Glycosylation Network Analysis Toolbox

(GNAT)

Reconstruction and analysis of reaction networks for Systems Glycobiology*

*** This is an integrated manual that includes all information related to GNAT V1.0 and 2.0**

By

Gang Liu and Sriram Neelamegham
Department of Chemical and Biological Engineering
State University of New York, Buffalo, NY 14260, USA.

Oct 17, 2013

Table of Contents

About this manual

- This is an integrated manual that includes information from both GNAT 1.0 and 2.0.
- Here, all functions are shown in bold ***italicized text***.
- All classes and variables appear in *plain italicized* fonts.
- All program code including examples appear in `green courier fonts.` The user may copy-paste these lines on the MATLAB command line after GNAT installation, in order to execute them.

  Source files used for these examples are stored in `<GNATInstallationDIR>`/toolbox/examplefiles.
- Program output appears in `red courier fonts`.
- For the purpose of manuscript review, major modifications made in version 2.0 are indicated by a border along the left margin. These borders will be removed after the review process is completed.

1. Introduction

Glycosylation is an important post-translational modification that alters a vast majority of mammalian secreted and cell surface proteins. Glycosylated proteins play structural and functional roles in diverse biological processes including inflammation, cancer and development. The recent development of advanced analytical tools has heralded the emergence of Glycomics, a field where entire glycomes of cell systems are characterized. These data are starting to be stored in glycosylation specific databases like the GlycomeDB (Ranzinger, Herget et al. 2011) and the Consortium for Functional Glycomics (CFG) website (Raman, Venkataraman et al. 2006). With the collection of large experimental datasets and repositories to store such information, opportunities exist for the development of biochemical reaction network models that can be employed to quantitatively analyze the emerging datasets. Such analysis of complex systems that focus on glycosylation processes is called *Systems Glycobiology* (Neelamegham and Liu 2011).

GNAT is an open source cross-platform MATLAB-based toolbox for Systems Glycobiology. It provides an integrated computational and visualization environment for the construction of glycans and their reaction networks. The following are salient operations enabled by the two versions of GNAT that have appeared thus far:

Components of GNAT 1.0

- Functions and classes to construct, manipulate and simulate glycans and their networks. This includes the ability to handle and manipulate glycans and reaction networks using various formats including glycoCT, glyde-II and LINUCS.
- Streamlined method to store glycan structure information in Systems Biology Markup Language (SBML) format files (Hucka, Finney et al. 2003). This enables sharing of glycosylation networks between investigators.
- Web-based querying of Glycomics databases enables collection of data that can be applied to refine reaction network model structures.
- High quality visualization of individual glycans and networks of glycosylation reactions in the MATLAB environment.
- Steady state and dynamic simulation of reaction networks.
- The above features are included in GNAT 1.0 (Liu, Puri et al. 2013)

Components of GNAT 2.0

- Enhanced database query functions that include the IUBMB (International Union of Biochemistry and Molecular Biology) nomenclature database for the construction of the enzyme (*Enz*) class.
- Enriched enzyme (*Enz*) and glycan class structures (*GlycanStruct*) that are geared towards integrating Glycomics experimental data into the GNAT environment. This includes newly-designed enzyme classes that handle glycosyltransferases (enzymes that transfer monosaccharides to a growing carbohydrate chain) and glycosidases (enzymes that remove monosaccharides from glycans). These classes have the ability to handle enzyme reactivity, stereospecificity and substrate specificity.

- Basic functionality to handle mass spectrometry data and to use these for the purpose of reaction network model generation.
- Methods for single reaction construction including: i. inference of a product given a substrate and enzyme ('product inference') and ii. inference of the reactant given a product and enzyme ('substrate inference').
- Methods for full network construction including: i. Product identification and network construction when starting glycan substrate and enzymes in a system are specified ('forward network inference'). ii. Network reconstruction when the final output glycan and enzymes are specified ('reverse network inference'). iii. Inference of intermediate substrates and reactions, when enzymes present in system, starting substrates and reaction products are specified ('connection network inference').
- Graph operations that facilitate species-specific and condition-specific cellular glycosylation pathway analysis. These operations: i. determine reaction paths linking two species in a network given a set of enzymes, ii. enable the identification of the connectivity between species including the determination of isolated reaction network elements, and iii. enable construction of subset models by either deleting a set of reactants, a set of reactions or by retaining salient aspects of the larger reaction network. As explained later, such operations are useful for advanced network analysis and model reduction operations.
- Extensive examples of the above.

What the program does NOT do

- Annotation of Glycomics and Glycoproteomic mass spectrometry data. Programs like SimGlycan, Cartoonist and GlycoWorkBench (Ceroni, Maass et al. 2008; Apte and Meitei 2010) perform these functions, but these other programs do not deal with the construction and simulation of glycosylation networks.

- Detailed simulations and analysis of biochemical reaction networks. Such functionality is enabled by programs like Copasi (Hoops, Sahle et al. 2006) and E-cell (Takahashi, Ishikawa et al. 2003), but these other programs are not suited to handle glycan structure data. GNAT only focuses on functions related to glycans and then uses various functions available in MATLAB for model analysis.

Tool	GNAT	COPASI/ E-CELL	SimGlycan/ Cartoonist/ GlycoWorkbench
SBML support	Yes	Yes	No
Network graphic visualization	Yes	Yes	No
ODE simulation	Yes	Yes	No
Glycan structure display	Yes	No	Yes
Glycan file format support	Yes	No	Yes
Glycosylation enzyme support	Yes	No	No
Mass spectra annotaion	No	No	Yes

2. Availability

GNAT is a free open-source MATLAB toolbox. It is:

- Written using MATLAB and Java.
- Supported in Windows, Linux and Mac OS platforms.
- Available from http://sourceforge.net/projects/gnatmatlab/files/.
- Detailed installation instructions are provided at the end of this manual.

3. Getting help on GNAT

Help documentation for GNAT can be obtained using the following:

1. This "GettingStarted.pdf" manual has detailed examples of individual commands.

2. Type "help <functionName>" or "help <className>" in MATLAB command window: This returns either the function description and usage, or class definition and associated syntax. Additional help regarding individual 'properties' and 'methods' can be obtained by clicking the corresponding link in the MATLAB command window.

❖ **Example 3.1: To get help on the glycosyltransferase class**

```
help GTEnz
```

❖ **Example 3.2: To get help on the inferGlyProd function**

```
help inferGlyProd
```

3. For similar information in the MATLAB help browser, type "doc <functionName>" or "doc <className>". This displays comment lines written into the GNAT source code.

4. Type "help <folderName>" in MATLAB command window to obtain information on the contents of that folder. This summarizes the information available in the contents.m file of that folder,

❖ **Example 3.3: To get help on the available classes**

```
help classes or doc classes
```

5. Type "echodemo <demofilename>" in MATLAB command window (in our case, "echodemo gnatdemo1", "echodemo gnatdemo2" … to… "echodemo gnatdemo7"). This runs the demo script as an echo-and-pause command line demo. The user can view the script, comments and results cell by cell. Buttons at the top of the window help to advance the demo to the next step or to stop it.

4. List of classes

4.1 Glycan, Enzyme and Network Classes

A variety of classes are introduced in GNAT to handle glycan and network objects (see list below). These classes can be classified into three groups that control features related to "Glycans", "Enzymes" and "Networks".

Classes related to glycans

GlycanStruct - a glycan sequence
Anomer - the orientation of anomeric hydroxyl group
GlycanBond - a glycosidic bond
GlycanLinkage - a glycan linkage between two glycan residues
MassOptions - data containing the options for mass calculation
GlycanResidue - a glycan residue
ResidueType - a type of glycan residue
RingType – a ring type for glycan residue
StereoConfig - a stereochemical property of the molecule

Classes related to enzymes

Enz - a generic enzyme
TFEnz - a transferase enzyme
GTEnz - a glycosyltransferase
HLEnz - a hydrolase enzyme
GHEnz - a glycosidases

Classes related to reaction network

GlycanNetModel – Network model including simulation results
Pathway - a biochemical reaction network
Compt - a compartment
Rxn - a single biochemical reaction or transport event
Species - a generic species in the network
GlycanSpecies – a glycan species that includes features from *Species* and *GlycanStruct*

4.2 Glycan, Enzyme and Network Object Manipulation

Manipulation of individual class properties using methods allows tailoring and refinement of glycosylation network models and individual glycans. The user is referred to use "help *classname*" (e.g. help *GlycanStruct*) or "doc *classname*" (doc *GlycanStruct*) command to look up all properties and methods related to individual classes. A few examples are given below:

4.2.1 Glycan object manipulation

This group of class operations deals with structure features related to individual glycans. This includes *GlycanStruct, GlycanResidue, GlycanLinkage, GlycanBond, Anomer, StereoConfig, RingType* and *ResidueType*. An example for *GlycanStruct* class is shown below:

❖ **Example 4.1: GlycanStruct/getnResidues get the number of residues of a glycan**

```
load man6_glycan.mat;     % load the predefined variable m6_glycan
nResidues = m6_glycan.getnResidues
getnResidues(m6_glycan);  % An alternate command for above
```

4.2.2 Enzyme object manipulation

This group of class objects allows the manipulation of enzyme objects, including glycosyltranferases and glycosidases

❖ **Example 4.2: To create an *Enz* structure using IUBMB website data and display the reaction catalyzed by this enzyme**

```
synthase = Enz([2;4;1;29]);
rxn = synthase.reaction
```

4.2.3 Network object manipulation

This group contains several classes that describe the elements composing the network including *GlycanNetModel*, *Pathway*, *Rxn*, *GlycanSpecies*, and *Compt*. An example for *GlycanNetModel* is below:

❖ **Example 4.3: GlycanNetModel /setName sets the name of the glycosylation network model**

```
load glycanNetSimExample.mat            %load predefined variable testGlycanNetModel
testGlycanNetModel.setName('testNetwork');
nameString=testGlycanNetModel.getName    % This returns the new name
```

5. List of functions

The functions of GNAT can be categorized into the seven groups below. Extensive documentation regarding each function can be obtained using the GNAT help command. **Chapters 6-12 provide background information and usage examples for each of these groups.**

5.1 Network Input/Output [Chapter 6]

File Input/Output
glycanMLread - read glycan structure file
glycanNetSBMLread - read glycosylation network SBML file
glycanMLwrite - write glycan structure file
glycanNetSBMLwrite - write glycosylation network SBML file
glycanStrread - read glycan structure string
glycanStrwrite - write glycan structure string

File Format Transformation
glycoctcond2Glyde - convert from GlycoCT-condensed format to GLYDE format
glycoctcond2Linucs - convert from GlycoCT-condensed format to LINUCS format
glycoct2Glycoctcond - convert from GlycoCT to GlycoCT-condensed format
glycoct2Linucs - convert from GlycoCT to LINUCS format
glycoctcond2Glycoct - convert from GlycoCT-condensed format to GlycoCT format
glycoctcond2Glyde - convert from GlycoCT-condensed format to GLYDE format
linucs2Glyde - convert from LINUCS to GLYDE format
linucs2Glycoct - convert from LINUCS to GlycoCT format
linucs2Glycoctcond - convert from LINUCS to GlycoCT-condensed format

SBML Function
addGlycanAnnotation - add glycan sequence annotation to SBML

5.2 Display glycans, enzymes & networks [Chapter 7]

glycanFileViewer - display glycan structure from glycan sequence file
glycanViewer - display glycan structure from GlycanStruct object
glycanNetFileViewer - display glycosylation network from file
glycanNetViewer - display glycosylation network from GlycanNetModel object
displayset - set display options
displayget - get display options
enzViewer - view properties of Enz or any other Enzyme class
glycanPathViewer - display glycosylation network from Pathway object *
glycanRxnViewer - view glycosylation reaction *

*** These functions are similar to glycanNetViewer only they accept *Pathway* objects and *Rxn* objects as inputs for visualization.**

5.3 Database Query [Chapter 8]

dbLocalConnect **- connect to local database
dbExactStructSearch ** - query local database by glycan sequence
queryGlycomeDB - query GlycomeDB by database ID
queryCFGDB - query CFGDB by database ID
queryIUBMBDB - query IUBMBDB by enzyme EC NO
enzdbmatload – load a local enzyme database from a MATLAB mat file
webGlycomicsDB - open a default web browser with search results from either GlycomeDB or the CFG database

** requires installation of the MATLAB Database Toolbox and local GlycomeDB database

5.4 Network Inference [chapter 9]

inferGlyProd - infer glycan product(s) based on enzyme and its substrate
inferGlySubstr - infer glycan substrate(s) based on enzyme and its product
inferGlyForwPath - construct glycosylation reaction network(s) based on the enzymes and single glycan as starting structure
inferGlyRevrPath - construct glycosylation reaction network(s) based on the enzymes and single glycan as an end-product
inferGlyConnPath - infer glycosylation reaction network(s) linking the starting glycans to end-product glycans based on enzymes

5.5 Graph Operation [chapter 10]

subnetgenbynumdel - generate subset network from the parent pathway by deleting specified number of species
subnetgenbyspecdel - generate subset network from the parent pathway by deleting specified species
subnetgenbyspeckeep - generate subset network from the parent pathway by keeping specified species
detectIsolatedSpecies - detect isolated species in the network
removeIsolatedSpecies - remove isolated species from the network
pathfinding - find a pathway between two species in the network

5.6 Mass Spectrometry Analysis [chapter 11]

readMS - read m/z and intensity from a raw data file in msd format
msprocess *** - process mass spectrometry data (m/z and intensity data) using a four-step method

*** requires installation of the MATLAB Bioinformatics Toolbox

5.7 Simulation [chapter 12]

glycanSSim - simulate glycosylation network under steady state
glycanDSim - simulate glycosylation network at a series of time points

6. Network Input/Output

GNAT supports the following glycan sequence formats:

- GlycoCT (Herget, Ranzinger et al. 2008)
- GlycoCT condensed form (Herget, Ranzinger et al. 2008)
- GLYDE (Sahoo, Thomas et al. 2005)
- LINUCS (Bohne-Lang, Lang et al. 2001)

Details regarding these formats can be seen in the references cited above. A variety of input-output functions, divided into three subgroups below, are enabled to handle these file formats. A series of examples are provided below in order to illustrate them:

A. **File input/output**: This subgroup provides six commands: ***glycanMLread***, ***glycanMLwrite***, ***glycanStrread***, ***glycanStrwrite***, ***glycanNetSBMLread*** and ***glycanNetSBMLwrite***. The function and usage of each command are as follows:

a) ***glycanMLread***/***glycanStrread*** reads the sequence of a glycan from the file or a MATLAB string variable. Below are two examples.

❖ **Example 6.1: *glycanMLread* reads glycan structure from a local file ('highmannose.glycoct_xml') and returns an object (m6_glycan) of MATLAB GlycanStruct class.**

```
m6_glycan = glycanMLread('highmannose.glycoct_xml','glycoct_xml');
```

❖ **Example 6.2: *glycanStrread* reads the glycan structure from a string (predefined variable m6glycanString_glycoct, stored in man6_glycan.mat) and return an object of MATLAB GlycanStruct class (m6_glycan). The MATLAB string in this case stores the xml structure.**

```
load  man6_glycan.mat;
m6_glycan = glycanStrread(m6glycanString_glycoct,'glycoct_xml');
```

b) ***glycanMLwrite***/**glycanStrwrite** commands write the sequence of a glycan to the file or a MATLAB string *variable*. Two examples are:

❖ **Example 6.3: *glycanMLwrite* writes the sequence of a glycan from a MATLAB GlycanStruct object to a local file ('m6_glycan.xml')**

```
load man6_glycan.mat;  % load the predefined variable m6_glycan
glycanMLwrite(m6_glycan,'m6_glycan.xml', 'glycoct_xml');
```

❖ **Example 6.4: *glycanStrwrite* writes the sequence of a glycan from a MATLAB GlycanStruct object to a string**

```
load man6_glycan.mat;    % load the predefined variable m6_glycan
glycanString_glycoct2=glycanStrwrite(m6_glycan,'glycoct_xml');
```

c) **glycanNetSBMLread** command parses an SBML file and returns a GlycanNetModel Object. This file and associated object describe an entire glycosylation reaction network. Note that XML based glycan structures are placed in the *annotation* field of the SBML *species element*. SBML read/write operations may take time due to the verbose nature of the document.

❖ **Example 6.5:**

```
glycanNetModel=glycanNetSBMLread('OlinkedModel_wtannot.xml');
```

d) **glycanNetSBMLwrite command** writes a GlycanNetModel Object to an SBML file. This is the inverse of the previous command.

❖ **Example 6.6:**

```
load glycanmodelexample.mat; % load a GlycanNetModel object"olinked_model"
glycanNetSBMLwrite(olinked_model, 'OlinkedModel_wtannot.xml');
```

B. File Transformation. Nine functions are provided in this category. These functions transform one format of glycan structure to another format in the MATLAB environment. ***glycoct2Glycoctcond***, ***glycoct2Glyde***, and ***glycoct2Linucs*** convert Glycoct format to Glycoctcond, Glyde and Linucs formats respectively. ***glycoctcond2Glycoct***, ***glycoctcond2Glyde***, and ***glycoctcond2Linucs*** convert Glycoctcond format to Glycoct, Glyde and Linucs respectively. ***linucs2Glycoct***, ***linucs2Glycoctcond*** and ***linucs2Glyde*** convert Linucs format to Glycoct, Glycoctcond and Glyde respectively. One example is shown below.

