~~1. acronyms~~

~~2. no bullet point --> table: three columns (key + description+ example)~~

~~3. combine Fig.5 & 6 (top & buttom)~~

4. writing style: ref. a figure --> (Fig.#); cite --> at the end;

5. results & discussion --> about simulation plz

6. repository & webpage --> set up a new repository & webpage

7. FVA???

8. redraw Fig. 4 Compiler Structure: remove “reserved words creator”, add frontend & backend.

Fig 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.

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

Fig 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.

Fig 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.

Fig 5A. Grammar Example. 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.

Fig 5B. Another Grammar Example.

Fig 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.