reconstruction tutorial for cobra toolbox

Author(s): Maureen Carey

INTRODUCTION

In the following tutorial you will learn how to manipulate a (genome-scale) constraint-based metabolic reconstruction with CobraToolbox such that you can conform with the recommendations of **Box 2** [in the manuscript on community standards](). From the caption of **Box 2** itself: "Proposed minimum standardized content for a metabolic network reconstruction. We propose that modelers use this list as a guide to help standardize accessibility, content, and quality; however, more comprehensive documentation and more interpretablee and accessible information can only improve the usability and biological relevance of the shared reconstruction."

Throughout this tutorial we will look at and improve the example reconstruction `iPfal19.xml` for *Plasmodium falciparum* 3D7 provided with the publication, as well as building a minimal example from scratch where you can easily see the generated SBML. Each section will show the MATLAB code needed to inspect and manipulate a metabolic reconstruction.

A Note on Terminology: In the COBRA community, usually a distinction is made between a *reconstruction* and a *model*. A *reconstruction*represents the bare metabolic network of biochemical reactions that may occur in an organism due to the catalytic action of enzymes encoded in the genome plus some spontaneous reactions. A *model* is then a specification (or parametrization) of a reconstruction by, for example, defining certain medium conditions, i.e., restricting uptake rates; by fixing the directionality of certain reactions to fit those conditions; and by introducing specific energetic maintenance costs of the organism in those conditions. Other forms of model parametrization exist but these are some typical examples.

This is the consensus within the community, however, both the SBML element to encode the metabolic network and the CobraToolbox object are called *model*. They do not make an explicit distinction between those two representations. Within this tutorial we will therefore loosely call everything a model.

The following tutorial was created with the following MATLAB version:

```
version()
```

```
ans = 
'9.6.0.1174912 (R2019a) Update 5'
```

and Cobra Toolbox version:

```
initCobraToolbox
```

```
     _____   _____   _____   _____     _____     |
    /  ___| /  _  \ |  _  \ |  _  \   / ___ \    |   COnstraint-Based Reconstruction and Analysis
    | |     | | | | | |_| | | |_| |  | |___| |   |   The COBRA Toolbox - 2019
    | |     | | | | |  _  { |  _  /  |  ___  |   |
    | |___  | |_| | | |_| | | | \ \  | |   | |   |   Documentation:
```

```
    \_____| \_____/ |_____/ |_|  \_\ |_|   |_|   |   http://opencobra.github.io/cobratoolbox
                                                |

> Checking if git is installed ...  Done (version: 2.14.3).
> Checking if the repository is tracked using git ...  Done.
> Checking if curl is installed ...  Done.
> Checking if remote can be reached ...  Done.
> Initializing and updating submodules (this may take a while)... Done.
> Adding all the files of The COBRA Toolbox ...  Done.
> Define CB map output... set to svg.
> TranslateSBML is installed and working properly.
> Configuring solver environment variables ...
  - [----] ILOG_CPLEX_PATH: --> set this path manually after installing the solver ( see instructions )
  - [---*] GUROBI_PATH: /Library/gurobi801/mac64/matlab
  - [----] TOMLAB_PATH: --> set this path manually after installing the solver ( see instructions )
  - [----] MOSEK_PATH: --> set this path manually after installing the solver ( see instructions )
  Done.
> Checking available solvers and solver interfaces ... Done.
> Setting default solvers ... Done.
> Saving the MATLAB path ... Done.
  - The MATLAB path was saved in the default location.

> Summary of available solvers and solver interfaces

  Support     LP   MILP     QP   MIQP    NLP
-------------------------------------------------------------------
gurobi        active             1      1      1      1      -
ibm_cplex     active             0      0      0      0      -
tomlab_cplex  active             0      0      0      0      -
glpk          active             1      1      -      -      -
mosek         active             0      -      0      -      -
matlab        active             1      -      -      -      1
cplex_direct  active             0      0      0      -      -
dqqMinos      active             1      -      -      -      -
pdco          active             1      -      1      -      -
quadMinos     active             1      -      -      -      -
qpng          passive            -      -      1      -      -
tomlab_snopt  passive            -      -      -      -      0
lp_solve      legacy             1      -      -      -      -
-------------------------------------------------------------------
Total         -                  7      2      3      1      1

+ Legend: - = not applicable, 0 = solver not compatible or not installed, 1 = solver installed.


> You can solve LP problems using: 'glpk' - 'matlab' - 'pdco'
> You can solve MILP problems using: 'glpk'
> You can solve QP problems using: 'pdco' - 'qpng'
> You can solve MIQP problems using:
> You can solve NLP problems using: 'matlab'

> Checking for available updates ...
> The COBRA Toolbox is up-to-date.
```