❖ **Example 6.7: glycoct2Linucs reads a glycan in the Glycoct format and returns a file in LINUCS format**

```
glycoct2Linucs('highmannose.glycoct_xml','highmannose.linucs');
```

C. SBML function. GNAT provides one function (***addGlycanAnnotation***) as a supplement to the SBML Toolbox functions for exporting network files. This command adds glycan XML structure to the *annotation* field in the *species element* of the SBML reaction network file. One example is below:

❖ **Example 6.8: *addGlycanAnnotation* adds glycan sequence to the annotation element in SBML structure or GlycanNetModel object**

```
linearNLinkedModel_SBML=TranslateSBML(which('gnat_test_woannot.xml'));
% SBML Toolbox command that reads SBML file to MATLAB workspace. The SBML
% file contains four species M9. M8, M7 and M6. We wish to associate these
% with xml structures specified below.

glycansMap = containers.Map; % create map structure

M9_glycan  = glycanMLread('M9.glycoct_xml');  % read glycan structure
M8_glycan  = glycanMLread('M8.glycoct_xml');  % read glycan structure
M7_glycan  = glycanMLread('M7.glycoct_xml');  % read glycan structure
M6_glycan  = glycanMLread('M6.glycoct_xml');  % read glycan structure

glycansMap('M9')=glycanStrwrite(M9_glycan);
glycansMap('M8')=glycanStrwrite(M8_glycan);
glycansMap('M7')=glycanStrwrite(M7_glycan);
glycansMap('M6')=glycanStrwrite(M6_glycan);
```

```
% create key-value pair linking species names to glycan object

NLinkedModel_SBMLAnnot=addGlycanAnnotation(linearNLinkedModel_SBML,...
glycansMap);
% Above adds annotation elements for M9-M6 species elements in SBML  structure

% To write this new model into an SBML format file use the GlycanNetModel
% constructor. Note that this command only works if all species in a given
% reaction are annotated and there are no missing elements.
GModel =  GlycanNetModel(NLinkedModel_SBMLAnnot);

% This can then be written to an xml file using:
glycanNetSBMLwrite(GModel,'GModel.xml');
```

7. Display glycans, enzymes and glycosylation networks

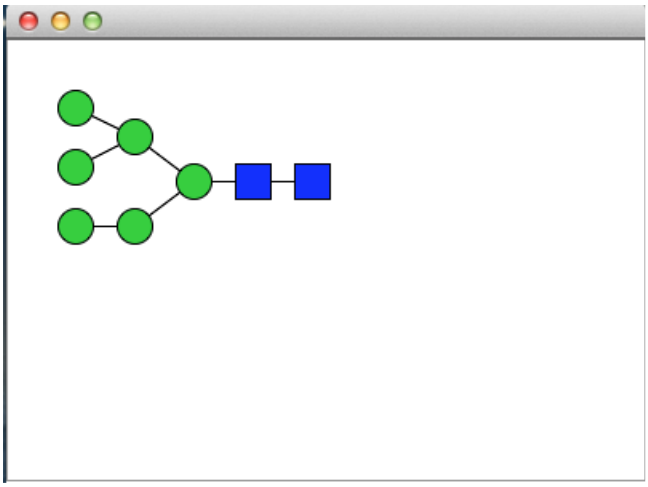
This category provides nine commands, viz. ***glycanFileViewer***, ***glycanViewer***, ***enzViewer***, ***glycanNetFileViewer***, ***glycanRxnViewer***, ***glycanPathViewer***, ***glycanNetViewer***, and ***displayset*** and ***displayget***. The first two functions, ***glycanFileViewer*** and ***glycanViewer*** can be used to view the glycan structure from the glycan structure file or a MATLAB variable. ***enzViewer*** displays properties related to the enzyme class definition. ***glycanNetFileViewer***, ***glycanRxnViewer***, ***glycanPathViewer*** and ***glycanNetViewer*** display the glycosylation reaction network along with the glycan structures. The input is either an SBML file or a MATLAB variable. The commands ***displayset*** and ***displayget*** either set or get the display options for the four display functions.

Below we show examples of ***glycanFileViewer***, ***enzViewer*** and ***glycanNetFileViewer***. The usage of other commands can be found using "help ***functionname***".

❖ **Example 7.1: *glycanFileViewer* reads the sequence of a glycan from a file or a MATLAB GlycanStruct object and return a graphical representation of its 2D structure. Default settings are used when options is not specified.**

```
glycanFileViewer('highmannose.glycoct_xml','glycoct_xml');
% This displays highmannose.glycoct_xml, a file in GlycoCT format
```

Output:

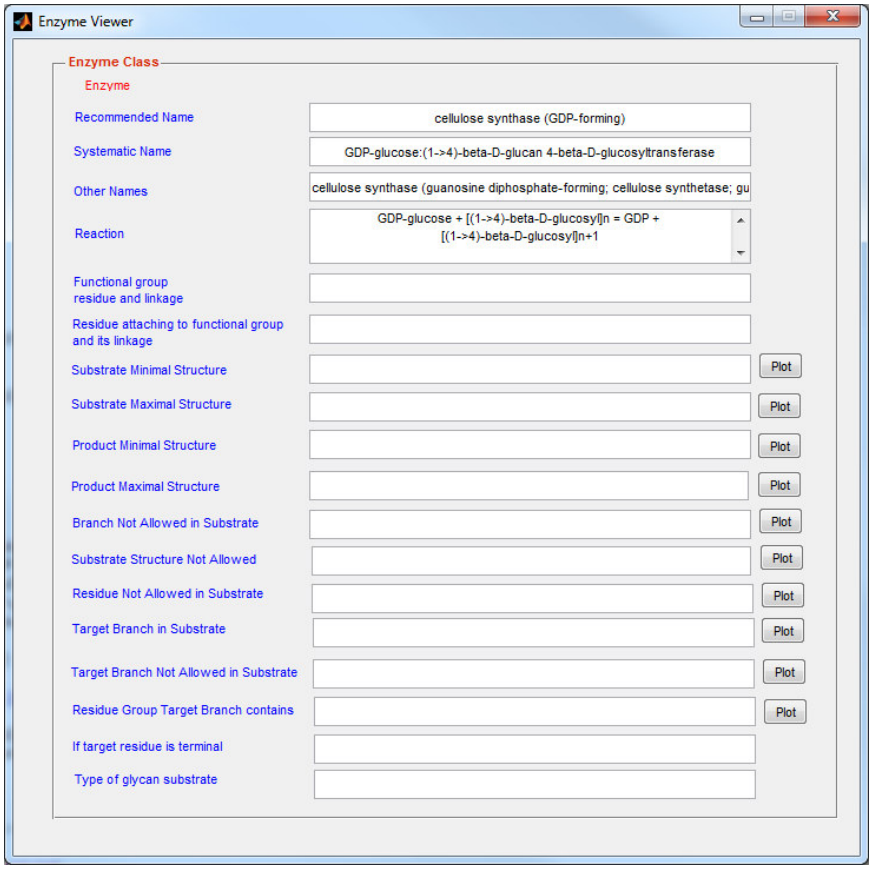


❖ **Example 7.2: enzViewer reads an enzyme object and returns a GUI window listing available enzyme properties defined in the object.**

Note: The enzViewer is the same for the parent *Enz* class and also subclasses which include *GTEnz* and *GHEnz*. In the case of *Enz*, only the first few fields are populated. Additional fields are included in *GTEnz* and *GHEnz* as described later in chapter 9.

```
synthase = Enz([2;4;1;29]);
enzViewer(synthase);
```

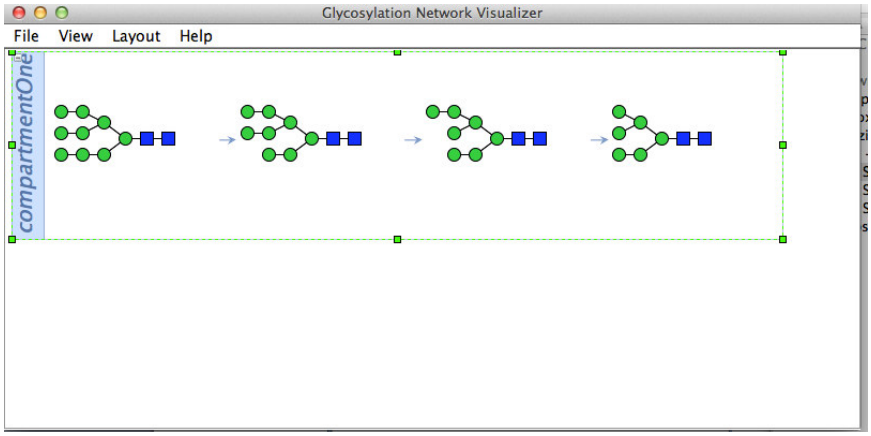
Output:



❖ **Example 7.3: glycanNetFileViewer reads an SBML file of a glycosylation network annotated with glycan structures and returns a graphical representation in a GUI window**

```
options=displayset('showRedEnd',false,'showLinkage',false,'netlayout','Compact
 Tree');                    % sets display options
glycanNetFileViewer('gnat_test_wtannot.xml','glycoct_xml',options);
                            % reads and displays network in SBML file
```

Output:



Note: The above figure illustrates the serial conversion of an N-glycan from a unit that contains nine mannose residues (‘M9’) to one that has six mannose residues ‘M5’

8. Database Query

Three options are available for querying databases:

1) query the online database with carbohydrate database ID to return results in default web browser. The function is ***webGlycomicsDB*.**

2) query the online database with carbohydrate database ID to retrieve results in MATLAB structures. The commands include ***queryCFGDB***, ***queryGlycomeDB***, and ***queryIUBMB*.**

3) query the local/offline database with carbohydrate structure to retrieve carbohydrate database ID. Output from this last step can be used with the online database commands described in steps 1 and 2.

The last step involves the use of two functions ***dbLocalConnect*** and ***dbExactStructSearch***. These functions are only enabled after setting up a local database using the MATLAB database toolbox (see section 14.5 for details). The remaining query operations (1 & 2) require only the standard GNAT installation steps described in section 14.2-14.4.

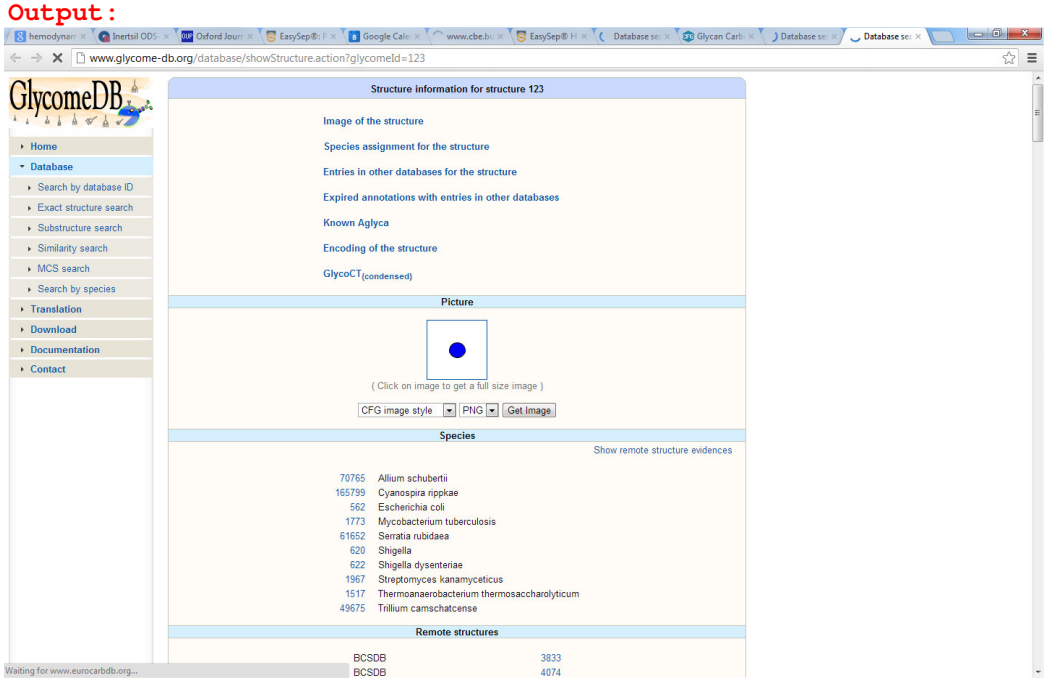
8.1 Online database query with web browser output

webGlycomicsDB displays CFG or GlycomeDB database results in default web browser. Here, the user provides the "resource name" and "dBID" as shown in example below. Currently, this command is limited to the GlycomeDB and CFG databases.

❖ **Example 8.1:**

```
webGlycomicsDB('GlycomeDB','123');
```

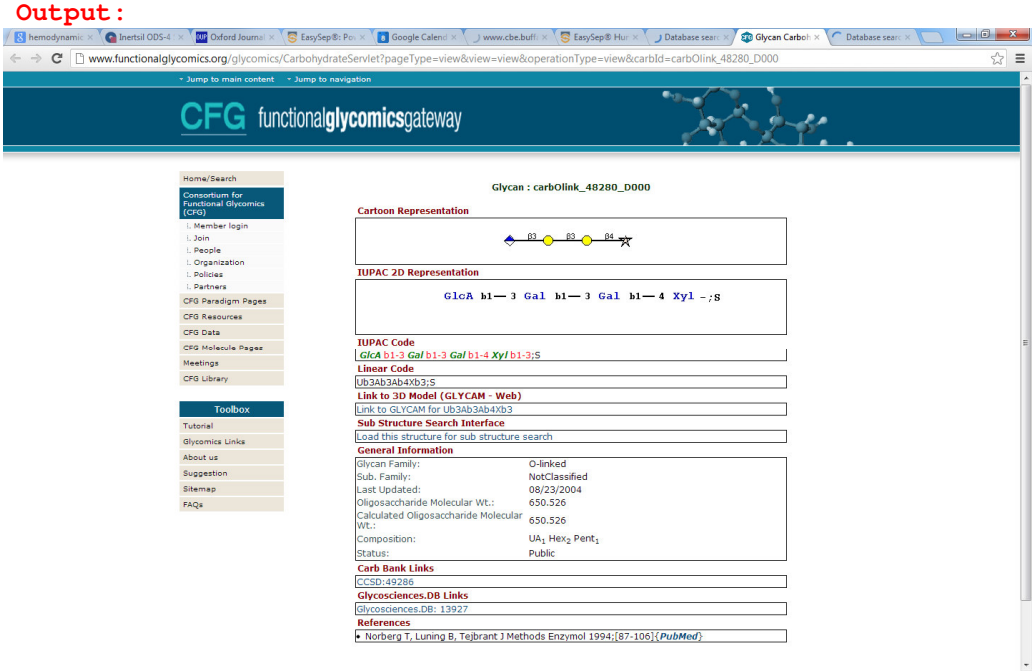
Output :



❖ **Example 8.2:**

```
webGlycomicsDB('CFG','carbOlink_48280_D000');
```

Output:



8.2 Online database query with MATLAB structure output

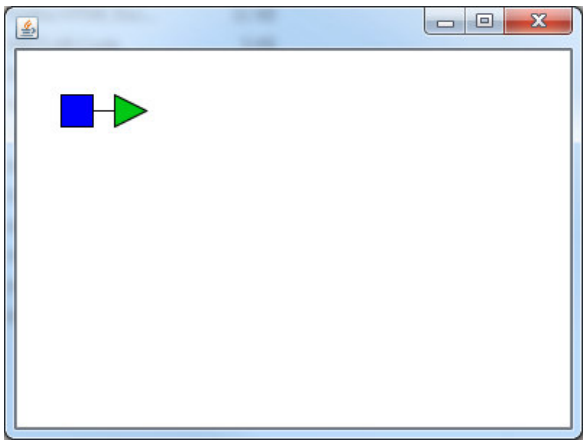
Extracting glycan information from GlycomeDB database

GNAT provides the ***queryGlycomeDB*** function to retrieve glycan structure information from the GlycomeDB database by searching for a database glycan ID (Example below). The information is stored in a MATLAB structure with three fields "species", "dbIDs" and "glycanSeq". Here, "species" reports the names of the species reported to bear the glycan structure, their corresponding NCBI ID and references in other databases such as CFG and BCDS. "dbIDs" stores the GlycomeDB IDs for the glycan structure queried. If the glycan structure was also reported in other databases, the IDs and the resources are also included. "glycanSeq" contains four subfields including a GlycanStruct object, GLYDE string, GlycoCT string, and GlycoCT-condensed string. Note that the speed of execution of this command depends on the amount of data that is being retrieved.

❖ **Example 8.3:**

```
queryGlycan = queryGlycomeDB('365'); % Query command
fn_structdisp(queryGlycan);  % To display output
glycanViewer(queryGlycan.glycanSeq.class);% To display glycan structure

Output:
      species: [1x1 struct]
        dbIDs: [1x6 struct]
    glycanSeq: [1x1 struct]
```

```
.species:
      name: 'Shigella flexneri'
    ncbiid: '623'
      refs: [1x2 struct]
.species.refs(1):
    name: 'carbbank'
      id: '12517'
.species.refs(2):
    name: 'eurocarbdb(ebi)'
      id: '418'
.dbIDs(1):
    name: 'carbbank'
      id: '12517'
.dbIDs(2):
    name: 'carbbank'
      id: '23702'
.dbIDs(3):
    name: 'eurocarbdb(ebi)'
      id: '418'
.dbIDs(4):
    name: 'glycosciences.de'
      id: '425'
.dbIDs(5):
    name: 'glycosciences.de'
      id: '27170'
.dbIDs(6):
    name: 'jcggdb'
      id: 'JCGG-STR024538'
.glycanSeq:
               glyde: [1x732 char]
             glycoct: [1x890 char]
    glycoctCondensed: [1x84 char]
               class: [1x1 GlycanStruct]
```

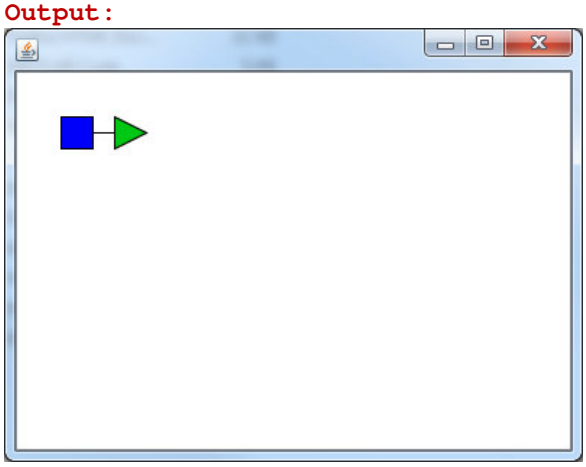


The user can also return more limited information regarding the glycan, for example only the structure, using the command below.

