SOFTWARE

SEML: a semi-English modeling language for constructing biological models in Julia

Zhiping Zhang and Jeffrey D Varner*

*Correspondence:
jdv27@cornell.edu
Department of Chemical and Biomolecular Engineering, Cornell University, 14853 Ithaca NY, USA
Full list of author information is available at the end of the article

Abstract

Background: The advent of post-genomic era features the importance of mathematical modeling in biological research. Markup languages are usually adopted to store biological networks, and programming languages, such as Matlab and python, are used to set up models and run simulations. Due to the high threshold of programming, biological researchers are bothered by the plight of usefulness of sophisticated biological models and the fact that constructing sophisticated models from scratch requires a significant proficiency in any programming language.

Results: In this software note, we present a compiler that can translate the input in semi-English modeling language (SEML) into executable code in Matlab, Julia, Python2 or Python3. SEML is a language specified by our grammar but very similar to simplified English and is used to store biological networks instead of markup languages. The compiler takes the input and goes through a sequence of steps, including tokenization, lemmatization and semantic checking, to check and decode the input to generate intermediate form (IR) as a list of reactions, and finally generates from IR runnable code of kinetic model or flux balance analysis (FBA) model. Any syntax or semantic errors encountered would be reported at the end in a very readable format with helpful debugging information. We demonstrate this compiler by modeling a simple example network which contains both metabolism and gene regulation. Kinetic model was used to study species concentration profiles of the example network, and FBA model was used to study the flux through each reaction.

Conclusions: SEML provides a promising tool for writing and sharing human-readable biological models and can be expanded to model more complicated systems. Our compiler prototype can now handle input in simple English to satisfy basic modeling demands of researchers, and with further development has the potential to handle more natural inputs and generate more complicated models. This compiler is open source, available under an MIT license, and can be installed using the julia package manager from the GitHub repository.

Keywords: semi-English; modeling language; compiler; Julia; kinetic model; flux balance analysis

Background

The rapid development of 'omics' technologies brings science into the post-genomic era, enabling observation of biological systems at unprecedented resolution [1–3]. The large amount of data urges scientists to study biological phenomena in a more systematic manner. Mathematical modeling has long been used as an effective way to synthesize this information to unravel the complexity of biological networks.

There are many markup languages for representing quantitative models in systems biology, each with pros and cons. The Systems Biology Markup Language (SBML) [4–6] and Cell Markup Language (CellML) [7–10] are undoubtedly two of the most popular. Although SBML and CellML certainly provide common languages for sharing model files, they have two intrinsic problems. One is verbosity and low readability caused by their complicated data structures. For example, to express the reaction

$$M\_acald\_c + M\_coa\_c + M\_nad\_c \rightarrow M\_accoa\_c + M\_h\_c + M\_nadh\_c,$$

it needs about 20 lines in SBML [11]. Therefore, most readers would suffer from reading the modeling file directly if without appropriate assistance from other tools. The other disadvantage is the unavailability of semantic error checking, since there is no semantic information in SBML or CellML. To improve the readability of model files and reduce the communication barrier, a natural language based markup language would be a good supplement to the current modeling language community [7]. Models in natural language not only can guarantee syntactic correctness, like other markup languages, but also can express semantic relationships directly.

In this software note, we propose a semi-English modeling language (SEML) which is based on simplified English as a new way of writing biological models. SEML takes input in semi-English and generates code in multiple programming languages, including Matlab, Julia, Python2 and Python3. This tool enables biologists to build their own sophisticated models in an easier way, reducing the difficulty from writing a model from scratch to just write model in simple English and modify parameters.