1. Model

Load the existing metabolic model:

```
iPfal19 = readCbModel('iPfal19_cobratoolbox.xml')
```

```
iPfal19 = struct with fields:
```

```
                              S: [983×1233 double]
                           mets: {983×1 cell}
                              b: [983×1 double]
                         csense: [983×1 char]
                           rxns: {1233×1 cell}
                             lb: [1233×1 double]
                             ub: [1233×1 double]
                              c: [1233×1 double]
                      osenseStr: 'max'
                          genes: {475×1 cell}
                          rules: {1233×1 cell}
                      geneNames: {475×1 cell}
                      compNames: {5×1 cell}
                          comps: {5×1 cell}
                       proteins: {475×1 cell}
                    metCharges: [983×1 double]
                   metFormulas: {983×1 cell}
                      metNames: {983×1 cell}
                     metHMDBID: {983×1 cell}
                metInChIString: {983×1 cell}
                     metKEGGID: {983×1 cell}
                    metChEBIID: {983×1 cell}
                  metMetaNetXID: {983×1 cell}
                    metSBOTerms: {983×1 cell}
                       rxnNames: {1233×1 cell}
                   rxnECNumbers: {1233×1 cell}
                      rxnKEGGID: {1233×1 cell}
                   rxnMetaNetXID: {1233×1 cell}
                     rxnSBOTerms: {1233×1 cell}
                      subSystems: {1233×1 cell}
                     description: 'iPfal19_cobratoolbox.xml'
                    modelVersion: [1×1 struct]
                       modelName: 'iPfal19'
                         modelID: 'iPfal19_v1'
   geneisEuPathDB__46__genesID: {475×1 cell}
   metisbigg__46__metaboliteID: {983×1 cell}
                 metisbiocycID: {983×1 cell}
              metisinchikeyID: {983×1 cell}
             metislipidmapsID: {983×1 cell}
     metisseed__46__compoundID: {983×1 cell}
proteinisEuPathDB__46__genesID: {475×1 cell}
                rxnisAUTHORSID: {1233×1 cell}
             rxnisCONFIDENCEID: {1233×1 cell}
               rxnisCURATIONID: {1233×1 cell}
             rxnisEC_NUMBERID: {1233×1 cell}
         rxnisEC__32__NUMBERID: {1233×1 cell}
                   rxnisKEGGID: {1233×1 cell}
              rxnisREFERENCEID: {1233×1 cell}
            rxnisREFERENCESID: {1233×1 cell}
   rxnisbigg__46__metaboliteID: {1233×1 cell}
     rxnisbigg__46__reactionID: {1233×1 cell}
                 rxnisbiocycID: {1233×1 cell}
          rxnisiPfal17_notesID: {1233×1 cell}
               rxnisreactomeID: {1233×1 cell}
                   rxnisrheaID: {1233×1 cell}
```

Note: the other iPfal19 model in this repository includes cobraPy specific information. Only the 'without_annotation' version will load with CobraToolbox.