❖ **Example 8.4:**

```
queryGlycan = queryGlycomeDB('365','sequenceOnly',true);
glycanViewer(queryGlycan);% To display glycan structure
```

```
Output:
```



Extracting glycan information from the CFG database

GNAT provides the ***queryCFGDB*** function to retrieve glycan structure information from the CFG glycan structure database. The returned structure contains four fields: 1) summary, including the glycan family name, sub family name, last updated date, MolWeight (calculated Molecular Weight), composition, and status; 2) references; 3) IDs found in the reported databases; 4) glycanSeq including the IUPAC code and linear code.

❖ **Example 8.5:**

```
queryResult = queryCFGDB('carbOlink_48280_D000');
fn_structdisp(queryResult);
Output:
      summary: [1x1 struct]
        dbIDs: [1x1 struct]
          ref: [1x71 char]
    glycanSeq: [1x1 struct]
.summary:
      glycanFamily: 'O-linked'
         subFamily: 'NotClassified'
    lastupdatedate: '08/23/2004'
         molweight: '650.526'
       composition: 'UA1  Hex2  Pent1'
.dbIDs:
    carbBank: 'CCSD:49286'
    glycoSci: 'Glycosciences.DB: 13927'
.glycanSeq:
          iupac: 'GlcA b1-3 Gal b1-3 Gal b1-4 Xyl b1-3;S'
    linearcode: 'Ub3Ab3Ab4Xb3;S'
```

Query Enzyme information from IUBMB website

GNAT provides the ***queryIUBMBDB*** function to retrieve enzyme information from the IUBMB website. The returned structure contains five fields: 1) systematic name; 2) accepted name; 3) comments; 4) reaction and 5) other names.

❖ **Example 8.6:**

```
queryEnzResult = queryIUBMBDB([2;3;1;29])
Output:
queryEnzResult =
      comments: [1x298 char]
      systname: 'acetyl-CoA:glycine C-acetyltransferase'
    othernames: [1x202 char]
          name: 'glycine C-acetyltransferase'
           rxn: 'acetyl-CoA + glycine = CoA + L-2-amino-3-oxobutanoate'
```

8.3 Local database query with MATLAB structure output

The features described above allow querying of glycomics databases using the glycan ID. If the ID for a specific glycan is not known, *a priori*, the ***dbExactStructSearch*** function can be utilized to determine the glycan ID in GlycomeDB using the glycan structure in XML format as input.

Before running such searches, however, a PostgresSQL server containing the Glycome DB database must be created on the local computer. Detailed instructions for installation are provided at the GlycomeDB website (http://www.glycome-db.org/getDownloadPage.action?page=glycomedb) website. The GlycomeDB database dump file can also be obtained from this site. A server account with username (say 'usrGlycan') and password (say 'pwdGlycan') will be established at this stage. To communicate with the database server, two commands ***dbLocalConnect*** and ***dbExactStructSearch*** are available in GNAT. Note that the execution of these functions requires the 'MATLAB Database toolbox' (http://www.mathworks.com/products/database/).

In the example below, we illustrate how the local PostgreSQL server database is searched in order to determine the GlycomeDB database ID corresponding to the glycan 'highmannose.glycoct_xml' (glycoCT format file). Here, ***dbLocalConnect*** allows connection to the local database. ***dbExactStructSearch*** then performs the search.

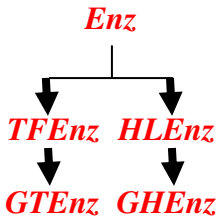
❖ **Example 8.7:**

```
M6_glycan   =  glycanMLread('highmannose.glycoct_xml');
conn        =  dbLocalConnect('GLYCOME','usrGlycan','pwdGlycan');
glycomedbID =  dbExactStructSearch(conn,M6_glycan)


Output:
glycomedbID =

     235
```

9. Glycosylation reaction network construction

9.1 The enzyme (Enz) class and related functions

Five enzyme classes are available in GNAT. These include *Enz, TFEnz, HLEnz, GTEnz* and *GHEnz*. These classes correspond to the generic enzyme, generic transferase, generic hydrolase, glycosyl tranferase, and glycosyl hydrolase respectively**.** The schematic below illustrates the hierarchy among these classes. Here, *Enz* contains general information that is pertinent to all biological enzymes while the other sub-classes contain more specific information pertinent to specific catalysis reaction types.



A detailed description of three of the enzyme classes (*Enz, GTEnz, GHEnz*) and associated functions is provided here since this is a key feature used for glycosylation reaction network (re)construction. A number of fields are provided in these classes in order to provide facilities to handle diverse enzyme activities and specificities, as described below. The user may populate as many of these fields as necessary in order to use pertinent GNAT functions, without the need to complete all the fields.

9.1.1 The enzyme class (*Enz*)

Enzymes are essential catalysts that lower the activation energy and accelerate the overall rate of biochemical reactions. The *Enz* class in GNAT is based on general guidelines established by the IUBMB. An object of the *Enz* class, thus, includes the following properties:

1. *name*: Text field containing the "recommended" or "common" name of the enzyme that often ends with the suffix *–ase*, e.g. *DNA polymerase.*
2. *systname*: This text field is the "systematic name" or more precise scientific name that is defined based on the substrate and catalyzed reactions, e.g. *dNTP:DNA dNMPtransferase.*
3. *ecno*: This 4-by-1 vector contains the "enzyme commission (EC) number", a four number classification to describe particular biochemical reactions. Here, the first number corresponds to one of six major categories, *viz.* oxidoreductases (EC 1), transferases (EC 2), hydrolases (EC 3), lyases (EC 4), isomerase (EC 5) and ligases (EC 6). Additional numbers follow and these provide information about the catalyzed reaction.
4. *othernames*: A text field that allows for alternate enzyme names.
5. *reaction*: This text field describes the enzyme-catalyzed reaction.
6. *organism*: This text field stores data on species where the enzyme is expressed.
7. *compartment*: This *compt* object specifies the cellular/extra-cellular enzyme location.

An example of how to construct an Enz object is shown in Example 9.1.

9.1.2 The glycosyltransferase (*GTEnz*) and glycosidase (*GHEnz*) class

GNAT handles two types of enzymes that are involved in the glycosylation process:

- Glycosyltransferases (GT) (EC 2.4.*x*.*y*): Enzymes that catalyze the synthesis of glycosidic linkages by the transfer of sugar residues from a donor to an acceptor substrate. These are a class of ~200 enzymes, where *x* represents the ring size of sugar residue to be transferred and *y* represents specific functional groups to be transferred. Thus, EC 2.4.1 represents hexosyltransferase, 2.4.2 corresponds to pentosyltransferase, and 2.4.99 transfers other glycosyl groups.
- Glycosidases (GH) (EC 3.2.1.*z*): Hydrolases that break glycosidic linkages. These enzymes belong to the EC 3.2.1 family and they are composed of 184 members that hydrolyze either o- or s-glycosyl compounds.

Two classes *GTEnz* and *GHEnz* are assembled to handle glycosyltransferases and glycosidases respectively. They inherited properties defined in *TfEnz* for transferase (subclass of *Enz*) and *HlEnz* for hydrolase (subclass of *Enz*) respectively. *TfEnz* and *HlEnz* catalyze differ types of enzymatic reactions:

For transferase (*TfEnz*) reactions: $donor + acceptor \rightleftharpoons donorProd + acceptorProd.$

For hydrolase (*HlEnz*) reactions: $substrate + H_20 \rightleftharpoons prod\_h + prod\_oh.$

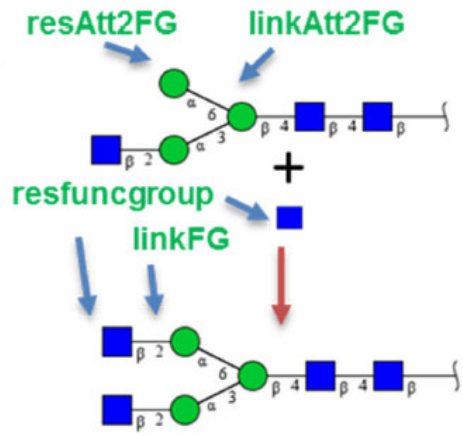
To define the specificity of enzymes, both classes augment their parent classes described above by providing 15 additional properties as follows:

The first four properties describe the **basic features** of the biochemical reactions (also see schematic below):

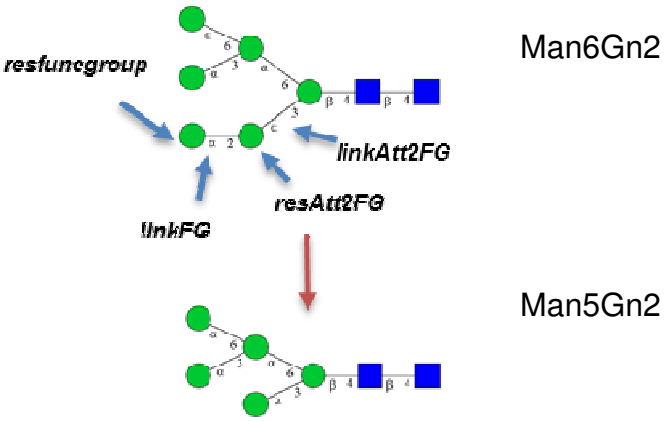
1. *resfuncgroup*: *ResidueType* object describing the precise functional group that the *GTEnz* transfers from donor to acceptor, or the *ResidueType* that *GHEnz* removes from the substrate.
2. *resAtt2FG*: In the case of *GTEnz*, this is the sugar residue (*ResidueType* object) on the acceptor where the *resfuncgroup* is added to. In the case of *GHEnz*, this is *ResidueType* object from which *resfuncgroup* is removed.
3. *linkFG*: A MATLAB structure with two fields, '*anomer*' (stereochemical info.) and '*bond*' (linkage position) that defines the glycosidic bond catalyzed by *GTEnz* including linkage position and stereochemical information. In the case of *GHEnz*, *linkFG* defines the bond of glycosidase action.
4. *linkAtt2FG*: A MATLAB data structure like *linkFG* only it describes the glycosidic bond between terminal residue (*resAtt2FG*) of the acceptor/substrate and the preceding monosaccharide. This field is only used for *GTEnz*, but is optional for *GHEnz*.

Example 9.2 provides an example of how to load these fields using the β1,2N-acetylglucosaminyltransferase (mgat2) as an example (see schematic below).

Example 9.3 provides a similar example only considering α1,2mannosidase as example (see schematic below).



Basic features of glycosyltransferases (*GTEnz*): Example of mgat2



Basic features of glycosidases (*GHEnz*): Example of α1,2 mannosidase

The next several properties describe the **minimal and maximal structural features of a substrate or product** including specific glycan structures that are excluded from the reaction.

5. *substMinStruct*: *GlycanStruct* object describing minimal glycan substrate. For example, MANI (mannosyl-oligosaccharide 1,2-alpha-mannosidase) can only act on substrate structure with minimal Man5Gn2 structure (N-glycan with 5 mannose and 2 GlcNAc residues as shown in the previous schematic).
6. *substMaxStruct*: *GlycanStruct* object describing maximal glycan substrate. For example, MANI (mannosyl-oligosaccharide 1,2-alpha-mannosidase) can only act on substrate structure with maximal structure of Man9Gn2 structure.
7. *substNAStruct*: *GlycanStruct* object describing substrate structure that enzymes do not act on.
8. *substNABranch*: *GlycanStruct* object describing substrate branch that prevent enzyme action. For example, MGAT4 cannot act on substrate with a bisecting GlcNAc structure.
9. *substNAResidue*: *ResidueType* object describing substrate's residue that prevent enzyme activity. For example, MGAT3 cannot act on substrate with a Galactose residue.

10. *prodMinStruct*: *GlycanStruct* object describing minimal glycan product.
11. *prodMaxStruct*: *GlycanStruct* object describing maximal glycan product.
12. *targetBranch*: *GlycanStruct* object describing substrate target branch where the enzyme acts. For example, MGAT3 acts on a substrate branch exclusively which is Manβ1,4GlcNAcβ1,4GlcNAc.
13. *targetNABranch*: *GlycanStruct* object describing substrate target branch where the enzyme does not act. The property allows user specify the branches that should be excluded from consideration.
    For example, GalT adds Galactose to GlcNAc residue in glycan substrate but it does not act on the GlcNAc residue in bisecting branch.
14. *isTerminalTarget*: *Logical value* indicating if the enzyme act on the terminal residue.
15. *glycanTypeSpec*: *String* indicating if enzyme acts on N-linked glycans, O-linked glycans or other types of glycoconjugates.

Additional details of each enzyme class can be seen in source code, by typing the command *help Enz* or *help Enz.subclassName* at the MATLAB command prompt, or using the doc command (*doc Enz*).

Example 9.4 provides an example where the substrate specificity of MGAT3 is defined.
Example 9.5 provides an example where the substrate specificity of the glycosidase Man I is defined.

9.2 Single step reaction inference

The network inference methods of GNAT are built upon two basic functions:

i. ***inferGlyProd*:** This function facilitates "product inference", i.e.it allows definition of reaction product when a starting substrate and enzyme are identified.

ii. ***inferGlySubstr*:** This function enables "substrate inference", i.e. it infers the initial reaction substrate, given the product and enzyme.

Below are details regarding these two functions.

9.2.1 Product inference

The ***inferGlyProd*** function follows a four-step algorithm to infer the product structure:
A. Verify the ability of a given enzyme to act on a candidate glycan-substrate;
B. Identify residues that can be enzymatically modified;
C. Create the reaction product;
D. Verify that the product follows enzyme specificity rules.

The following are the detailed steps:

A. Six sequential steps are undertaken to confirm that an enzyme can act on a glycan substrate:

1. Verified that *GTEnz.glycanTypeSpec* or *GHEnz.glycanTypeSpec* is the same as *GlycanSpecies.glycanType*. This confirms that the O-/N-glycan specificity of the enzyme corresponds to the glycan type.
2. Verify that substrate structure contains the minimal structure defined in *GTEnz.substMinStruct* or *GHEnz.substMinStruct*.
3. Verify that substrate structure is a subset of the maximum allowable structure defined in *GTEnz.substMaxStruct* or *GHEnz.substMaxStruct*.
4. Verify that substrate structure is not the same as *GTEnz.substNAStruct* or *GHEnz.substNAStruct.*
5. Verify that substrate structure does not contain *GTEnz.substNABranch* or *GHEnz.substNABranch*.
6. Verify that substrate structure does not have *GTEnz.substNAResidue* or *GHEnz.substNAResidue*.

B. Five sequential steps to identify candidate residue on substrate where the enzyme can act:

1. For *GTEnz*, this step selects potential *resAtt2FG* on substrate. This residue is a terminal monosaccharide if *GTEnz.isterminaltarget* =true and internal if *GTEnz.isterminaltarget* =false. In the case of *GHEnz*, this step identifies terminal *resfuncgroup* that are candidates for hydrolysis.
2. Linkage information is now applied to further select *resAtt2FG* for *GTEnz*, and *resfuncgroup* for *GHEnz.* These linkage criteria are specified in either *GTEnz.linkresAtt2FG* or *GHEnz.linkFG*. Additionally, if the *GHEnz.resAtt2FG* and *GHEnz.linkresAtt2FG* fields of *GHEnz* are populated, these data are also considered for refining the selection of *resfuncgroup*.
3. Further select the *resAtt2FG/ resfuncgroup* by verifying that the branch of the candidate residue is the same as either *GTEnz.targetBranch* or *GHEnz.targetBranch*.
4. Verify that the candidate residue-branch does not contain either *GTEnz.targetNABranch* or *GHEnz.targetNABranch*.
5. Verify that the candidate residue-branch contains either *GTEnz.targetBranchcontains* or *GHEnz.targetBranchcontains*.

C. The new ‘product-glycan’ is created by modifying the substrate:

1. For reaction involving *GTEnz,* the *GTEnz.resfuncgroup* is added to the acceptor glycan structure as child residue of *GTEnz.resAtt2FG*. The linkage information comes from *GTEnz.linkFG*.
2. For reaction involving *GHEnz,* the *GHEnz.resfuncgroup* is removed from the substrate. *GHEnz.linkFG* linkage is also removed.

D. Verifies that the product structure obeys two rules. If they do, then glycan product is returned. If not, an empty glycan structure is returned:

1. Verify that product structure contains the minimal structure *GTEnz.prodMinStruct* or *GHEnz.prodMinStruct*.
2. Verify that product structure is not larger than *GTEnz.prodMaxStruct* or *GHEnz.prodMaxStruct*.

Example 9.6 provides an example of ***inferGlyProd*** when MGAT2 is enzyme and Man3Gn is substrate

9.2.2 Substrate Inference

The function ***inferGlySubstr*** used for reverse reaction inference follows four steps similar to ***inferGlyProd***.

For the glycosyltransferases (*GTEnz*), this procedure infers the original substrate given a product. The four steps followed:

A. Determine if the product could have been formed by the given enzyme based on criteria defined in *GTEnz.glycanTypeSpec*, *GTEnz.prodMinStruct, and GTEnz.prodMaxStruct*.

B. Determine candidate-residues in the product that would result from the action of the *GTEnz*. Similar to above, this involves: i. determination of residues that contain *GTEnz.resfuncgroup* and *GTEnz.linkFG* that match the specific enzyme; ii. verification that the residue next to *GTEnz.resfuncgroup is GTEnz.resAtt2FG* and its linkage is *GTEnz.linkresAtt2FG*; iii. Confirming that *GTEnz.resfuncgroup* belongs to *GTEnz.targetBranch* and that it does not belong to *GTEnz.targetNABranch*.