Implementation

SEML is a modeling language, hence, it has its own grammar. It currently supports 11 types of sentences which contain a special type for specifying conversions between user-defined tags and SEML-defined tags, and 10 types of biological reactions. The full list of 10 biological reaction types and corresponding usages are:

- React: modeling the source or sink of species;
- Uptake: modeling the cross-membrane transportation from outside to inside of a cell;
- Secrete: modeling the cross-membrane transportation from inside to outside of a cell;
- Bind: modeling the binding of species;
- Unbind: modeling the dissociation of a complex;
- Phosphorylation: modeling the phosphorylation of species;
- Dephosphorylation: modeling the dephosphorylation of species;
- Activation: modeling the upregulation of gene expression and/or the promotion of enzyme activity;
- Inhibition: modeling the repression of gene expression and/or the inhibition of enzyme activity;
- Catalyze: modeling general biological reactions with or without enzyme.

The full grammar are available in Supplement.

The SEML compiler, implemented in Julia, takes input in SEML and goes through a sequence of steps as shown in Fig. 4 [12]. As a preparation step, a reserved words

creator takes in *synonyms.dat*, which lists all keywords and corresponding synonyms in grammar, to generate *reservedWords.jl*, which stores all keywords in a Julia dictionary. Given an input model in semi-English, the compiler takes seven steps to process the input to generate codes. A preprocessing step to get rid of all comments and blank lines in the input. Then, a tokenizer using recursive descent parser to break each sentence into tokens. In lemmatization, useful tokens are tagged with corresponding lemmas while unrelated tokens are thrown away. In information extraction step, the compiler identifies the reaction type, reactants, products and enzymes of a reaction, or regulators and target genes/proteins of a transcriptional/-translational statement. In semantic checking step, it checks semantic errors in the input. If no error occurs, intermediate representation (IR) generator generates IR of the input. In the final step, model generation, models in specified programming language is generated from IR.

Fig. 5(a) is a simple example of "uptake" type to show the work flow of SEML compiler. In preprocessing, the comment after "#" is deleted. The input sentence is decomposed into six tokens in tokenization step. In lemmatization, "BioSym" is assigned as the tag for all biological symbols, and keywords are reduced to their base form, while dummy tokens are thrown away. In information extraction, the compiler identifies three properties for each biological symbol: species type, unique name, and compartment. Then in IR generation, a reaction with stoichiometry is generated.

Fig. 5(b) shows a more complex example of "catalyze" type, which is exactly the reaction mentioned in Background part. The work flow is basically the same as previous Example, just notes that in lemmatization step, all logic connectors, e.g. comma, "and", are tagged as "BioSym" since they denote important logical information. In information extraction, each consecutive sequence of "BioSym" is treated as a group and undergoes logical analysis to figure out internal relationships. In this example, there are two groups, one is of reactants which consists of three species, the other is of product consisting of three species. And then, two reactions are generated since this sentence has "reversible" in the description.

Both examples do not show semantic checking since the inputs are made to be correct so as to generate the IR form. An example of semantic error would be using "g_D_c" instead of "p_D_c" in the second example ("g_" denotes gene), since "gene" is not treated as catalyst in a biochemical reaction. Full semantic error checking rule is available in Supplement. If any syntactic and/or semantic errors occur, the compiler will print out error messages, instead of going to IR generation.

In the final step, model generation, the compiler will generate kinetic model or flux balance analysis (FBA) and flux variability analysis (FVA) model, depending of user inputs [13, 14]. Users can also specify the output language for their convenience. This version of compiler supports output in Julia, Matlab, Python 2 and Python3.

Results and Discussion

To generate a kinetic or FBA model, one needs to write a semi-English description of the network and in terminal runs:

```
julia make_model.jl –m <path to the input file>
```

```
[-o <path to output directory> -s <host type>
-l <target programming language> -r <modeling framework>]
```

As a proof-of-concept demonstration, we tested SEML on an example network as shown in Fig.1. The semi-English description of the metabolism of example network used to generate FBA model is as shown in the third frame of Fig. 6. After running:

```
julia make_model.jl -m mikecase.txt -o FBAbmc
-l julia -r FBA
```