We can also creat an empty model for easier manipulation:

```
emptymodel = createModel()
```

```
emptymodel = struct with fields:
          rxns: {0×1 cell}
             S: []
            lb: [0×1 double]
            ub: [0×1 double]
             c: [0×1 double]
          mets: {0×1 cell}
             b: [0×1 double]
         rules: {0×1 cell}
         genes: {0×1 cell}
     osenseStr: 'max'
        csense: ''
    rxnGeneMat: []
```

You can inspect the existing compartments.

```
iPfal19.comps
```

```
ans = 5×1 cell array
    {'cytosol'      }
    {'extracellular'}
    {'apicoplast'   }
    {'mitochondria' }
    {'food_vacuole' }
```

These compartment will be abbreviated and added as a suffix to each metabolite to describe its location.

Since we have not yet added anything to our empty model, there are no compartments.

```
emptymodel.comps = cellstr(['cytosol      ';'extracellular'])
```

```
emptymodel = struct with fields:
          rxns: {0×1 cell}
             S: []
            lb: [0×1 double]
            ub: [0×1 double]
             c: [0×1 double]
          mets: {0×1 cell}
             b: [0×1 double]
         rules: {0×1 cell}
         genes: {0×1 cell}
     osenseStr: 'max'
        csense: ''
    rxnGeneMat: []
         comps: {2×1 cell}
```

Recognized naming convention

The existing model `iPfal17` follows the recommended practice for model identifiers. Quoted from **Box 2**:

recommended approach: `i + species indicator + iteration identifier`, e.g., iPfal17 for P. falciparum published in 2017

Let us inspect this the model ID:

```
iPfal19.modelID
```

```
ans =
'iPfal19_v1'
```

iPfal19*v1* = **P. fal**ciparum_ reconstruction published in (hopefully) 20**19**, version 1

Software versioning is also documented here:

```
iPfal19.modelVersion
```

```
ans = struct with fields:
      SBML_level: 3
    SBML_version: 1
     fbc_version: 2
```

Some model descripters are not yet supported as model features by Cobra Toolbox. This is why model README files or a COMBINE archive are useful. Check out the CobraPy tutorial for discussion of MIRIAM annotation such as:

1) Machine-readable reference to organism and species embedded via MIRIAM annotation

2) NCBI reference genome

3) Author(s) contact information embedded

However, some of this information can be stored in the notes section:

```
iPfal19.modelNotes = "This model is the third iteration of the asexual blood-stage Plas
```

```
iPfal19 = struct with fields:
                        S: [983×1233 double]
                     mets: {983×1 cell}
                        b: [983×1 double]
                   csense: [983×1 char]
                     rxns: {1233×1 cell}
                       lb: [1233×1 double]
                       ub: [1233×1 double]
                        c: [1233×1 double]
                osenseStr: 'max'
                    genes: {475×1 cell}
                    rules: {1233×1 cell}
                geneNames: {475×1 cell}
                compNames: {5×1 cell}
                    comps: {5×1 cell}
                 proteins: {475×1 cell}
              metCharges: [983×1 double]
            metFormulas: {983×1 cell}
```

```
                        metNames: {983×1 cell}
                       metHMDBID: {983×1 cell}
                   metInChIString: {983×1 cell}
                        metKEGGID: {983×1 cell}
                       metChEBIID: {983×1 cell}
                    metMetaNetXID: {983×1 cell}
                      metSBOTerms: {983×1 cell}
                         rxnNames: {1233×1 cell}
                      rxnECNumbers: {1233×1 cell}
                         rxnKEGGID: {1233×1 cell}
                      rxnMetaNetXID: {1233×1 cell}
                        rxnSBOTerms: {1233×1 cell}
                        subSystems: {1233×1 cell}
                       description: 'iPfal19_cobratoolbox.xml'
                     modelVersion: [1×1 struct]
                        modelName: 'iPfal19'
                          modelID: 'iPfal19_v1'
    geneisEuPathDB__46__genesID: {475×1 cell}
    metisbigg__46__metaboliteID: {983×1 cell}
                  metisbiocycID: {983×1 cell}
                metisinchikeyID: {983×1 cell}
               metislipidmapsID: {983×1 cell}
      metisseed__46__compoundID: {983×1 cell}
proteinisEuPathDB__46__genesID: {475×1 cell}
                  rxnisAUTHORSID: {1233×1 cell}
               rxnisCONFIDENCEID: {1233×1 cell}
                 rxnisCURATIONID: {1233×1 cell}
                rxnisEC_NUMBERID: {1233×1 cell}
           rxnisEC__32__NUMBERID: {1233×1 cell}
                     rxnisKEGGID: {1233×1 cell}
                rxnisREFERENCEID: {1233×1 cell}
               rxnisREFERENCESID: {1233×1 cell}
    rxnisbigg__46__metaboliteID: {1233×1 cell}
      rxnisbigg__46__reactionID: {1233×1 cell}
                  rxnisbiocycID: {1233×1 cell}
           rxnisiPfal17_notesID: {1233×1 cell}
                rxnisreactomeID: {1233×1 cell}
                    rxnisrheaID: {1233×1 cell}
                     modelNotes: "This model is the third iteration of the asexual blood-stage Plasmodi
```