C. Create the new glycan 'substrate' by modifying the product based on enzyme information. Specifically, a monosaccharide of the *ResidueType* of *GTEnz.resfuncgroup* is removed from product. A *GTEnz.linkFG* is also removed from this product.

D. Verify that the new glycan substrate(s) follows additional specificity rules defined in *GTEnz.substMinStruct, GTEnz.substMaxStruct, GTEnz.substNAStruct, GTEnz.substNABranch, GTEnz.substNAResidue, GTEnz.isterminaltarget* and *GTEnz.targetBranchcontains*.
If the above are satisfied, the new glycan substrate(s) are returned.

For the hydrolases (*GHEnz*), this step involves:

A. Determination that the glycan may be the product of enzymatic hydrolysis by checking the *GHEnz.glycanTypeSpec, GHEnz.prodMinStruct*, and *GHEnz.prodMaxStruct* rules similar to above.

B. Identification of candidate-residue(s) that the enzyme may have been acted upon in the glycan by: i. Checking if the candidate-residue and its linkage are the same as *GHEnz.resAtt2FG* and *GHEnz.linkresAtt2FG*; ii. verifying that *GHEnz.resAtt2FG* belongs

to *GHEnz.targetBranch* and that it is not part of *GHEnz.targetNABranch*; iii. checking that the candidate-residue branch contains *GHEnz.targetBranchcontains*.

C. Once a candidate-residue is identified in the previous step, a new glycan ('substrate') is created by addition of a new residue *GHEnz.resfuncgroup* to the candidate-residue with linkage *GHEnz.linkFG*.

D. This new glycan product is returned from this function if it meets five more specificity criteria that are defined by *GHEnz.substMinStruct, GHEnz.substMaxStruct, GHEnz.substNAStruct, GHEnz.substNABranch, GHEnz.substNAResidue* and *GHEnz.targetBranchcontains*.

Example 9.7 provides an example of ***inferGlySubstr*** when the glycosidase ManI acts to produce a Man8 structure.

9.3 Network reconstruction

GNAT utilizes the basic functionality enabled by the single step reaction inference functions, ***inferGlySubstr*** and ***InferGlyProd***, to enable glycosylation network construction. These implemented methods include:

i. ***inferGlyForwPath***: "Forward Inference" of all products and the reaction network given a starting substrate and a set of enzymes.

ii. ***inferGlyRevrPath***: "Reverse Inference" of starting substrates and network given a set of reaction products and enzymes.

iii. ***inferGlyConnPath***: "Connect pathway" by inferring intermediate glycan structures and the network when a group of glycans that may represent either starting substrates or end products are provided.

9.3.1 Forward Network Inference

inferGlyForwPath performs forward network construction when initial substrate glycans and enzymes are provided as input. This algorithm follows a four-step procedure:

1) It starts with one glycan from the list of glycan inputs. Assuming this glycan as a substrate, the program predicts the structure of glycan product(s) for a single reaction if the enzymes present in the system act on the substrate. In this step, reaction feasibility is verified and glycan products are determined based on the function ***inferGlyProd***.
2) Next, each product glycan from the above step is used as the substrate. Step 1) is then repeated to form additional new product(s) until the number of new products formed in a given iteration is zero (i.e., no more new products can be formed with this set of enzymes).
3) All newly generated glycans and reactions from the above steps are incorporated into a *Pathway* object.
4) For *m* glycans provided as input, the above steps (1)–(3) are repeated *m* times. All *GlycanSpecies* (species) and *Rxn* (reactions) that were not previously included are added to the *Pathway* object.

Example 9.8 presents an example of Forward Network Inference for the case of N-linked glycosylation.

9.3.2 Reverse Network Inference

inferGlyRevrPath supports reverse inference of the pathway given the final product(s) and the enzyme list. The algorithm is similar to the ***inferGlyForwPath*** except for the following:

1) The program iteratively predicts the substrate structure based on the enzymatic mechanism defined in the enzyme class and the product structure.
2) The program ends when no more substrates are formed. The program uses the function ***inferGlySubstr***.

Example 9.9 presents an example of Reverse Network Inference.

9.3.3 Connection Network Inference

inferGlyConnPath supports inference of the pathway by joining pairs of input glycans using enzyme-substrate specificity data provided in the *Enz* class. The detailed algorithm follows:

1. Choose a pair of input glycans. Here:
   a) Determine the monosaccharides that are different in the two entities, and thus identify specific *ResidueTypes* and monosaccharide numbers that account for the difference between the two molecules.
   b) Determine if the system has enzymes (*Enz*) that are capable of adding or deleting the *ResidueTypes* in order to connect these two molecules. At this step, linkage information or detailed substrate specificity is not considered. Depending on whether the utilized enzymes are transferases or hydrolases, designate one of the glycans as the 'initial substrate' and the other as the 'final product'.
2. Infer the pathways that link the 'initial substrate- final product' glycan pair. Here:
   a) Perform detailed pathway analysis to infer the 'connecting glycan(s)'. To this end, ***inferGlySubstr*** is used to determine the 'connecting glycans' by stepwise deleting/adding monosaccharides from the 'final product' until the 'initial substrate' is achieved. This step utilized the detailed enzyme specificity rules described in the *Enz* class as discussed in section 9.2.2 above.
   b) If the 'initial substrate' is determined, all reactions (*Rxn*) determined above and intermediate glycans (*GlycanSpecies*) are added to the *Pathway* object, else there is no addition.
   c) The link between an 'initial substrate-final product' pair may not be unique and thus, steps a)-b) are repeated iteratively to identify all possible connecting routes (*Rxn* and *GlycanSpecies*). Non-redundant information is added to *Pathway.*
3. For *m* input glycans, steps 1)–2) are repeated m×(m-1)/2 times to account for all possible input-pair combinations. The consolidated *Pathway* object is thus constructed.

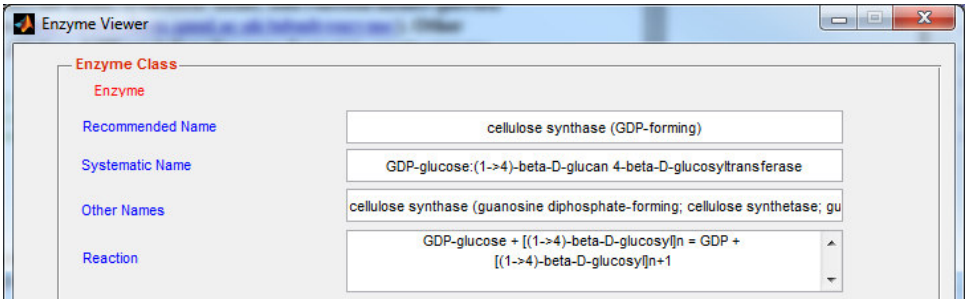
Example 9.10 presents an example of Connection Network Inference.

9.4 Usage examples

Example 9.1: The function *Enz* reads the EC number from a double vector. It uses this information to query the IUBMB website (http://www.chem.qmul.ac.uk/iubmb/enzyme/) in order to populate relevant fields of the enzyme object including enzyme name, systematic name, and reaction names (see below). This information is same for all enzyme associated classes: *Enz*, *HLEnz*, *TFEnz*, *GTEnz* and *GHEnz*.

```
synthase = Enz([2;4;1;29]);
%The EC of cellulose synthase is 2.4.1.29
enzViewer(synthase)
```

Output :

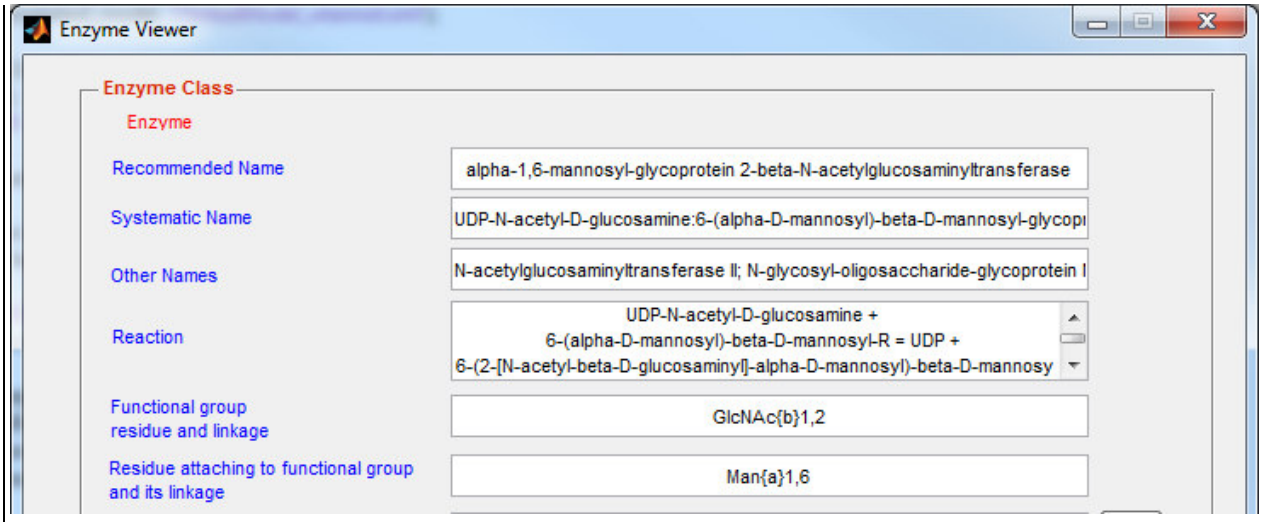


Example 9.2: MGAT2 whose systematic name is alpha-1,6-mannosyl-glycoprotein 2-beta-N- acetylglucosaminyltransferase is a glycosyltransferase. It adds GlcNAc to α1,6 linked mannose. A β1,2GlcNAc to mannose is catalyzed by mgat2.

Here, *resfuncgroup* is GlcNAc, *linkFG* is β1,2, *resAtt2FG* is mannose, *linkAtt2FG* is α1,6 . The commands for describing these features in GNAT are as below:

```
residueMap          = load('residueTypes.mat');
manResType          = residueMap.allresidues('Man');
mgat2               = GTEnz([2;4;1;143]);
mgat2.resfuncgroup  = residueMap.allresidues('GlcNAc');
manResType          = residueMap.allresidues('Man');
manBond             = GlycanBond('6','1');
mgat2.resAtt2FG     = manResType;
mgat2.linkresAtt2FG = struct('bond', manBond,'anomer','a');
glcnacbond          = GlycanBond('2','1');
mgat2.linkFG        = struct('anomer','b','bond',glcnacbond);
enzViewer(mgat2)

% Note: residueMap contains ResidueType information for 78 commonly used monomer units
% in the field of Glycobiology and their derivatives, like Glc, HexNAc, Fru, Xyl etc.
% To obtain a list of the entities covered, type `residueMap.allresidues.keys'at the
% MATLAB command line.
```

Output :

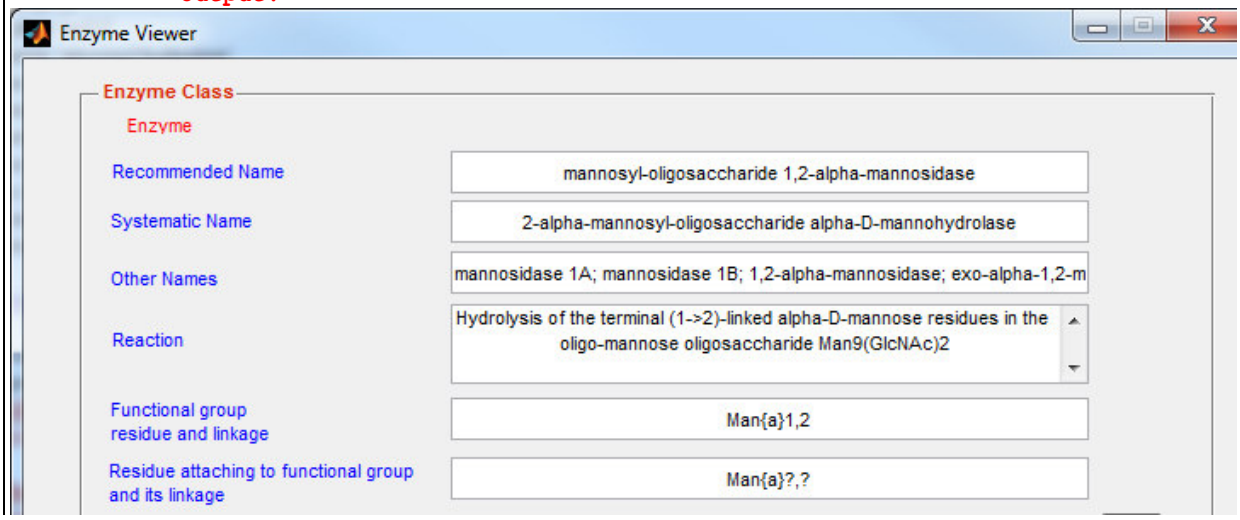


Example 9.3: Man I is short name of mannosyl-oligosaccharide 1,2-alpha-mannosidase. Its ECNO is 3.2.1.78. As shown in the diagram below, it removes α1,2mannose from high mannose structure containing at least five mannose residues.

Based on this information, *resfuncgroup* is mannose, *linkFG* is α1,2, *resAtt2FG* is mannose, *linkAtt2FG* is α?,? (non-specific). The commands are as follows:

```
residueMap          = load('residueTypes.mat');
manResType          =residueMap.allresidues('Man');
mani                = GHEnz([3;2;1;113]);
mani.resfuncgroup   = manResType;
manBond             = GlycanBond('2','1');
mani.linkFG         = struct('anomer', 'a', 'bond', manBond);
mani.resAtt2FG      = manResType;
manunknownBond      = GlycanBond('?','?');
mani.linkresAtt2FG  = struct('anomer', 'a', 'bond', manunknownBond);
enzViewer(mani);
```

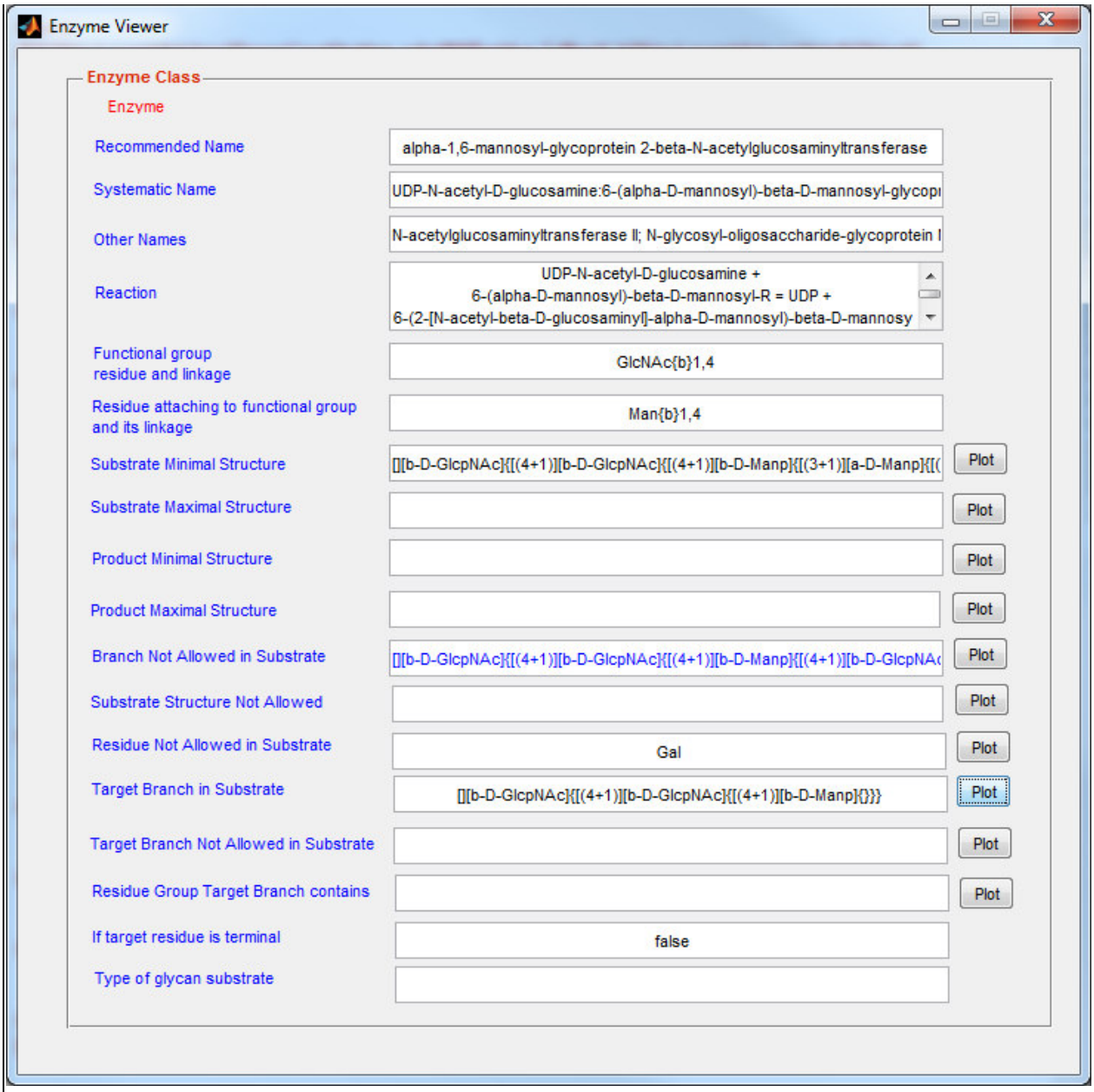
Output :



