmax:
        y1 = solver(funcs, x0, y0, step)
        if nonODEfunc:
            y1 = nonODEfunc(y1, step)
        if lower_bound:
            y1 = boundary_checker(y1, lower_bound, 'lower')
        if upper_bound:
            y1 = boundary_checker(y1, upper_bound, 'upper')
        y0 = y1
        x0 = x0 + step
        yield [x0] + y0

ODE = list(range(19))
ODE[0] = glucose
ODE[1] = g6p
ODE[2] = f6p
ODE[3] = f16p
ODE[4] = gadp
ODE[5] = dhap
ODE[6] = bpg13
ODE[7] = pg3
ODE[8] = pg2
ODE[9] = pep
ODE[10] = pyr
ODE[11] = pgl6
ODE[12] = pg6
ODE[13] = r5p
ODE[14] = ri5p
ODE[15] = x5p
ODE[16] = gly3p
ODE[17] = sedo7p
ODE[18] = ery4p
```

```
y = list(range(19))
y[0] = 1e-3    # glucose : D-glucose
y[1] = 1e-6    # g6p : a-D-Glucose-6-phosphate
y[2] = 1e-6    # f6p : b-D-Fructose-6-phosphate
y[3] = 1e-6    # f16p : b-D-Fructose-1,6-phosphate
y[4] = 1e-6    # gadp : D-glyceraldehyde 3-phosphate
y[5] = 1e-6    # dhap : Dihydroxyacetone phosphate
y[6] = 1e-6    # bpg13 : D-1,3-bisphosphoglycerate
y[7] = 1e-6    # pg3 : 3-phosphoglycerate
y[8] = 1e-6    # pg2 : 2-phosphoglycerate
y[9] = 1e-6    # pep : phosphoenolpyruvate
y[10] = 1e-6    # pyr : pyruvate
y[11] = 1e-6    # pgl6 : 6-phosphogluconolactone
y[12] = 1e-6    # pg6 : 6-phosphogluconate
y[13] = 1e-6    # r5p : ribulose-5-phosphate
y[14] = 1e-6    # ri5p : ribose-5-phosphate
y[15] = 1e-6    # x5p : xylulose-5-phosphate
y[16] = 1e-6    # gly3p : glyceraldehyde-4-phosphate
y[17] = 1e-6    # sedo7p : sedoheptulose-7-phosphate
y[18] = 1e-6    # ery4p : erythose-4-phosphate

labels = ['time', 'glucose', 'g6p', 'f6p', 'f16p', 'gadp',
'dhap', 'bpg13', 'pg3', 'pg2', 'pep', 'pyr', 'pgl6', 'pg6',
'r5p', 'ri5p', 'x5p', 'gly3p', 'sedo7p', 'ery4p']

lowerbound = {'0': [0.0, 0.0], '1': [0.0, 0.0], '2': [0.0, 0.0],
'3': [0.0, 0.0], '4': [0.0, 0.0], '5': [0.0, 0.0], '6': [0.0,
0.0], '7': [0.0, 0.0], '8': [0.0, 0.0], '9': [0.0, 0.0], '10':
[0.0, 0.0], '11': [0.0, 0.0], '12': [0.0, 0.0], '13': [0.0, 0.0],
'14': [0.0, 0.0], '15': [0.0, 0.0], '16': [0.0, 0.0], '17': [0.0,
0.0], '18': [0.0, 0.0], }

upperbound = {'0': [0.001, 0.001], '1': [0.001, 0.001], '2':
[0.001, 0.001], '3': [0.001, 0.001], '4': [0.001, 0.001], '5':
[0.001, 0.001], '6': [0.001, 0.001], '7': [0.001, 0.001], '8':
[0.001, 0.001], '9': [0.001, 0.001], '10': [0.001, 0.001], '11':
[0.001, 0.001], '12': [0.001, 0.001], '13': [0.001, 0.001], '14':
[0.001, 0.001], '15': [0.001, 0.001], '16': [0.001, 0.001], '17':
[0.001, 0.001], '18': [0.001, 0.001], }

model = RK4(ODE, 0.0, y, 1, 21600, None, lowerbound, upperbound)
```

S26. Example: Execution of Simulation Script

The above generated simulation script, `glycolysis_ppp.py`, is executed using the following command and the results file is `glycolysis_ppp_simulation.csv`:

```
python astools.py runODE \
    --odefile=glycolysis_ppp.py \
    --sampling=500 \
```

```
    --resultfile= glycolysis_ppp_simulation.csv
```

S27. Example: Running Sensitivity Analysis

Sensitivity analysis is executed to evaluate the effects of varying enzyme concentrations on the distribution of metabolites after 21600 seconds (6 hours) by varying each enzyme concentration one at a time. The parameters to vary for sensitivity analysis will be parameters listed in the `Variables` section of the model. The command is

```
python astools.py LSA \
    --modelfile=models/asm/glycolysis_ppp.modelspec \
    --prefix=sen01 \
    --mtype=ASM \
    --multiple=100 \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --cleanup=True \
    --outfmt=reduced \
    --resultfile= glycolysis_ppp_sensitivity_analysis.csv
```

S28. Example: Simplified Full Model from Yeoh et al. (2020)

```
Filename: yeoh_manuscript.modelspec

[Specification]
type: 1

[Identifiers]
name: simplified full model
author: Maurice Ling

[Objects]
glycerol: glycerol
oxygen: oxygen
OD: optical density
FA: ferulic acid
FAcoA: feruyl-co-A
Vanillin: vanillin

[Initials]
glycerol: 0.5
oxygen: 0.05
OD: 0.005
FA: 0.5
FAcoA: 0.0
Vanillin: 0.0

[Variables]
FCS: 0.05
ECH: 0.05
```

```
[Reactions]
r1: glycerol + oxygen -> OD | 0.16666 * (glycerol / (glycerol
+ 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
r2: oxygen -> | 2.0979e-3 - (0.333 * oxygen)
r3: FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA / (FA
+ 0.2913))
r4: FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))
```

S29. Example: Simplified Cell Growth Model from Yeoh et al. (2020)

```
Filename: yeoh_cell_growth_manuscript.modelspec

[Specification]
type: 1

[Identifiers]
name: simplied cell growth model
author: Maurice Ling

[Objects]
glycerol: glycerol
oxygen: oxygen
OD: optical density

[Initials]
glycerol: 0.5
oxygen: 0.05
OD: 0.005

[Variables]

[Reactions]
r1: glycerol + oxygen -> OD | 0.16666 * (glycerol / (glycerol
+ 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
r2: oxygen -> | 2.0979e-3 - (0.333 * oxygen)
```

S30. Example: Simplified Vanillin Production Model from Yeoh et al. (2020)

```
Filename: yeoh_vanillin_production_manuscript.modelspec

[Specification]
type: 1

[Identifiers]
name: simplied vanillin production model
author: Maurice Ling
```

```
[Objects]
FA: ferulic acid
FAcoA: feruyl-co-A
Vanillin: vanillin

[Initials]
FA: 0.5
FAcoA: 0.0
Vanillin: 0.0

[Variables]
FCS: 0.05
ECH: 0.05

[Reactions]
r1: FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA / (FA
+ 0.2913))
r2: FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))
```

S31. Example: Simplified Merged Model from Yeoh et al. (2020)

```
Filename: yeoh_merged_manuscript.modelspec

[Specification]
type = 1

[Identifiers]
name = simplied vanillin production model
author = Maurice Ling
name_1 = simplied cell growth model
author_1 = Maurice Ling

[Objects]
FA = ferulic acid
FAcoA = feruyl-co-A
Vanillin = vanillin
glycerol = glycerol
oxygen = oxygen
OD = optical density

[Initials]
FA = 0.5
FAcoA = 0.0
Vanillin = 0.0
glycerol = 0.5
oxygen = 0.05
OD = 0.005
```

```
[Variables]
FCS = 0.05
ECH = 0.05

[Reactions]
exp1 = FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA /
(FA + 0.2913))
exp2 = FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))
exp3 = glycerol + oxygen -> OD | 0.16666 * (glycerol /
(glycerol + 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
exp4 = oxygen -> | 2.0979e-3 - (0.333 * oxygen)
```