Metabolite

A Note on Terminology: A tangent on the term *metabolite*: In the field of metabolic modeling four different terms are often used without clear distinction. There are compounds, chemicals, metabolites, and species. For the purpose of this document, we say 'metabolite' or 'species' to mean the most representative (most common) tautomer at the given pH. Implicitly, this acknowledges that interconvertible groups of tautomers may participate in a reaction. We will use 'compound' or 'chemical' to mean an exact chemical representation only.

An excellent resource for metabolites is MetaNetX. It merges information from several source databases (among them KEGG, ChEBI, BiGG) and aims to provide consistent cross references. At the time of writing, there is information on around 700 k compounds in MetaNetX.

For this tutorial, we will look at 1-dodecanoyl-sn-glycerol 3-phosphate. On the referenced page you can see that a lot of information that we will need is directly provided for us.

When creating a metabolite identifier, we should be aware of several conventions. As noted before, every metabolite must be allocated to a compartment. Here, we reference the cytosol by its short identifier `"c"`. Since

the same metabolite can appear in multiple compartments, it is common practice to add the compartment as a suffix to the identifier of the metabolite. Thus making the identifier unique within the model.

In principle, a metabolite identifier can be any string that satisfies the SBML constraints (which can be expressed by the following regular expression `[a-zA-Z_][a-zA-Z_0-9]*`). However, many modelers choose BiGG identifiers because they often resemble their common names and are thus easier to reason about quickly. We will use this convention here.

```
toymodel = addMetabolite(emptymodel, ...
                         '1ddecg3p[c]',  ...
                         'metName', '1_dodecanoyl_sn_glycerol_3_phosphate',  ...
                         'metFormula', 'C15H29O7P1',  ...
                         'ChEBIID', 'chebi:62840', ...
                         'KEGGId','',...
                         'PubChemID','',...
                         'InChi','InChI=1S/C15H31O7P/c1-2-3-4-5-6-7-8-9-10-11-15(17)21-
                         'charge',-2, 'b',0)
```

```
toymodel = struct with fields:
              rxns: {0×1 cell}
                 S: [1×0 double]
                lb: [0×1 double]
                ub: [0×1 double]
                 c: [0×1 double]
              mets: {'1ddecg3p[c]'}
                 b: 0
             rules: {0×1 cell}
             genes: {0×1 cell}
         osenseStr: 'max'
            csense: 'E'
        rxnGeneMat: []
             comps: {2×1 cell}
        metChEBIID: {'chebi:62840'}
       metCharges: -2
    metInChIString: {'InChI=1S/C15H31O7P/c1-2-3-4-5-6-7-8-9-10-11-15(17)21-12-14(16)13-22-23(18,19)20/h14,
        metKEGGID: {''}
      metPubChemID: {''}
       metFormulas: {'C15H29O7P1'}
          metNames: {'1_dodecanoyl_sn_glycerol_3_phosphate'}
```