❖ **Example 9.4: Substrate specificities for MGAT3 are defined as follows.**

```
mgat3                = GTEnz([2;4;1;143]);
mgat3.resfuncgroup   = residueMap.allresidues('GlcNAc');
manResType           = residueMap.allresidues('Man');
manBond              = GlycanBond('4','1');
mgat3.resAtt2FG      = manResType;
mgat3.linkresAtt2FG = struct('bond', manBond,'anomer','b');
glcnacbond           = GlycanBond('4','1');
mgat3.linkFG         = struct('anomer','b','bond',glcnacbond);
m3gn                 = glycanMLread('m3gn.glycoct_xml');
mgat3.substMinStruct= m3gn;
mgat3.substNABranch = glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');
mgat3.targetBranch   = glycanMLread('mgat3targetbranch.glycoct_xml');
mgat3.substNAResidue= residueMap.allresidues('Gal');
enzViewer(mgat3)
```

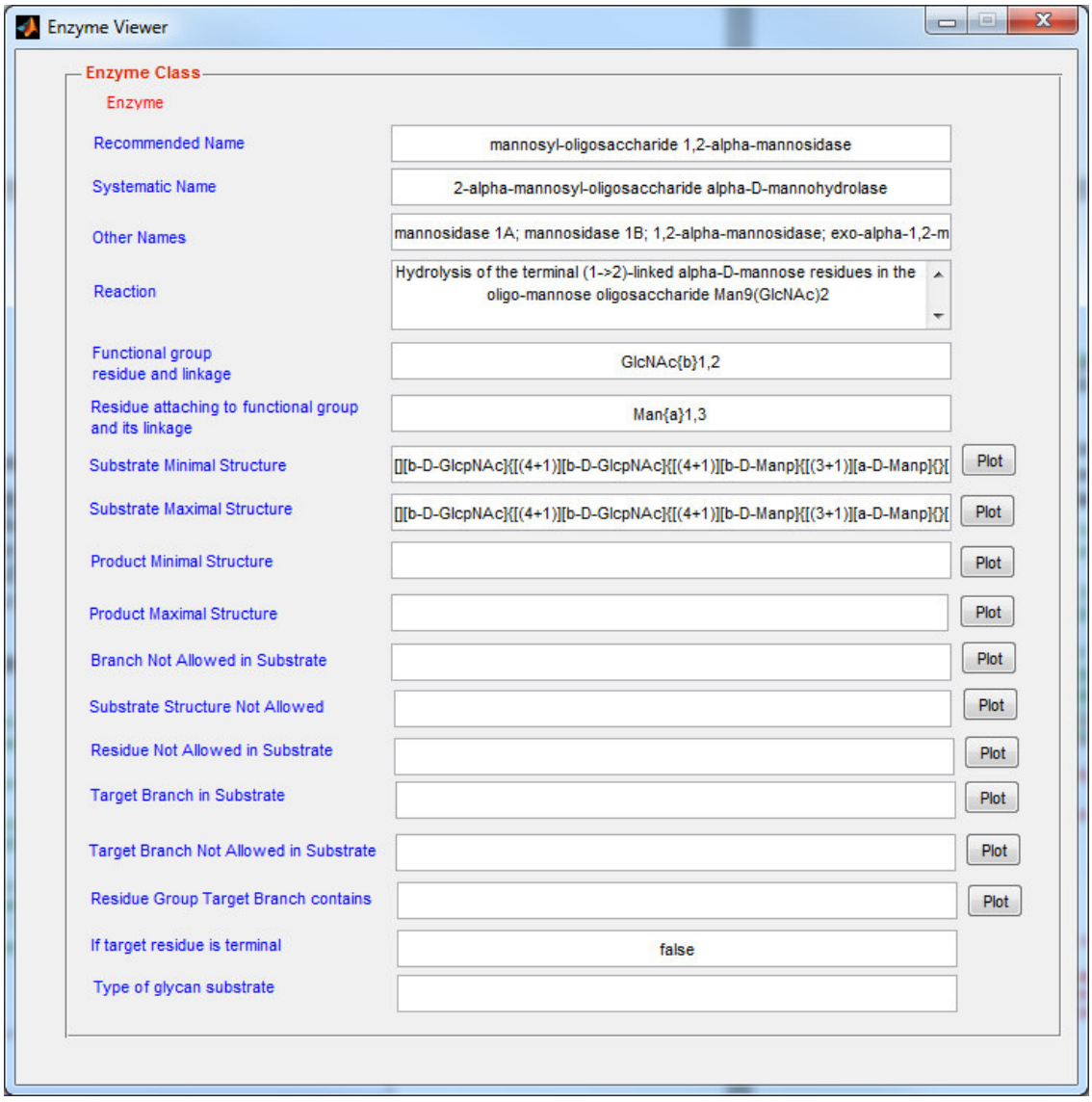
Note: Substrate information in the above example is read using .xml format files. In the enzViewer window glycan structures are depicted in LINUCS format. Clicking the plot button next to indicated glycans results in a graphical representation of the glycan using glycanViewer.



❖ **Example 9.5: Substrate specificities for glycosidases ManI are defined in the "*substMinStruct*" and "*substMaxStruct*" properties.**

```
residueMap          = load('residueTypes.mat');
manResType          = residueMap.allresidues('Man');
mani                = GHEnz([3;2;1;113]);
mani.resfuncgroup   = manResType;
manBond             = GlycanBond('2','1');
mani.linkFG         = struct('anomer', 'a', 'bond', manBond);
mani.substMinStruct = glycanMLread('M6.glycoct_xml');
% mannosidase I can only work on M6-M9 glycans
mani.substMaxStruct = glycanMLread('M9.glycoct_xml');
enzViewer(mani);
```

Output:



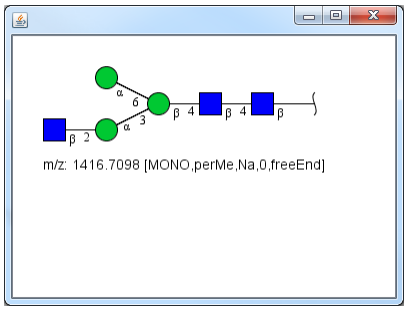
Example 9.6: *inferGlyProd* infer the product glycan if MGAT2 is the enzyme and m3gn (Man3Gn structure) is the substrate

```
residueMap            = load('residueTypes.mat');
mgat2                 = GTEnz([2;4;1;143]);
mgat2.resfuncgroup    = residueMap.allresidues('GlcNAc');
manResType            = residueMap.allresidues('Man');
manBond               = GlycanBond('6','1');
mgat2.resAtt2FG       = manResType;
glcnacbond            = GlycanBond('2','1');
mgat2.linkFG          = struct('anomer','b','bond',glcnacbond);
mgat2.linkresAtt2FG   = struct('bond', manBond,'anomer','a');
m3gn                  = glycanMLread('m3gn.glycoct_xml');
mgat2.substMinStruct  = m3gn;
mgat2.substMaxStruct  = m3gn;
m3gnspecies = GlycanSpecies(glycanMLread('m3gn.glycoct_xml'));
options         = displayset('showmass',true,'showLinkage',true,...
                  'showRedEnd',true);
```

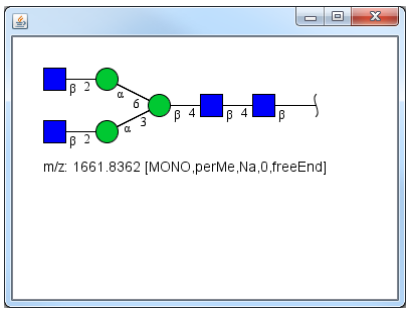
```
    [nprods,m3gngnspecies] = inferGlyProd(m3gnspecies,mgat2);
    disp('The structure of substrate will be shown below:')
    glycanViewer(m3gnspecies.glycanStruct,options);
    disp('All possible products will be shown below');
    for i=1: nprods
       glycanViewer(m3gngnspecies.get(i).glycanStruct,options);
    end
```

Outputs:

```
The structure of substrate will be shown below:
```




```
All possible products will be shown below:
```

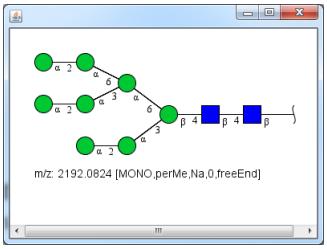


Example 9.7: *inferGlySubstr* infers the substrate glycan when the enzyme is mani and the product is Man8 high mannose structure.

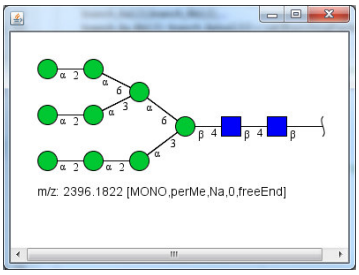
```
residueMap          = load('residueTypes.mat');
manResType          = residueMap.allresidues('Man');
mani                = GHEnz([3;2;1;113]);
mani.resfuncgroup   = manResType;
manBond             = GlycanBond('2','1');
mani.linkFG         = struct('anomer', 'a', 'bond', manBond);
mani.substMinStruct =  glycanMLread('M6.glycoct_xml');
% mannosidase I cannot work    on M5 glycan
mani.substMaxStruct = glycanMLread('M9.glycoct_xml');
m8species           = GlycanSpecies(glycanMLread('M8.glycoct_xml'));
options             =  displayset('showmass',true,'showLinkage',true,...
                'showRedEnd',true);
[nprods,substrspecies] = inferGlySubstr(m8species,mani);
        disp('Glycan structure of the product will be shown as below:')
glycanViewer(m8species.glycanStruct,options);
        disp('Glycan structures of substrates will be shown as below:')
for i=1: nprods
   glycanViewer(substrspecies.get(i).glycanStruct,options);
end
```

Outputs:

```
Glycan structure of the product will be shown as below:
```



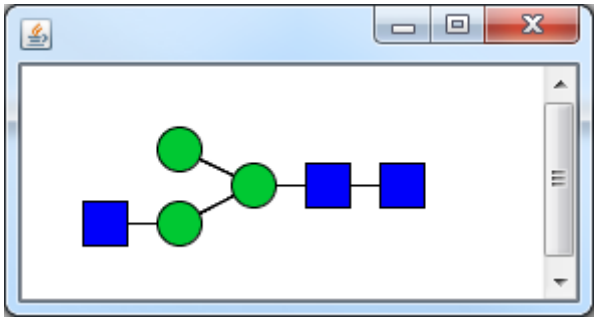

```
Glycan structures of substrates will be shown as below.
```



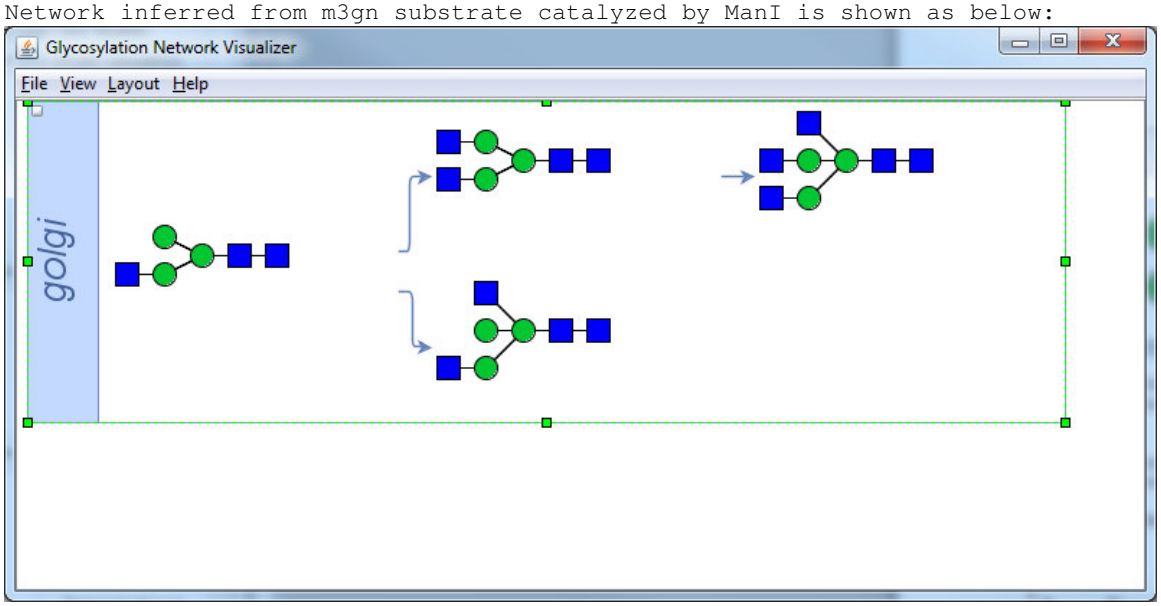
Example 9.8: *inferGlyForwPath* reconstructs the network when initial substrate glycans and enzymes are provided as inputs. In this example starting with the M3Gn (GlcNAcMan3GlcNAc2) substrate and MGAT2 & 3 enzyme, the pathways resulting in the formation of branched N-Glycans is inferred.

```
clear;

% % The following steps retrieve the enzyme MGAT2&3 from a local enzyme database. These
two enzymes add GlcNAc to different mannose residues in the substrate.

enzdbmatfilename   = 'glyenzDB.mat';
enzdb = enzdbmatLoad(enzdbmatfilename);
mgat2 = enzdb('mgat2');
mgat3 = enzdb('mgat3');


% All enzymes are placed in a CellArrayList object called enzArray
enzArray = CellArrayList;
enzArray.add(mgat2);
enzArray.add(mgat3);

% Starting substrate is the m3gn glycan (GlcNAcMan3GlcNAc2). This is read from .xml file
% and placed in another CellArrayList object
m3gnspecies    = GlycanSpecies(glycanMLread('m3gn.glycoct_xml')) ;
substrateArray = CellArrayList;
substrateArray.add(m3gnspecies);

% These steps run the inferGlyForwPath function and display results
fprintf(1,'Initial substrate structure is shown as below:');
glycanViewer(m3gnspecies.glycanStruct);
[isPath,nglycanpath] = inferGlyForwPath(substrateArray, enzArray);
fprintf(1,'Network inferred from M3Gn substrate catalyzed by MGAT2 and MGAT3 is shown as
below:\n');
glycanPathViewer(nglycanpath);

Output:
```

```
   Initial substrate structure is shown as below:
```



```
round:1
number of total species in the pathway: 3
number of total reactions in the pathway: 2

round:2
number of total species in the pathway: 4
number of total reactions in the pathway: 3

Network inferred from m3gn substrate catalyzed by ManI is shown as below:
```



❖ Example 9.9: *inferGlyRevrPath* infers the glycosylation network given a final N-glycan product and a list of five enzymes

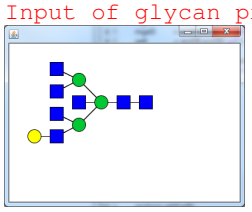
```
clear;

% The following steps specify the enzymes being used. These include mgat2, mgat3
% mgat4, mgat5 and GalT. All enzymes are placed in CellArrayList object.

residueMap                      = load('residueTypes.mat');
% define mgat2 enzyme
mgat2                           = GTEnz([2;4;1;143]);
mgat2.resfuncgroup              = residueMap.allresidues('GlcNAc');
manResType                      = residueMap.allresidues('Man');
manBond                         = GlycanBond('6','1');
mgat2.resAtt2FG                 = manResType;
mgat2.linkresAtt2FG             = struct('bond', manBond,'anomer','a');
glcnacbond                      = GlycanBond('2','1');
mgat2.linkFG                    = struct('anomer','b','bond',glcnacbond);
```

```
m3gn                              = glycanMLread('m3gn.glycoct_xml');
mgat2.isTerminalTarget            = true;
mgat2.substMinStruct              = m3gn;
mgat2.targetBranch                = glycanMLread('mgat2actingbranch.glycoct_xml');
mgat2.substNABranch               = CellArrayList;
mgat2.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat2.substNABranch.add(glycanMLread('mgat2substrateNAbranch.glycoct_xml'));
mgat2.substNAResidue= residueMap.allresidues('Gal');

% define mgat3 enzyme
mgat3                             = GTEnz([2;4;1;143]);
mgat3.resfuncgroup                = residueMap.allresidues('GlcNAc');
glcnacbond                        = GlycanBond('4','1');
mgat3.linkFG                      = struct('anomer','b','bond',glcnacbond);
manResType                        = residueMap.allresidues('Man');
manBond                           = GlycanBond('4','1');
mgat3.resAtt2FG                   =  manResType;
mgat3.linkresAtt2FG               = struct('bond', manBond,'anomer','b');
m3gn                              = glycanMLread('m3gn.glycoct_xml');
mgat3.substMinStruct              = m3gn;
mgat3.substNABranch               = glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');
mgat3.targetBranch                = glycanMLread('mgat3targetbranch.glycoct_xml');
mgat3.substNAResidue              = residueMap.allresidues('Gal');

% define mgat4 enzyme
mgat4                             = GTEnz([2;4;1;14]);
mgat4.resfuncgroup                = residueMap.allresidues('GlcNAc');
manResType                        = residueMap.allresidues('Man');
manBond                           = GlycanBond('3','1');
glcnacbond                        = GlycanBond('4','1');
mgat4.linkFG                      = struct('anomer','b','bond',glcnacbond);
mgat4.resAtt2FG                   = manResType;
mgat4.linkresAtt2FG               = struct('bond', manBond,'anomer','a');

mgat4.substMinStruct              = m3gn;
mgat4.targetBranch                =  glycanMLread('mgat4targetbranch.glycoct_xml');
mgat4.substNABranch               = CellArrayList;
mgat4.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat4.substNABranch.add(glycanMLread('mgat4subsNAbranch.glycoct_xml'));
mgat4.substNAResidue              = residueMap.allresidues('Gal');

% define mgat5 enzyme
mgat5                             = GTEnz([2;4;1;155]);
mgat5.resfuncgroup                = residueMap.allresidues('GlcNAc');
glcnacbond                        = GlycanBond('6','1');
mgat5.linkFG                      = struct('anomer','b','bond',glcnacbond);
manResType                        = residueMap.allresidues('Man');
manBond                           = GlycanBond('6','1');
mgat5.resAtt2FG                   = manResType;
mgat5.linkresAtt2FG               = struct('bond', manBond,'anomer','a');
m3gn                              = glycanMLread('m3gn.glycoct_xml');
%mgat5.substMinStruct             = m3gn;
mgat5.targetBranch                = glycanMLread('mgat5targetBranch.glycoct_xml');
mgat5.substMinStruct              = glycanMLread('mgat5substMinStruct.glycoct_xml');
mgat5.substNABranch               = CellArrayList;
mgat5.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat5.substNABranch.add(glycanMLread('mgat5substrateNAbranch.glycoct_xml'));

% define GalT enzyme
galt                              = GTEnz([2;4;1;38]);
galt.isTerminalTarget             = true;
galt.resfuncgroup                 = residueMap.allresidues('Gal');
galtbond                          = GlycanBond('4','1');
```