; We get a bunch of files under folder "FBAbmc". One can simply run "Solve.jl" to get a flux distribution as shown in Fig. 3. Simply modify the semi-English description, we can get the input for generating a kinetic model. After running:

```
julia make_model.jl -m mikecase.txt -o kineticsbmc
-l julia -r Kinetics
```

; we get a bunch of files under folder "kineticsbmc". Running "Simulations.jl" would generate concentration profiles of important species of this network, as Fig. 2. Substrate A is introduced at time $t = 3hr$ after the running system to steady steady, and removed at time $t = 13hr$. After removing the substrate, the system goes back to the original steady state, except that some substrate A is converted into extracellular products B and C.

SEML features a very helpful error checking function. Suppose instead of feeding the right description for FBA model to SEML compiler, one writes a model as in the first frame of Fig. 6 and feeds it to SEML, SEML will output error messages in terminal as shown in the 2nd frame of Fig. 6. It becomes much more easier for users to make the modification. The differences between two inputs are highlighted in red in the 3rd frame in Fig. 6.

SEML provides easy access to commonly used analysis tools in systems biology, enables users to set up FBA, FVA and/or kinetic models without painful experience of programming and debugging from scratch [15]. With its feature of semi-English language inputs, users with basic programming language knowledge can handle large models fastly without worrying much about details of code. There are several features that are under consideration for further development. Under the current framework, SEML can be expanded very easily to support more reaction types and incorporate more analysis tools. The current version supports only Michaelis-Menten kinetics, which is one of the most commonly used kinetic equations in enzyme kinetics, but it certainly would be better to expand SEML to support user-specified kinetics for each reaction or each type of reactions [16]. The IRs generated by SEML could be easily converted into SBML, which is widely identifiable by other tools and parsers [4, 17], hence, with simple modification, SEML can work as a frontend for most tools in systems biology so as to largely reduce the barrier of using them. Most importantly, we are also considering using other techniques, such as machine learning, to do preprocessing and tokenization, instead of knowledge-based technique that is adopted in this version, to support more natural input. This idea is tightly related to machine reading comprehension which is a hot research topic in computer science [18].

Conclusions

In this software note, we present SEML, a semi-English based modeling language compiler in Julia, which adopts compiler technology to support the writing and sharing of human-readable biological models in semi-English. SEML supports most commonly used analysis tools in systems biology, including FBA and FVA, and can generates codes in multiple programming languages, including Matalab and Python. SEML is written in Julia, whose feature of cross-platform, high-performance makes SEML easy to use and run fast in different operating systems [19]. We welcome both bug reports and requests for assistance on our Github page.

Availability and requirements

SEML is open source, available under an MIT software license. The SEML source code is freely available from the SEML GitHub repository at https://github.com/varnerlab/. All example codes used in this note are included in the example of the SEML GitHub repository. SEML can be run on all common operating system environments (Linux, Mac OS and Windows) with Julia pre-installed.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Text for this section . . .

Acknowledgements
Text for this section . . .