Human readable, descriptive name

The full names are used when displaying more information about a metabolite. This is very helpful when the identifier is rather hard to guess, as is the case for our example, and it will often be the only identifying piece of information that biologists can work with.

```
indx = (strcmp('1ddecg3p[c]',toymodel.mets));
toymodel.metNames{indx}
```

```
ans =
'1_dodecanoyl_sn_glycerol_3_phosphate'
```

Charge and Chemical formula

The metabolite charge and formula are used to ensure that the model is mass balanced..

Load the existing metabolic model:

```
toymodel.metCharges
```

```
ans = -2
```

```
toymodel.metFormulas{indx}
```

```
ans =
'C15H29O7P1'
```

InChI strings

The InChI is a very information rich, unique description of a compound. In cobrapy we can provide it as an annotation to the metabolite.

```
toymodel.metInChIString{indx}
```

```
ans =
'InChI=1S/C15H31O7P/c1-2-3-4-5-6-7-8-9-10-11-15(17)21-12-14(16)13-22-23(18,19)20/h14,16H,2-13H2,1H3,(H2,18
```

At least one database identifier from a reliable resource

It might seem annoying and boring work but really: the more the merrier. When manually curating a model, keep in mind that it is relatively easy to add all of the annotations for one metabolite or reaction at a time. It is much harder to add annotations for hundreds of metabolites and reactions after the fact (e.g. explore iPfal19).

Fortunately, there are also tools that can help you automate this process! But, they might also introduce subtle mistakes.

More cross references are better because:

1. There are no one-to-one mappings of identifiers between identifiers and the more you use the better determined your metabolite is.
2. Other users of your model will have data in a myriad of formats. They will thank you deeply, if the identifier namespace of their data already exists in the model.

```
toymodel.metChEBIID{indx}
```

```
ans =
'chebi:62840'
```

SBO terms

The systems biology ontology (SBO) provides terms that can help specify the role of and allow reasoning about an element within the model. For metabolites we recommend to at least use the term SBO:0000247 for 'simple chemical' but other terms like polysaccharide might be more appropriate and informative.

```
temp = repmat({''},length(toymodel.mets),1);
toymodel.metSBOTerms = temp;
toymodel.metSBOTerms{indx} = "SBO:0000247"
```

```
toymodel = struct with fields:
              rxns: {0×1 cell}
                 S: [1×0 double]
                lb: [0×1 double]
                ub: [0×1 double]
                 c: [0×1 double]
              mets: {'1ddecg3p[c]'}
                 b: 0
             rules: {0×1 cell}
             genes: {0×1 cell}
         osenseStr: 'max'
            csense: 'E'
        rxnGeneMat: []
             comps: {2×1 cell}
        metChEBIID: {'chebi:62840'}
       metCharges: -2
    metInChIString: {'InChI=1S/C15H31O7P/c1-2-3-4-5-6-7-8-9-10-11-15(17)21-12-14(16)13-22-23(18,19)20/h14,
      metKEGGID: {''}
     metPubChemID: {''}
      metFormulas: {'C15H29O7P1'}
         metNames: {'1_dodecanoyl_sn_glycerol_3_phosphate'}
       metSBOTerms: {["SBO:0000247"]}
```

Biochemical reaction

Similarly to metabolites, MetaNetX is a great resource for biochemical reactions. Likewise, the identifiers easiest to interpret for human beings are BiGG symbols.