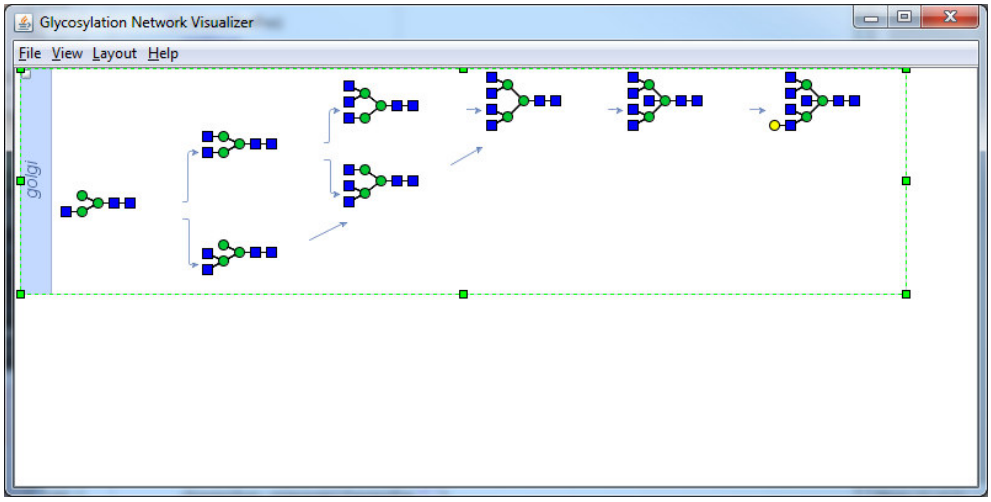
```
galt.linkFG                          = struct('anomer','b','bond',galtbond);

glcnacResType                        = residueMap.allresidues('GlcNAc');
glcnacBond                           = GlycanBond('?','1');
galt.resAtt2FG                       = glcnacResType;
galt.linkresAtt2FG                   = struct('bond', glcnacBond,'anomer','b');
galt.targetNABranch=glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');

enzArray = CellArrayList;
enzArray.add(mgat2);
enzArray.add(mgat3);
enzArray.add(mgat4);
enzArray.add(mgat5);
enzArray.add(galt);

% Define the end product (‘tetreprimeglycan’) and place this in CellArrayList
tetraprimglycan     = ...
GlycanSpecies(glycanMLread('tetriprime_gal_nlinkedglycan.glycoct_xml')) ;
prodArray = CellArrayList;
prodArray.add(tetraprimglycan);


% Perform reaction inference and display starting product and inferred network
[isPath,nlinkedpath]=inferGlyRevrPath(prodArray, enzArray);
fprintf(1,'Input of glycan product structure is \n');
glycanViewer(tetraprimglycan.glycanStruct);
fprintf(1,'Inferred network is shown below:\n');
glycanPathViewer(nlinkedpath);
```

Output:

```
Input of glycan product structure is
```



```
round:1
number of species added:2
number of reactions added:1
round:2
number of species added:3
number of reactions added:2
round:3
number of species added:5
number of reactions added:4
round:4
number of species added:7
number of reactions added:7
round:5
number of species added:8
number of reactions added:9
Inferred network is shown below:
```



❖ **Example 9.10: *inferGlyConnPath* builds the network to connect all the input glycans.**

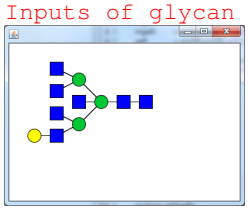
```
clear;
residueMap                  = load('residueTypes.mat');

% The following steps specify the enzymes being used. These include mgat2, mgat3
% mgat4, mgat5 and GalT. All enzymes are placed in CellArrayList object.

% define mgat2 enzyme
mgat2                       = GTEnz([2;4;1;143]);
mgat2.resfuncgroup          = residueMap.allresidues('GlcNAc');
manResType                  = residueMap.allresidues('Man');
manBond                     = GlycanBond('6','1');
mgat2.resAtt2FG             = manResType;
mgat2.linkresAtt2FG         = struct('bond', manBond,'anomer','a');
glcnacbond                  = GlycanBond('2','1');
mgat2.linkFG                = struct('anomer','b','bond',glcnacbond);
m3gn                        = glycanMLread('m3gn.glycoct_xml');
mgat2.isTerminalTarget      = true;
mgat2.substMinStruct        = m3gn;
mgat2.targetBranch          = glycanMLread('mgat2actingbranch.glycoct_xml');
mgat2.substNABranch         = CellArrayList;
mgat2.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat2.substNABranch.add(glycanMLread('mgat2substrateNAbranch.glycoct_xml'));
%mgat2.substNAResidue= residueMap.allresidues('Gal');

% define mgat3 enzyme
mgat3                       = GTEnz([2;4;1;143]);
mgat3.resfuncgroup          = residueMap.allresidues('GlcNAc');
glcnacbond                  = GlycanBond('4','1');
mgat3.linkFG                = struct('anomer','b','bond',glcnacbond);
manResType                  = residueMap.allresidues('Man');
manBond                     = GlycanBond('4','1');
mgat3.resAtt2FG             =  manResType;
mgat3.linkresAtt2FG         = struct('bond', manBond,'anomer','b');
m3gn                        = glycanMLread('m3gn.glycoct_xml');
mgat3.substMinStruct        = m3gn;
mgat3.substNABranch         = glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');
mgat3.targetBranch          = glycanMLread('mgat3targetbranch.glycoct_xml');
mgat3.substNAResidue        = residueMap.allresidues('Gal');
```

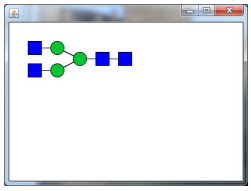
```
% define mgat4 enzyme
mgat4                       = GTEnz([2;4;1;14]);
mgat4.resfuncgroup          = residueMap.allresidues('GlcNAc');
manResType                  = residueMap.allresidues('Man');
manBond                     = GlycanBond('3','1');
glcnacbond                  = GlycanBond('4','1');
mgat4.linkFG                = struct('anomer','b','bond',glcnacbond);
mgat4.resAtt2FG             = manResType;
mgat4.linkresAtt2FG         = struct('bond', manBond,'anomer','a');

mgat4.substMinStruct        = m3gn;
mgat4.targetBranch          =  glycanMLread('mgat4targetbranch.glycoct_xml');
mgat4.substNABranch         = CellArrayList;
mgat4.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat4.substNABranch.add(glycanMLread('mgat4subsNAbranch.glycoct_xml'));
mgat4.substNAResidue = residueMap.allresidues('Gal');

%% define mgat5 enzyme
mgat5                       = GTEnz([2;4;1;155]);
mgat5.resfuncgroup          = residueMap.allresidues('GlcNAc');
glcnacbond                  = GlycanBond('6','1');
mgat5.linkFG                = struct('anomer','b','bond',glcnacbond);
manResType                  = residueMap.allresidues('Man');
manBond                     = GlycanBond('6','1');
mgat5.resAtt2FG             = manResType;
mgat5.linkresAtt2FG         = struct('bond', manBond,'anomer','a');
m3gn                        = glycanMLread('m3gn.glycoct_xml');
%mgat5.substMinStruct       = m3gn;
mgat5.targetBranch          = glycanMLread('mgat5targetBranch.glycoct_xml');
mgat5.substMinStruct        = glycanMLread('mgat5substMinStruct.glycoct_xml');
mgat5.substNABranch         = CellArrayList;
mgat5.substNABranch.add(glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat5.substNABranch.add(glycanMLread('mgat5substrateNAbranch.glycoct_xml'));

%% define GalT enzyme
galt                        = GTEnz([2;4;1;38]);
galt.isTerminalTarget       = true;
galt.resfuncgroup           = residueMap.allresidues('Gal');
galtbond                    = GlycanBond('4','1');
galt.linkFG                 = struct('anomer','b','bond',galtbond);

glcnacResType               = residueMap.allresidues('GlcNAc');
glcnacBond                  = GlycanBond('?','1');
galt.resAtt2FG              = glcnacResType;
galt.linkresAtt2FG          = struct('bond', glcnacBond,'anomer','b');
galt.targetNABranch=glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');

enzArray                    = CellArrayList;
enzArray.add(mgat2);
enzArray.add(mgat3);
enzArray.add(mgat4);
enzArray.add(mgat5);
enzArray.add(galt);

% The connection between two glycans is inferred in this example. These are
% read from .xml files below and added to a CellArrayList

tetreprimglycan             = ...
GlycanSpecies(glycanMLread('tetriprime_gal_nlinkedglycan.glycoct_xml')) ;
m3gngn                      = GlycanSpecies(glycanMLread('m3gngn.glycoct_xml')) ;
glycanArray                 = CellArrayList;
glycanArray.add(tetreprimglycan);
glycanArray.add(m3gngn);  % an Array containing all input glycans
```

```
% Network inference is performed. The input glycans and inferred network
% are then displayed

fprintf(1,'Inputs of glycan product structures are: \n');
glycanViewer(tetreprimglycan.glycanStruct);
glycanViewer(m3gngn.glycanStruct);
[isPath,nlinkedpath]=inferGlyConnPath(glycanArray, enzArray);
fprintf(1,'Inferred network is shown below:\n');
glycanPathViewer(nlinkedpath);
```

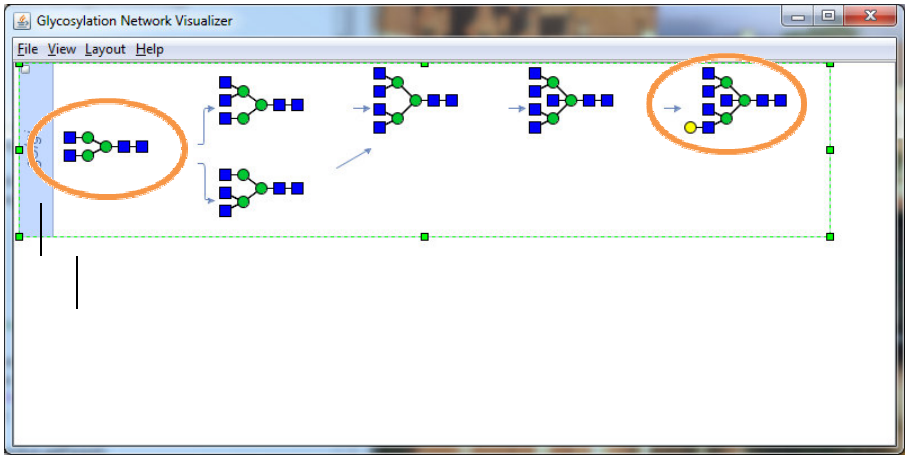
Output:

Inputs of glycan product structures are:





```
the round: 1
number of total species in the pathway: 8
number of total reactions in the pathway: 9
Inferred network is shown below:
```



10. Graph operations

Once a glycosylation reaction network is constructed using methods outlined in the previous chapter, additional methods where the network is perturbed by either adding or deleting selected reactions or groups of species/reactions is necessary for the purpose of network analysis and model reduction (Liu, Marathe et al. 2008). GNAT provides three types of high-level graph operation algorithms for such work. These include:

- ***pathfinding***: This algorithm finds the reaction path between two species in a network
- ***detectIsolatedSpecies*** and ***removeIsolatedSpecies*** : This is applied to detect and remove species that are not connected to any other species in a reaction network.
- ***subnetgenbynumdel***, ***subnetgenbyspecdel*** and ***subnetgenbyspeckeep***: These functions enable the generation of subset reaction networks where selected components of the larger reaction are isolated for independent analysis.

The *Pathway* class is composed of a variety of species that are connected by reactions. From the perspective of Graph Theory, the species can be considered to be vertices and the reactions can be thought to represent the edges. Following this logic, a variety of algorithms in graph theory may be applied for reaction network analysis for the addition or removal of the species, finding pathways between two nodes, etc. The applications discussed here draw analogy from applications in computer engineering where highly interconnected system such as the internet, airline connections, city road map, etc. are analyzed using similar methods.

10.1 Path finding

The ***pathfinding*** function identifies all possible biosynthetic routes that link two species in a glycosylation reaction network. One of these is defined by the user to be the 'starting species' and the other is the 'target glycan'. The 'Depth-first search (DFS)' algorithm is used in this command to find the linkage between the two inputs. This involves the following steps:

1. The first child node of the 'starting species' is identified and checks are performed to determine if the child is the 'target glycan'.
2. If not, the algorithm continues to examine the children of this first child, and then proceeds stepwise deeper and deeper until it finds its target node or a node that does not have children.
3. In case it finds its target, the identified pathway is stored.
4. In case it reaches the node that has no successors, it returns back to the most preceding species and searches additional unvisited routes.
5. The process proceeds recursively, until all possible connecting links from the 'starting species' have been examined.

Example 10.1 provides an example that applies this ***pathfinding*** function.

10.2 Detection and removal of isolated species

GNAT provides a command ***detectIsolatedSpecies*** to search for isolated species in the network. In this command, the algorithm enumerates all the species defined in the network. Then, it checks if the species is involved in any reaction in the network. Removal of such species is enabled using a command ***removeIsolatedSpecies***.

Example 10.2 provides an example of how to create, detect and remove isolated species from a reaction network.

10.3 Subset Network Generation

The subset network is a network resulting from the removal of one or more species and their associated reactions from the pathway. Three commands ***subnetgenbynumdel***, ***subnetgenbyspecdel*** and ***subnetgenbyspeckeep*** can be used to create a number of subset networks from a parent pathway. These three commands allow generation of subset network:

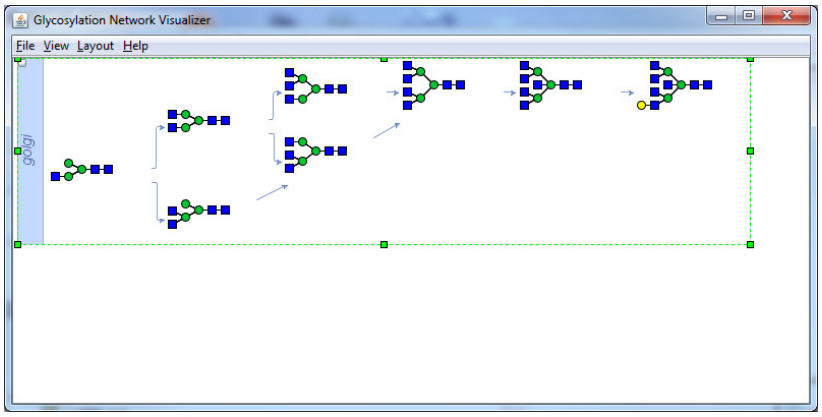
- *By removing the user-specified number of species*. This will result in a group of product reaction networks with the specified number of species deleted in each of them.
- *By removing the user-specified species*. Here, the user specifies which species is to be deleted. All other species are retained.
- *By keeping the user-specified species*. Here, the user specifies which species are to be saved. All others are deleted.

Example 10.3 presents an example of subset network generation.