References

1. Wayman, J.A., Varner, J.D.: Biological systems modeling of metabolic and signaling networks. Current Opinion in Chemical Engineering **2**(4), 365–372 (2013)
2. Henry, C.S., DeJongh, M., Best, A.A., Frybarger, P.M., Linsay, B., Stevens, R.L.: High-throughput generation, optimization and analysis of genome-scale metabolic models. Nature biotechnology **28**(9), 977 (2010)
3. Lee, J.M., Gianchandani, E.P., Eddy, J.A., Papin, J.A.: Dynamic analysis of integrated signaling, metabolic, and regulatory networks. PLoS computational biology **4**(5), 1000086 (2008)
4. Keating, S.M., Le Novere, N.: Supporting sbml as a model exchange format in software applications. In: In Silico Systems Biology, pp. 201–225. Springer, ??? (2013)
5. Bornstein, B.J., Keating, S.M., Jouraku, A., Hucka, M.: Libsbml: an api library for sbml. Bioinformatics **24**(6), 880–881 (2008)
6. Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., *et al.*: The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. Bioinformatics **19**(4), 524–531 (2003)
7. Miller, A.K., Marsh, J., Reeve, A., Garny, A., Britten, R., Halstead, M., Cooper, J., Nickerson, D.P., Nielsen, P.F.: An overview of the cellml api and its implementation. BMC bioinformatics **11**(1), 178 (2010)
8. Garny, A., Nickerson, D.P., Cooper, J., dos Santos, R.W., Miller, A.K., McKeever, S., Nielsen, P.M., Hunter, P.J.: Cellml and associated tools and techniques. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **366**(1878), 3017–3043 (2008)
9. Moraru, I.I., Schaff, J.C., Slepchenko, B.M., Blinov, M., Morgan, F., Lakshminarayana, A., Gao, F., Li, Y., Loew, L.M.: Virtual cell modelling and simulation software environment. IET systems biology **2**(5), 352–362 (2008)
10. Resasco, D.C., Gao, F., Morgan, F., Novak, I.L., Schaff, J.C., Slepchenko, B.M.: Virtual cell: computational tools for modeling in cell biology. Wiley Interdisciplinary Reviews: Systems Biology and Medicine **4**(2), 129–140 (2012)
11. Orth, J.D., Fleming, R.M., Palsson, B.O.: Reconstruction and use of microbial metabolic networks: the core escherichia coli metabolic model as an educational guide. EcoSal plus (2010)
12. Cooper, K., Torczon, L.: Engineering a Compiler. Elsevier, ??? (2011)
13. Covert, M.W., Schilling, C.H., Palsson, B.: Regulation of gene expression in flux balance models of metabolism. Journal of theoretical biology **213**(1), 73–88 (2001)
14. Mahadevan, R., Edwards, J.S., Doyle III, F.J.: Dynamic flux balance analysis of diauxic growth in escherichia coli. Biophysical journal **83**(3), 1331–1340 (2002)
15. Gudmundsson, S., Thiele, I.: Computationally efficient flux variability analysis. BMC bioinformatics **11**(1), 489 (2010)
16. Ainsworth, S.: Michaelis-menten kinetics. In: Steady-state Enzyme Kinetics, pp. 43–73. Springer, ??? (1977)
17. Ebrahim, A., Lerman, J.A., Palsson, B.O., Hyduke, D.R.: Cobrapy: constraints-based reconstruction and analysis for python. BMC systems biology **7**(1), 74 (2013)

18. Jurafsky, D., Martin, J.H.: Speech and Language Processing vol. 3. Pearson London:, ??? (2014)
19. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM review **59**(1), 65–98 (2017)