We will use phosphofructokinase as an example.

```
toymodel = addReaction(toymodel, ...
                       'PFK',...
                       'reactionName','phosphofructokinase',...
                       'reactionFormula','atp[c] + f6p[c] --> adp[c] + fdp[c] + h[c]',...
                       'reversible',false,...
                       'lowerBound',0,...
                       'upperBound',1000,...
                       'objectiveCoef',0,...
                       'subSystem','Glycolysis',...
                       'checkDuplicate',true)
```

```
PFK atp[c] + f6p[c]  -> adp[c] + fdp[c] + h[c]
toymodel = struct with fields:
              rxns: {'PFK'}
                 S: [6×1 double]
```

```
                lb: 0
                ub: 1000
                 c: 0
              mets: {6×1 cell}
                 b: [6×1 double]
             rules: {''}
             genes: {0×1 cell}
          osenseStr: 'max'
             csense: [6×1 char]
        rxnGeneMat: [1×0 double]
             comps: {2×1 cell}
        metChEBIID: {6×1 cell}
       metCharges: [6×1 double]
    metInChIString: {6×1 cell}
         metKEGGID: {6×1 cell}
      metPubChemID: {6×1 cell}
       metFormulas: {6×1 cell}
          metNames: {6×1 cell}
       metSBOTerms: {6×1 cell}
          rxnNames: {'phosphofructokinase'}
        subSystems: {{1×1 cell}}
           grRules: {''}
```

A reaction formula is automatically translated into stoichiometric coefficients and flux bounds. This can be modified at any point before, during, or after creation of the reaction.

Here, we can change that reaction to be reversible.

```
toymodel = addReaction(toymodel, ...
                       'PFK',...
                       'reactionName','phosphofructokinase',...
                       'reactionFormula','atp[c] + f6p[c] --> adp[c] + fdp[c] + h[c]',...
                       'reversible',true,...
                       'lowerBound',-1000,...
                       'upperBound',1000,...
                       'objectiveCoef',0,...
                       'subSystem','Glycolysis',...
                       'checkDuplicate',true)
```

```
Warning: Reaction with the same name already exists in the model, updating the reaction
PFK atp[c] + f6p[c]  <=> adp[c] + fdp[c] + h[c]
toymodel = struct with fields:
                rxns: {'PFK'}
                   S: [6×1 double]
                  lb: -1000
                  ub: 1000
                   c: 0
                mets: {6×1 cell}
                   b: [6×1 double]
               rules: {''}
               genes: {0×1 cell}
            osenseStr: 'max'
               csense: [6×1 char]
          rxnGeneMat: [1×0 double]
               comps: {2×1 cell}
          metChEBIID: {6×1 cell}
         metCharges: [6×1 double]
      metInChIString: {6×1 cell}
           metKEGGID: {6×1 cell}
```

```
      metPubChemID: {6×1 cell}
       metFormulas: {6×1 cell}
          metNames: {6×1 cell}
       metSBOTerms: {6×1 cell}
          rxnNames: {'phosphofructokinase'}
        subSystems: {{1×1 cell}}
           grRules: {''}
```

At least one database identifier from a reliable resource

Unfortunately, there is rarely a one-to-one relation between identifiers. However, like with metabolites, the more identifiers you include, the easier it is for a viewer to understand the reaction.

```
temp = repmat({''},length(toymodel.rxns),1);
toymodel.rxnMetaNetXID = temp;
indx = (strcmp('PFK',toymodel.rxns));
toymodel.rxnMetaNetXID{indx} = 'MNXR102507'
```

```
toymodel = struct with fields:
             rxns: {'PFK'}
                S: [6×1 double]
               lb: -1000
               ub: 1000
                c: 0
             mets: {6×1 cell}
                b: [6×1 double]
            rules: {''}
            genes: {0×1 cell}
        osenseStr: 'max'
           csense: [6×1 char]
       rxnGeneMat: [1×0 double]
            comps: {2×1 cell}
       metChEBIID: {6×1 cell}
       metCharges: [6×1 double]
   metInChIString: {6×1 cell}
        metKEGGID: {6×1 cell}
      metPubChemID: {6×1 cell}
       metFormulas: {6×1 cell}
          metNames: {6×1 cell}
       metSBOTerms: {6×1 cell}
          rxnNames: {'phosphofructokinase'}
        subSystems: {{1×1 cell}}
           grRules: {''}
    rxnMetaNetXID: {'MNXR102507'}
```