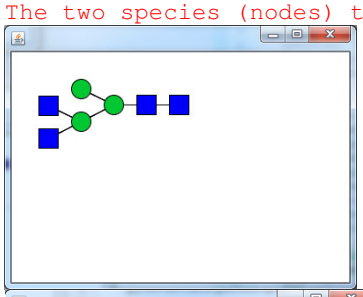
10.4 Usage examples

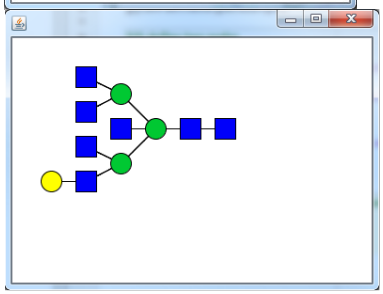
❖ Example 10.1: pathfinding finds the path in a defined network

```
% Load original example pathway
pathexample = Pathway.loadmat('nlinkedpath.mat');
disp('An example pathway is shown below:');
glycanPathViewer(pathexample);

% Define two nodes or species between which path has to be found.
% Designate one to be the 'starting species' or reactant and other to be the
% 'target glycan' or product
tetreprimglycan  = GlycanSpecies(...
  glycanMLread('tetriprime_gal_nlinkedglycan.glycoct_xml')) ;
glycanViewer(tetreprimglycan.glycanStruct)

m3gngnlowerglycan = GlycanSpecies(...
  glycanMLread('m3gngnlowerglycan.glycoct_xml'));
glycanViewer(m3gngnlowerglycan.glycanStruct)

pair.react=m3gngnlowerglycan;
pair.prod= tetreprimglycan;

% find path between the two species in the Pathway and output solution
pathfound=pathfinding(pathexample,pair);
glycanPathViewer(pathfound);

Output:
The original pathway is shown below:
```

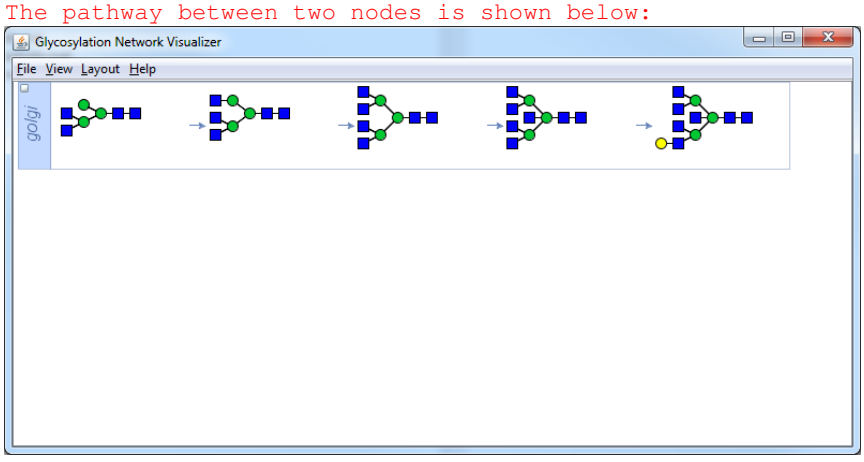



```
The two species (nodes) to search in the pathway are
```





```
The pathway between two nodes is shown below:
```



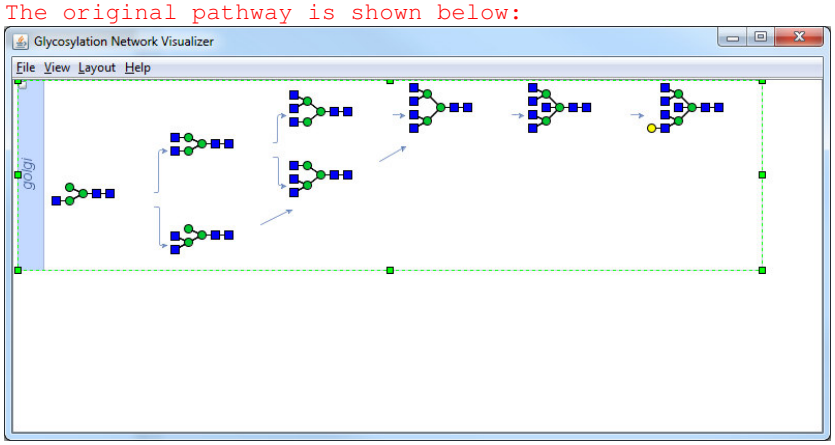
❖ Example 10.2: removeIsolatedSpecies removes "isolated species" from the network.

```
% load original example path and display it
pathexample = Pathway.loadmat('nlinkedpath.mat');
disp('An example pathway is shown below:');
glycanPathViewer(pathexample);

% set a species to be "isolated"
```

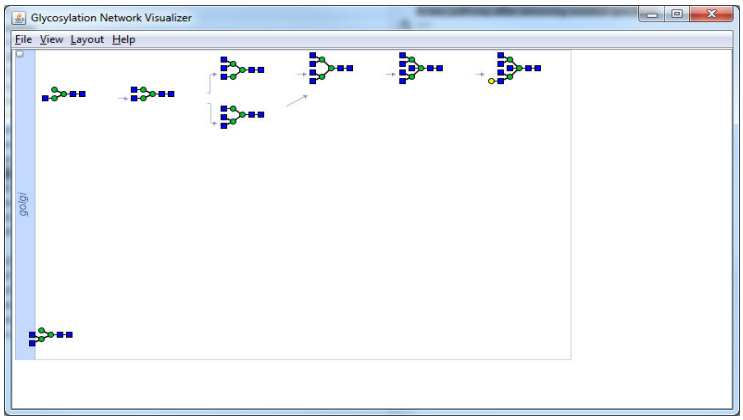
```
m3gngnlowerglycanStruct =  glycanMLread('m3gngnlowerglycan.glycoct_xml');
pathexample.setSpeciesIsolated(m3gngnlowerglycanStruct);
disp('An example pathway is shown below:');
glycanPathViewer(pathexample);

% find path between two species in the network
% detectIsolatedSpecies(pathexample);
% glycanViewer(ans.glycanStruct)  -- These two lines can be used to view the isolated species
removeIsolatedSpecies(pathexample);
disp('A new pathway after removing isolated species is:');
glycanPathViewer(pathexample);
```

Output:

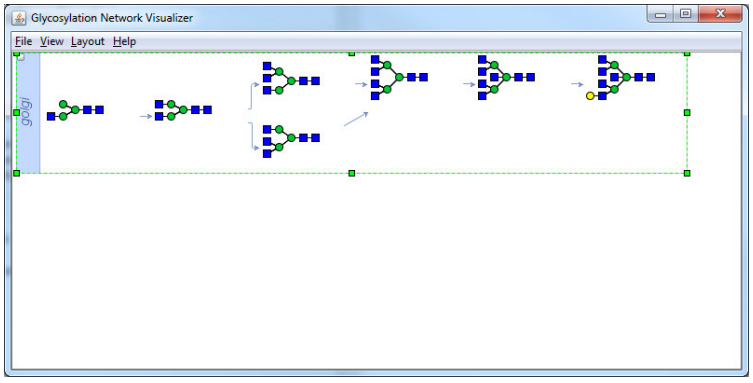
The original pathway is shown below:



After setting a species "isolated", the pathway is as below:



A new pathway after removing isolated species is:

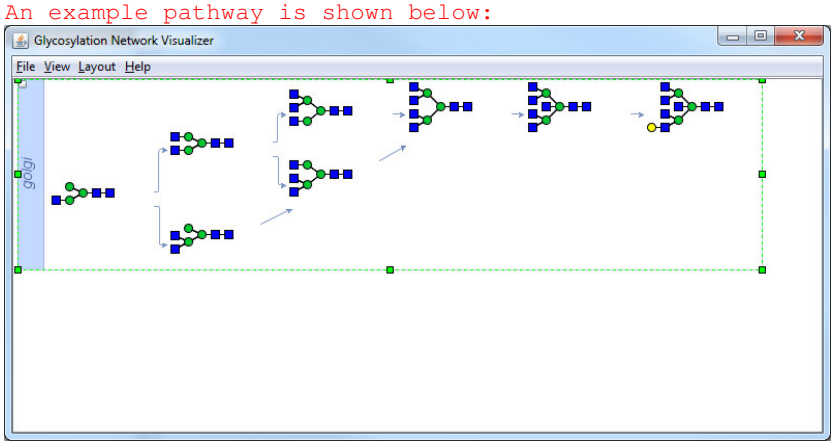


❖ Example 10.3: subnetgenbynumdel generates the subset model by deleting species

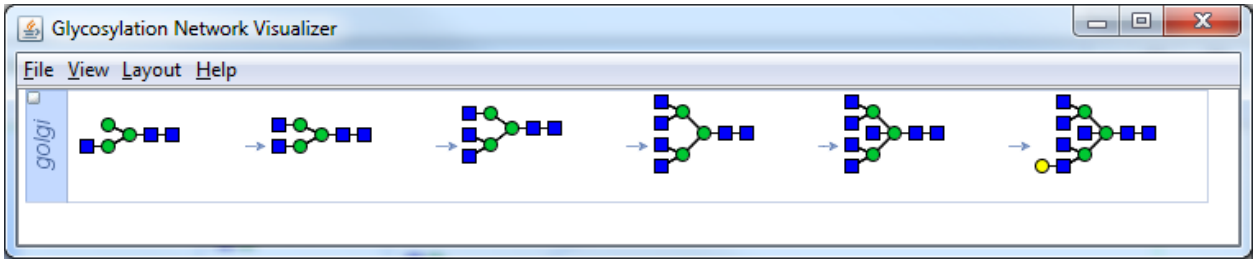
```
% load example path
pathexample = Pathway.loadmat('nlinkedpath.mat');
disp('An example pathway is shown below:');
glycanPathViewer(pathexample);

% generate a list of subset pathways by deleting specified number (say 2) of species from
% the network and view number of pathways that result
subpathlist = subnetgenbynumdel(pathexample,2);
fprintf(1,'Number of the subset pathways generated is: %i \n',subpathlist.length);

% to view any one of these models, say the 8th model in this list
disp('The eighth subset model is shown below:');
glycanPathViewer(subpathlist.get(8));
```

Output:

```
An example pathway is shown below:
```




```
Number of the subset pathways generated is: 28
The eighth subset model is shown below:
1
2
...
28
```



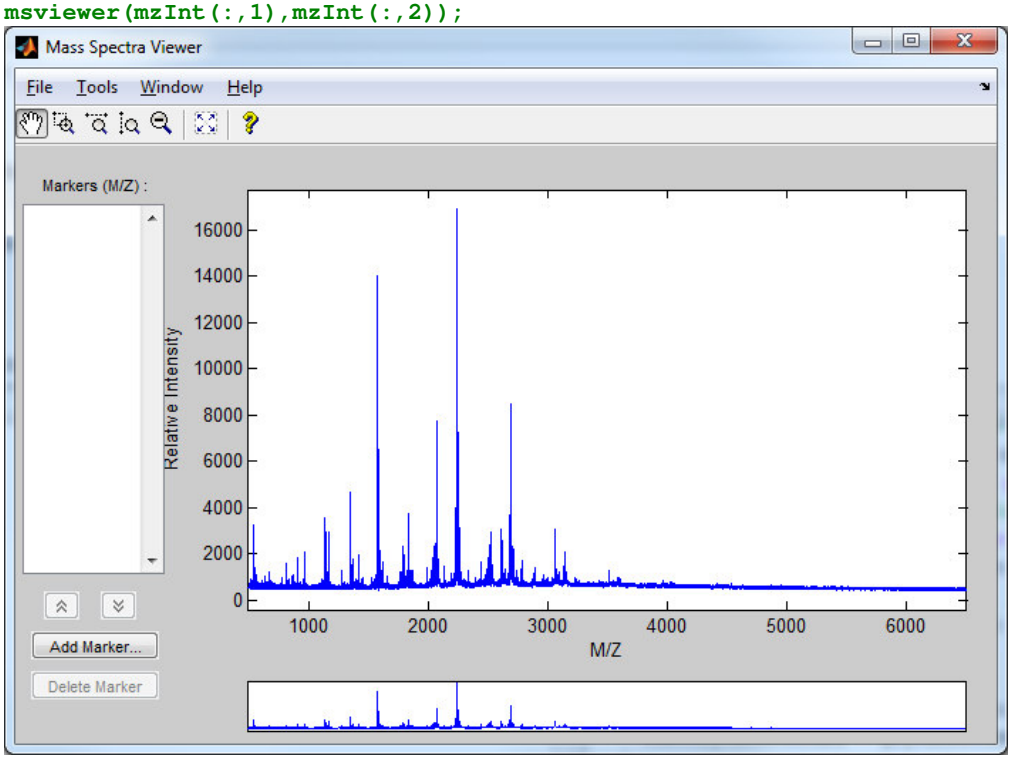
11. Mass spectrometry analysis

Two basic functions (***readMS*** and ***msprocess***) are available for processing mass spectrometry data. ***readMS*** reads the raw data from mass spectrometry. ***msprocess*** handles the MS data using a four-step procedure that includes background adjustment, data normalization, noise removal, and peak identification. Here, the MATLAB bioinformatics toolbox functions have been used to support ***msprocess*** command. Thus bioinformatics toolbox is required. Below we show examples for these two commands.

Once the peaks are identified above, the glycans associated with each peak can be annotated either manually or semi-automatically using programs like GlycanWorkBench. Annotated glycan information can then be used as input into GNAT to generate the reaction network as shown in Chapter 13.4 below.

❖ **Example 11.1: readMS reads the mass spectrometry data in msd format.**

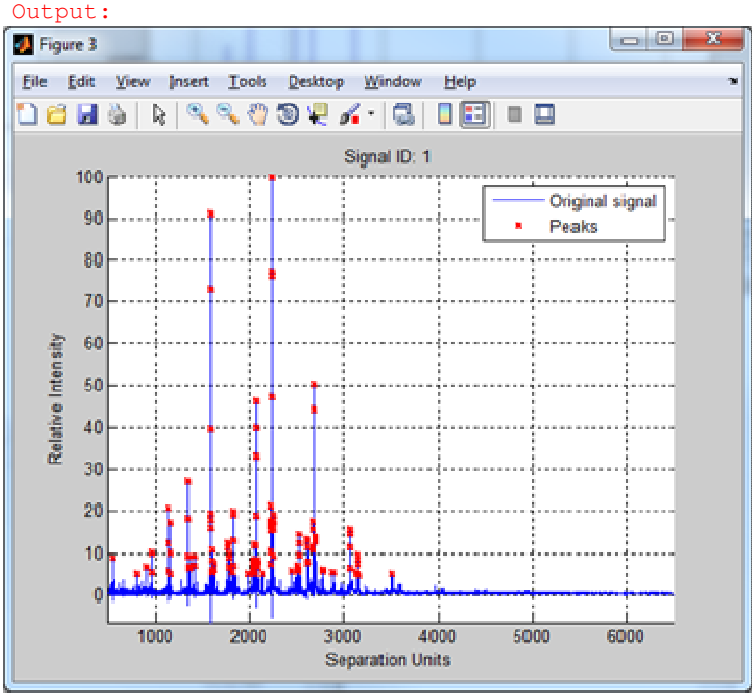
```
mzInt = readMS('testCHO.msd');
msviewer(mzInt(:,1),mzInt(:,2));
```



❖ **Example 11.2: msprocess converts raw peak data to a list of peaks and their half height width using a four step method. It starts with background adjustment, followed by the normalization, signal noise removal, and ends with peak finding.**

```
mzInt = readMS('testCHO.msd');
options.showplot = true;
[peaklist,pfwhh]=msprocess(mzInt);
```

Output:



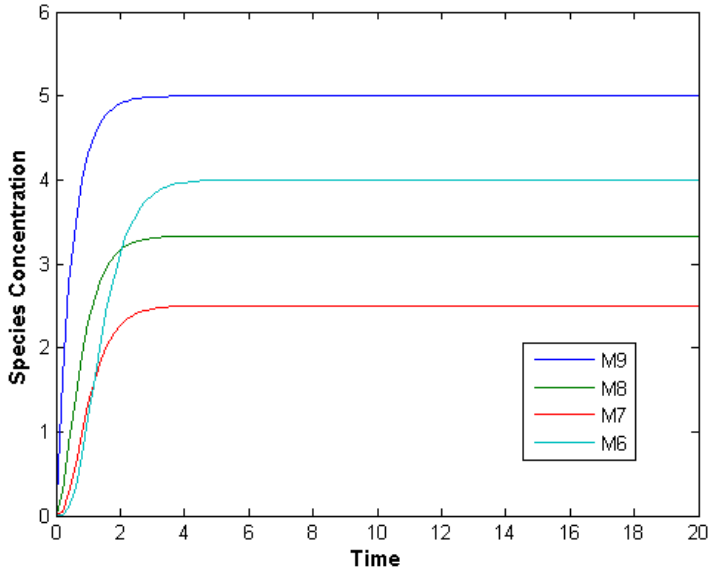
12 Steady-state and dynamic simulation

Two functions (***glycanDSim*** and ***glycanSSim***) are available for simulation of glycosylation reaction networks. ***glycanDSim*** simulates the initial value problem, i.e. the time course of the reaction network, by solving a set of ordinary differential equations (ODEs). ***glycanSSim*** simulates the steady state solution. Here, the MATLAB Optimization toolbox function ***fsolve*** is used to solve the algebraic equations that result from the ODEs at steady state. Below we show examples for these two commands.

❖ **Example 12.1: glycanDSim simulates the glycosylation reaction network at various times**

```
testGlycanNetModel = glycanNetSBMLread(which('gnat_test_sim.xml'));
exampleODEFileName = [testGlycanNetModel.glycanNet_sbmlmodel.id '.m'];
endTime=20;tsteps=100;

[tSpan, speciesconc, speciesname] = glycanDSim(testGlycanNetModel,...
    exampleODEFileName,endTime,tsteps);    % Performs calculation

% To see simulation result in plot type:
   plot(tSpan, speciesconc);legend(speciesname);
   xlabel('Time','fontsize',11,'fontweight','b');
   ylabel('Species Concentration','fontsize',11,'fontweight','b')
```

Output:



- Note: The reaction series in this pathway contains a constant source for species M9 and a sink for M6. This results in an output that reaches steady state within 4 time units as illustrated above.
- Here, `testGlycanNetModel` is the reaction network described in the MATLAB GlycanNetModel object. `exampleODEFileName` is the associated ode function file name that is generated using the SBML toolbox (type 'help glycanDSim' for more information). `endTime` is the end time point for the simulation. `tsteps` is the number of time points simulated.