Figures

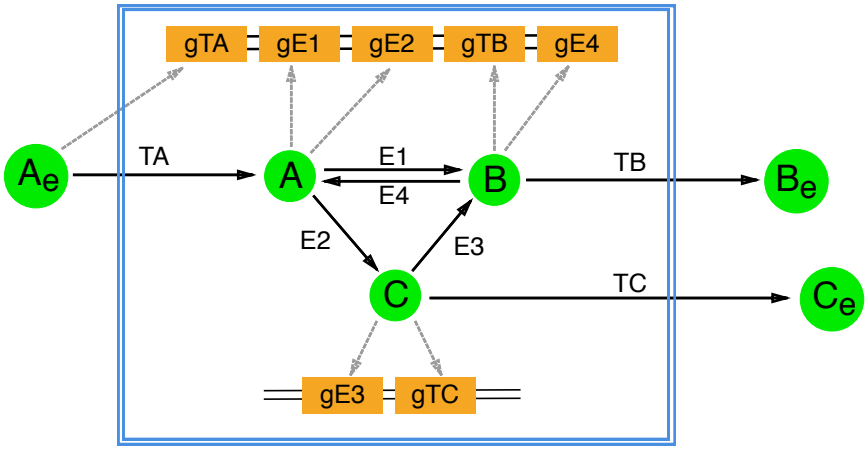


Figure 1 Example network. Metabolism: Substrate A is transported into the cell through membrane protein TA. A can be converted into B reversibly, with enzyme E1 catalyzing the forward reaction and enzyme E4 catalyzing the backward reaction. A can also be converted into C with E2 as the enzyme. C can convert into B by enzyme E3, or be secreted as extracellular product through membrane protein TC. Metabolite B is secreted via membrane protein TB as another extracellular product in this network. Transcriptional regulation: Substrate A can activate gene gTA which encodes protein TA. Metabolite A promotes the expression of genes gE1 and gE2 which encode protein E1 and E2, respectively. Metabolite B induces the expression of genes gTB and gE4 which encode protein TB and E4, respectively. Metabolite C promotes the expression of genes gE3 and gTC which encode protein E3 and TC, respectively.

Additional Files

Additional file 1 — Full grammar
Including notations, grammars and error checking rules.

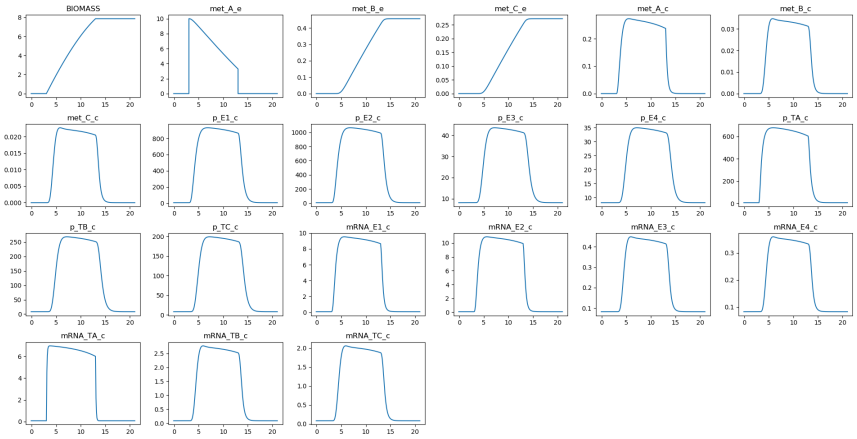


Figure 2 Simulation result from kinetic model of the example network. Substrate A was introduced at t = 2 hr, and removed at t = 4 hr.

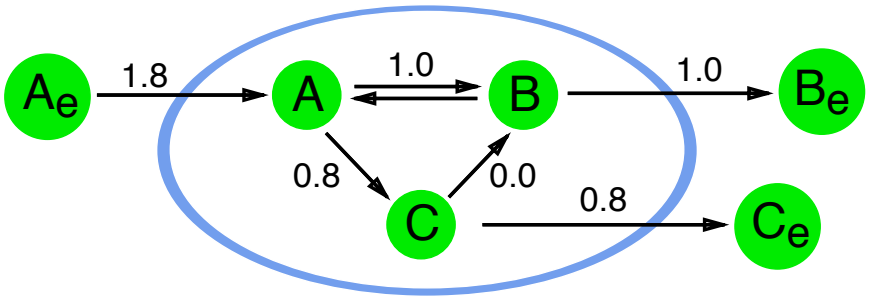


Figure 3 Simulation result from FBA model of the example network. The objective is to maximize the production of metabolites B and C while the maximum substrate uptake rate is 1 mmol/gDW/hr.