EC number

Just a reminder, 'EC' stands for 'Enzyme Commission'. At the risk of stating the obvious, only enzymes will have EC numbers and so a model is not expected to have an EC number for every reaction. Transporters, exchange reactions, and pseudoreactions will not have this annotation field.

```
temp = repmat({''},length(toymodel.rxns),1);
indx = (strcmp('PFK',toymodel.rxns));
toymodel.rxnECNumbers = temp;
toymodel.rxnECNumbers{indx} = '2.7.1.11'
```

```
toymodel = struct with fields:
              rxns: {'PFK'}
                 S: [6×1 double]
                lb: -1000
                ub: 1000
                 c: 0
              mets: {6×1 cell}
                 b: [6×1 double]
             rules: {''}
             genes: {0×1 cell}
         osenseStr: 'max'
            csense: [6×1 char]
       rxnGeneMat: [1×0 double]
             comps: {2×1 cell}
       metChEBIID: {6×1 cell}
       metCharges: [6×1 double]
   metInChIString: {6×1 cell}
        metKEGGID: {6×1 cell}
     metPubChemID: {6×1 cell}
      metFormulas: {6×1 cell}
         metNames: {6×1 cell}
      metSBOTerms: {6×1 cell}
         rxnNames: {'phosphofructokinase'}
       subSystems: {{1×1 cell}}
          grRules: {''}
    rxnMetaNetXID: {'MNXR102507'}
     rxnECNumbers: {'2.7.1.11'}
```

SBO

SBO terms for reactions are extremely useful in order to clearly distinguish a few categories of reactions without having to rely on naming conventions.

- Typical biochemical reactions should be annotated with SBO:0000176 or better yet with one of the more specific child terms.
- Transport reactions should receive SBO:0000655 or a more specific term. This obviates the need to append a `t` to a reaction identifier, as is often done for BiGG reactions such as PHEMEt.
- Exchange reactions should be annotated with SBO:0000627 rather than solely relying on an `EX_` identifier prefix.
- Demand reactions should be annotated with SBO:0000628 rather than solely relying on a `DM_` identifier prefix.
- Sink reactions should be annotated with SBO:0000632 rather than solely relying on an `SK_` identifier prefix.
- The ATP maintenance reaction should be labelled with SBO:0000630.
- All biomass reactions if any exist should be annotated with SBO:0000629.

```
temp = repmat({''},length(toymodel.rxns),1);
toymodel.rxnSBOTerms = temp;
toymodel.rxnSBOTerms{indx} = 'SBO:0000176'
```

```
toymodel = struct with fields:
              rxns: {'PFK'}
                 S: [6×1 double]
                lb: -1000
```

```
              ub: 1000
               c: 0
            mets: {6×1 cell}
               b: [6×1 double]
           rules: {''}
           genes: {0×1 cell}
       osenseStr: 'max'
          csense: [6×1 char]
      rxnGeneMat: [1×0 double]
           comps: {2×1 cell}
      metChEBIID: {6×1 cell}
      metCharges: [6×1 double]
  metInChIString: {6×1 cell}
       metKEGGID: {6×1 cell}
    metPubChemID: {6×1 cell}
     metFormulas: {6×1 cell}
        metNames: {6×1 cell}
     metSBOTerms: {6×1 cell}
        rxnNames: {'phosphofructokinase'}
     subSystems: {{1×1 cell}}
        grRules: {''}
   rxnMetaNetXID: {'MNXR102507'}
    rxnECNumbers: {'2.7.1.11'}
    rxnSBOTerms: {'SBO:0000176'}
```

Gene

Gene resources depend a lot on your organism. You may find information on MetaCyc, KEGG, NCBI, or more specialized databases. For p_falciparum, we are using plasmodb.org, a malaria-parasite database that is part of the EuPathDB project. Many automatic reconstruction pipelines will take a genome identifier and include genes for you in the draft reconstruction.