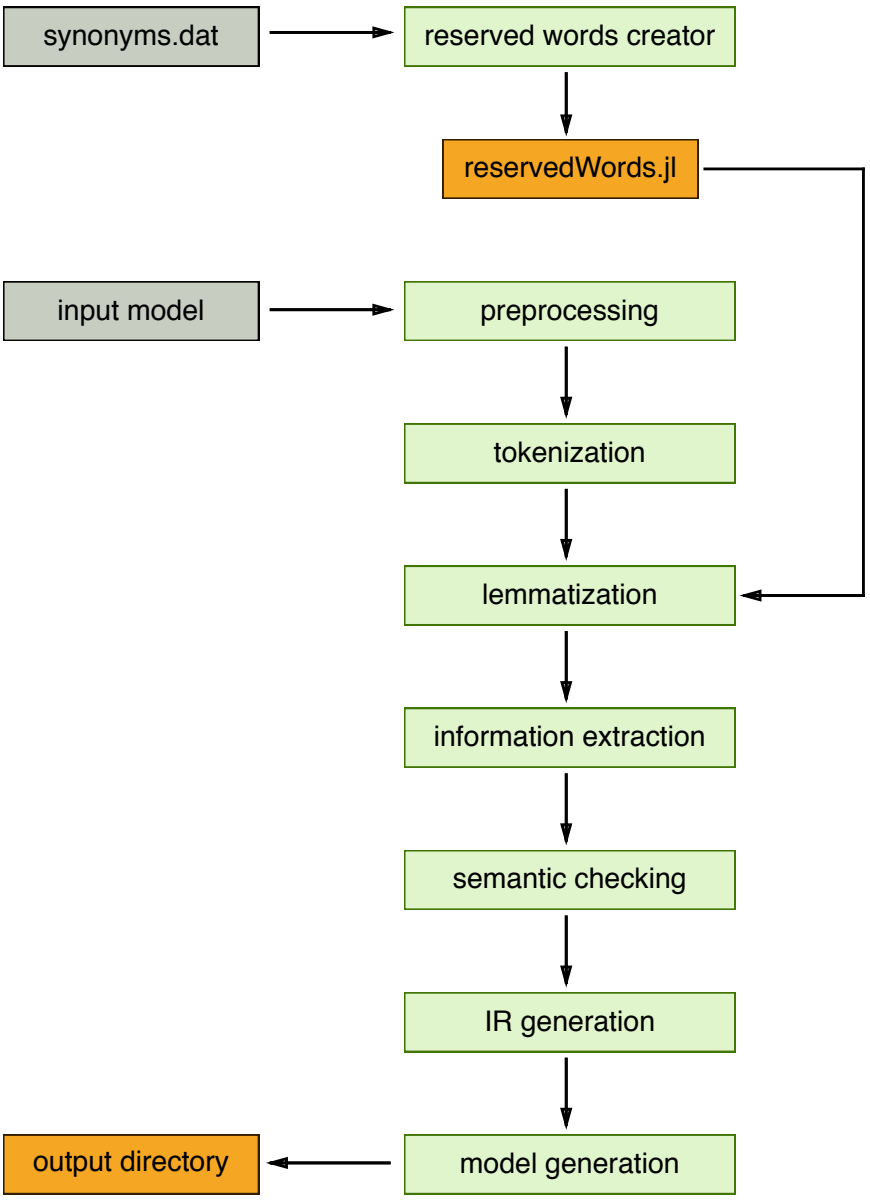
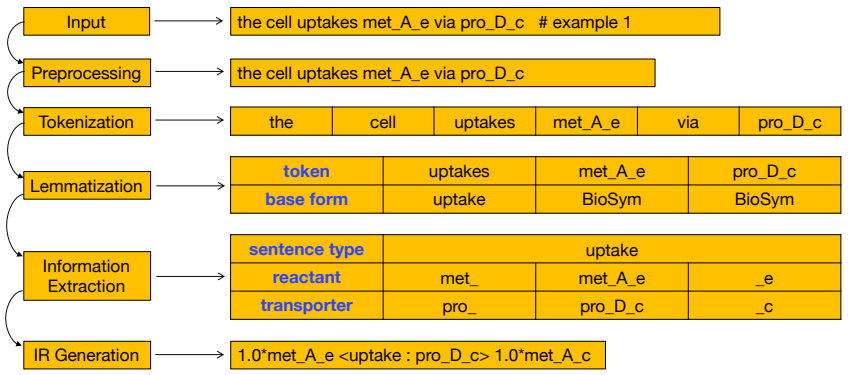
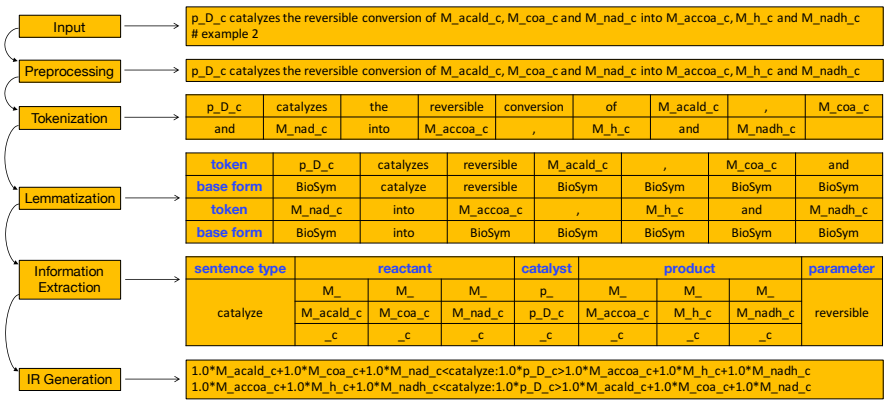