We will use one of the genes encoding for PFK as an example.

```
toymodel = changeGeneAssociation(toymodel, 'PFK', 'PF3D7_1128300 or PF3D7_0915400')
```

```
toymodel = struct with fields:
              rxns: {'PFK'}
                 S: [6×1 double]
                lb: -1000
                ub: 1000
                 c: 0
              mets: {6×1 cell}
                 b: [6×1 double]
             rules: {'x(2) | x(1)'}
             genes: {2×1 cell}
         osenseStr: 'max'
            csense: [6×1 char]
        rxnGeneMat: [1 1]
             comps: {2×1 cell}
        metChEBIID: {6×1 cell}
        metCharges: [6×1 double]
    metInChIString: {6×1 cell}
         metKEGGID: {6×1 cell}
      metPubChemID: {6×1 cell}
       metFormulas: {6×1 cell}
          metNames: {6×1 cell}
       metSBOTerms: {6×1 cell}
          rxnNames: {'phosphofructokinase'}
       subSystems: {{1×1 cell}}
```

```
           grRules: {'PF3D7_1128300 or PF3D7_0915400'}
      rxnMetaNetXID: {'MNXR102507'}
       rxnECNumbers: {'2.7.1.11'}
        rxnSBOTerms: {'SBO:0000176'}
```

```
toymodel.genes
```

```
ans = 2×1 cell array
    {'PF3D7_0915400'}
    {'PF3D7_1128300'}
```

Name and/or identifier

Note, there are no human-readable gene names, so we must add them.

```
%toymodel.geneNames
```

```
temp = repmat({''},length(toymodel.genes),1);
toymodel.geneNames = temp;
indx = (strcmp('PF3D7_1128300',toymodel.genes));
toymodel.geneNames{indx} = '6-phosphofructokinase'
```

```
toymodel = struct with fields:
              rxns: {'PFK'}
                 S: [6×1 double]
                lb: -1000
                ub: 1000
                 c: 0
              mets: {6×1 cell}
                 b: [6×1 double]
             rules: {'x(2) | x(1)'}
             genes: {2×1 cell}
         osenseStr: 'max'
            csense: [6×1 char]
        rxnGeneMat: [1 1]
             comps: {2×1 cell}
        metChEBIID: {6×1 cell}
       metCharges: [6×1 double]
    metInChIString: {6×1 cell}
         metKEGGID: {6×1 cell}
      metPubChemID: {6×1 cell}
       metFormulas: {6×1 cell}
          metNames: {6×1 cell}
       metSBOTerms: {6×1 cell}
          rxnNames: {'phosphofructokinase'}
        subSystems: {{1×1 cell}}
           grRules: {'PF3D7_1128300 or PF3D7_0915400'}
      rxnMetaNetXID: {'MNXR102507'}
       rxnECNumbers: {'2.7.1.11'}
        rxnSBOTerms: {'SBO:0000176'}
          geneNames: {2×1 cell}
```

```
toymodel.geneNames{indx}
```

```
ans =
'6-phosphofructokinase'
```

Note, (to the author's knowledge), Cobra Toolbox does not currently support the addition of other gene identifying information.

Generating all the annotations manually required quite a bit of online research in different databases. We would like to emphasize that a good reconstruction tool will provide you with a lot of this information thus saving you a lot of tedious annotation work. However, if you ever get tired of annotating your model, please consider that you doing it once correctly for your reconstruction will provide great value to the countless researchers applying your model in other scenarios.