Figure 4 Compiler structure. As a preparation step, a reserved words creator takes in synonyms.dat, which lists all keywords and corresponding synonyms in grammar, to generate reservedWords.jl, which stores all keywords in a Julia dictionary. Given an input model in semi-English, the compiler takes 7 steps to process the input to generate codes. A preprocessing step to get rid of all comments and blank lines in the input. Then, a tokenizer using recursive descent parser to break each sentence into tokens. In lemmatization, useful tokens are tagged with corresponding lemmas while unrelated tokens are thrown away. In information extraction step, the compiler identifies the reaction type, reactants, products and enzymes of a reaction, or regulators and target genes/proteins of a transcriptional/translational statement. In semantic checking step, it checks semantic errors in the input. If no error occurs, IR generator generates IR (Intermediate Representation) of the input. In the final step, model generation, models in specified programming language is generated from IR.



(a) Example A



(b) Example B

Figure 5 Grammar examples. (a) This figure uses an example to show the workflow of the compiler. In preprocessing, note that comment is deleted. In lemmatization, "BioSym" is a tag for all biological symbols. In information extraction, each biological symbol has three properties: species type, unique name, and compartment. This figure doesn't show semantic checking since the input is correct so as to generate the IR form. (b) another example.

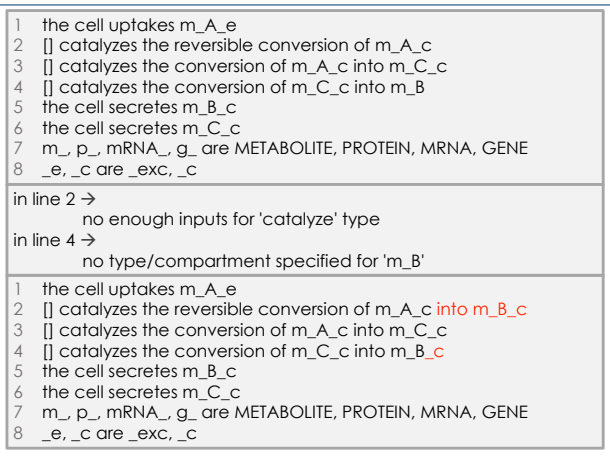


Figure 6 Error report. This figure shows the error report function of our compiler. Inside the first frame is the input of our FBA model but has some errors. The second frame shows the error report for input in the first frame. The third frame shows the correct input with modifications in red.