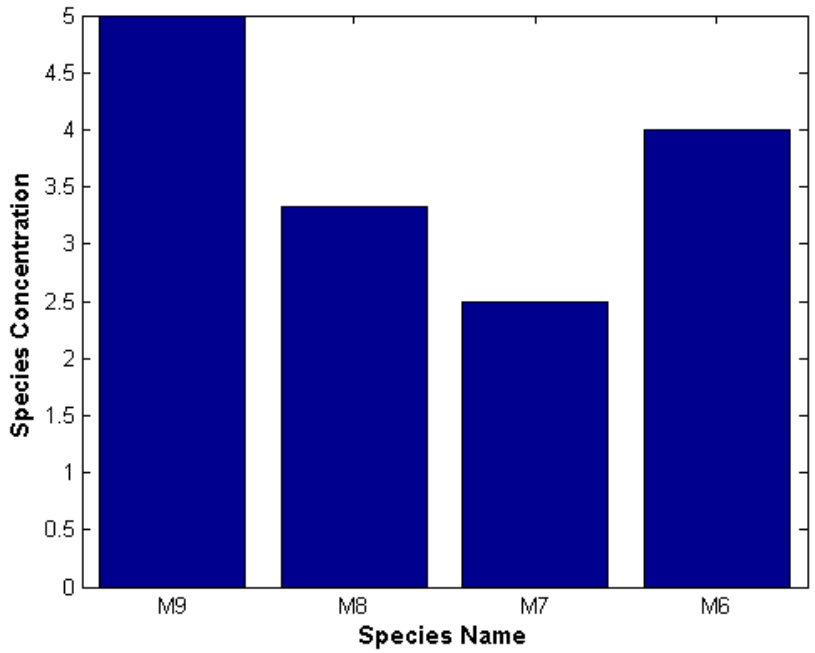
❖ **Example 12.2: glycanSSim simulates the glycosylation reaction network under steady state. This time we load the SBML file and run steady state calculation using fsolve**

```
testGlycanNetModel = glycanNetSBMLread('gnat_test_sim.xml');
exampleODEFileName = [testGlycanNetModel.glycanNet_sbmlmodel.id...
 '.m'];
[exitflag, speciesconc, speciesnames, funceval] = ...
glycanSSim(testGlycanNetModel,exampleODEFileName);

  % To see simulation result in plot, type:
  bar(speciesconc);
  set(gca,'Xtick',1:4,'XTickLabel',speciesnames);
  ylabel('Species Concentration','fontsize',11,'fontweight','b')
  xlabel('Species Name','fontsize',11,'fontweight','b');
```

Output:

Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	5	100	20	0.01	
1	10	0.00516564	0.0497	0.001	7.57118
2	15	5.32026e-009	5.14e-005	0.0001	0.0719942
3	20	5.64974e-017	5.52e-009	1e-005	7.40281e-005
4	25	5.87544e-027	5.33e-014	1e-006	7.63227e-009

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
```



13 User Case Studies

This section provides detailed usage guides for the simulation of glycosylation reaction networks (section 13.1) and network reconstruction (section 13.2 and 13.3)

13.1 Model simulation

This case study provides a detailed description of how to simulate a deterministic glycosylation reaction network. The example we use is the multi-compartment model by Umana and Bailey (Umana and Bailey 1997) that is commonly referred to as the 'UB1997' model.

It is best to run the commands in this section sequentially since loading the UB1997 model is a prerequisite for all subsequent steps.

Read the UB1997 Model from SBML format file

First, read the relevant SBML file using the **glycanNetSBMLread** function. This file has the glycan structures embedded in the species annotation field. The resulting GlycanNetModel object is called 'UB1997Model'. Note that reading a large file such as this can take a minute or two,

```
UB1997Model = glycanNetSBMLread('ub1997.xml');
```

Display Model Summary

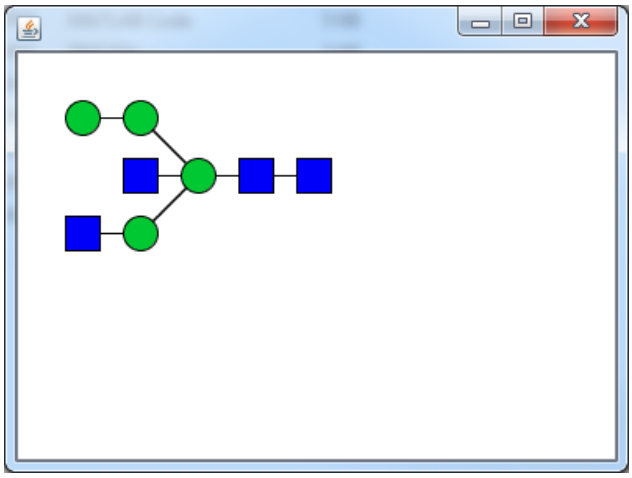
Display the summary of the model using '**get** methods'. This allows the user to determine the number of species, the number of reactions, and the number of compartments.

```
fprintf(1,'Model summary: \n');
fprintf(1,'Name of the model is  %s\n',UB1997Model.getName);
fprintf(1,'The number of reactions    is %i\n',...
UB1997Model.getGlycanPathway.getNReactions);
fprintf(1,'The number of species      is %i\n',...
UB1997Model.getGlycanPathway.getNSpecies);
fprintf(1,'The number of compartments is %i\n',...
UB1997Model.getCompartment.length);

Output:
Model summary:  Name of the model is UB1997Model
The number of reactions    is 297
The number of species      is 132
The number of compartments is 4
```

Glycan Structure Visualization

There are two ways to display specific glycans (e.g. the 32nd glycan) as shown below

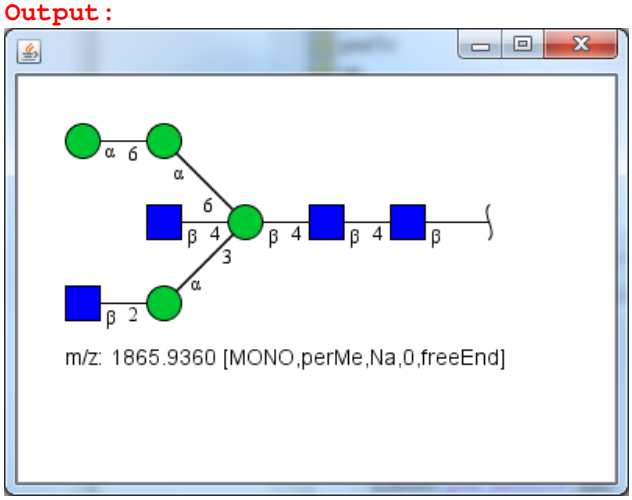
```
Option 1
ithNum = 32;
ithSpeciesStruct = UB1997Model.getGlycanPathway.getGlycanStruct(ithNum);
glycanViewer(ithSpeciesStruct);

Output:
```



Option 2

```
    gDispOption2 =
displayset('showmass',true,'showLinkage',true,'showRedEnd',true);
glycanJavaObj =  ithSpeciesStruct.structMat2Java;
glycanViewer(glycanJavaObj,gDispOption2);
```

Output :

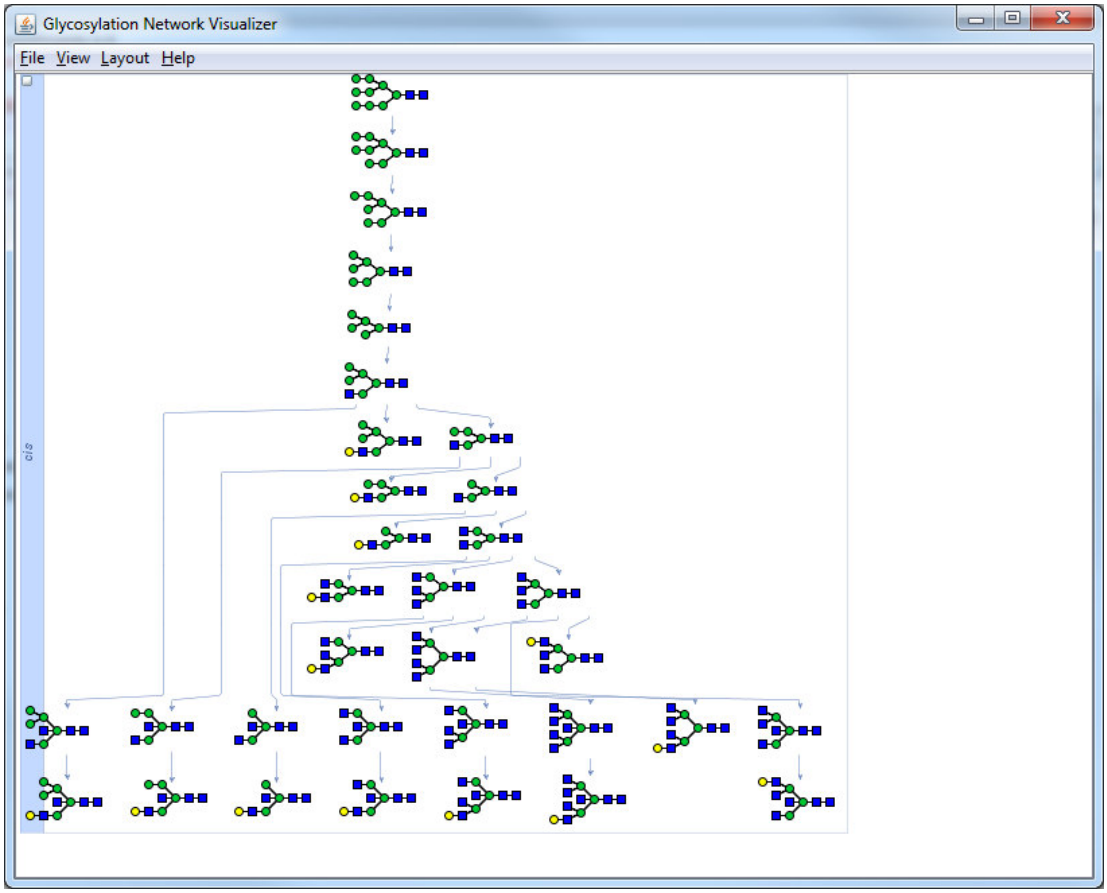


Network Visualization

Entire glycosylation reaction networks are visualized using the **glycanNetFileViewer** command. To display only the first compartment in hierarchical format, type the following command. The **glycanNetViewer** command may take a few moments.

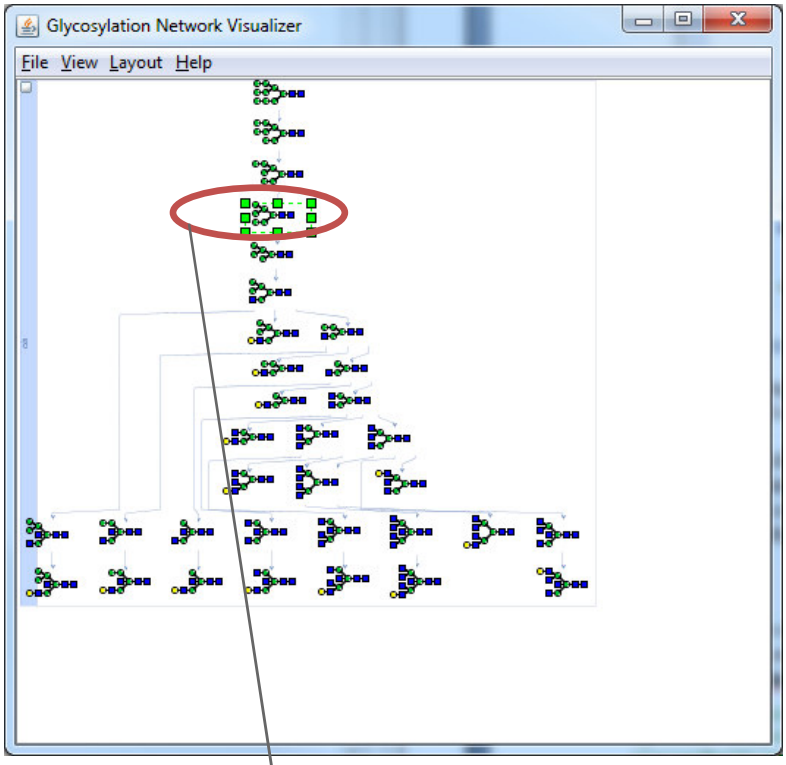
```
    options  = displayset('showRedEnd',false,'showLinkage',false,...
    'NetLayout','hierarchical');
    %options =
displayset(options,'dbname','GLYCOME','dbusername','usrGlycan');
    %options = displayset(options,'dbpassword','pwdGlycan');
    % use the options setup above only if the local PostgreSQL server is set up
and
    % the dbusername/dbpassword/dbname are the same as above
    glycanNetViewer(UB1997Model,options,0); % '0' for the first compt. only
```

Output :

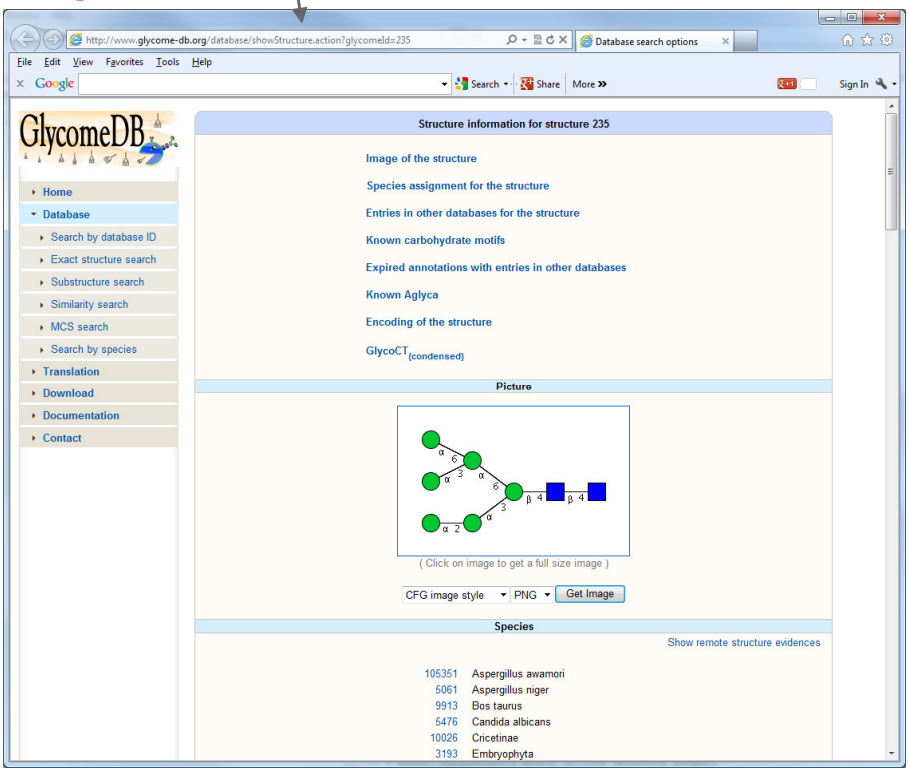


Try zooming in- and out- of the window by pressing the control key along with spinning the mouse wheel. This will reveal new details.

Double clicking on individual glycans will open the corresponding information webpage in the GlycomeDB database if the glycan structure is found in the local PostgreSQL server. If the server is not installed or if a corresponding glycan structure is not found, this operation will return the default GlycomeDB webpage. An example is shown below where the glycan circled in red is double-clicked:


Glycosylation Network Visualizer
File View Layout Help

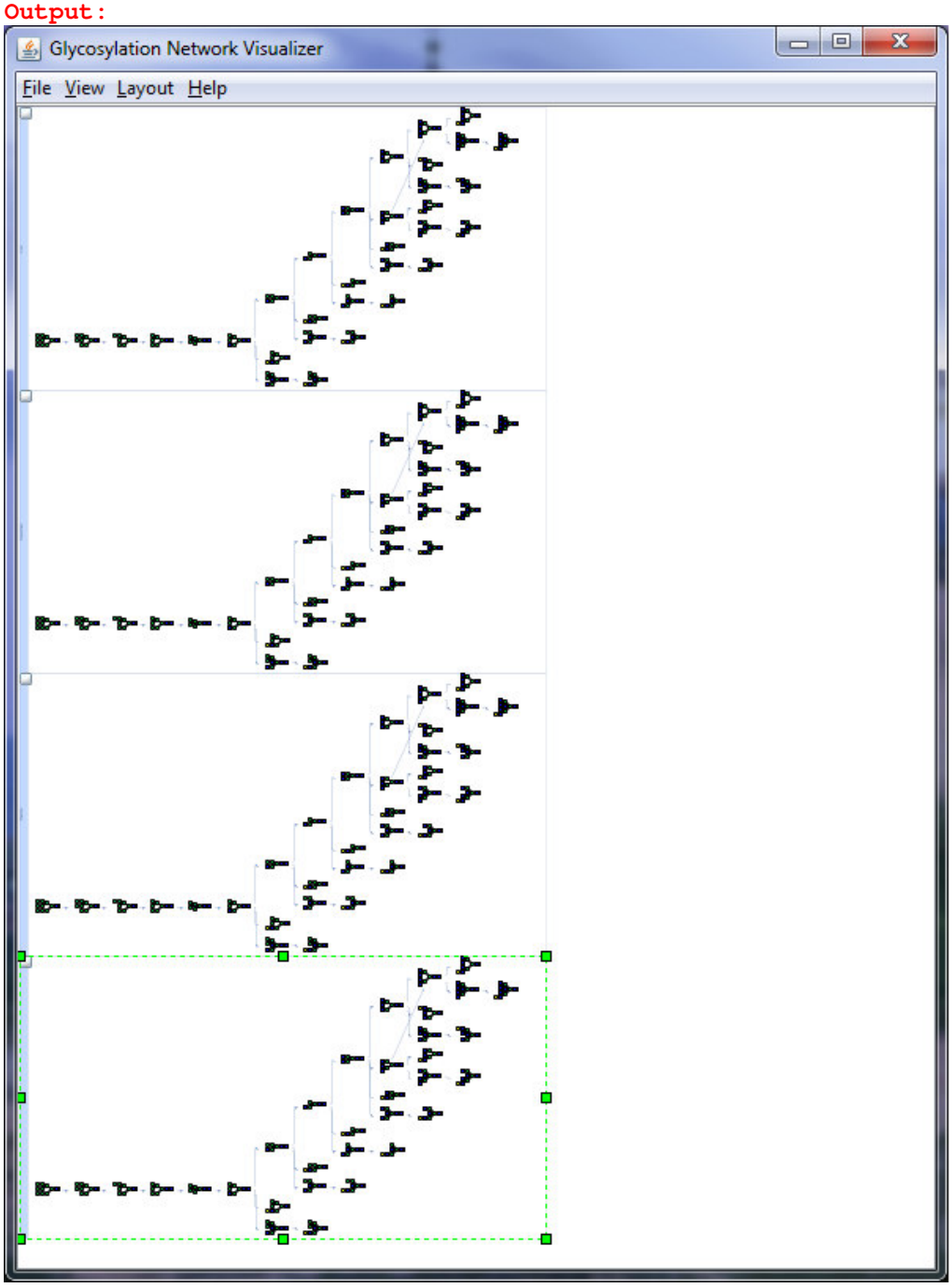
Output:


http://www.glycome-db.org/database/showStructure.action?glycomeId=235
Database search options
File Edit View Favorites Tools Help
Google
Search
Share
More
Sign In
GlycomeDB
Home
Database
Search by database ID
Exact structure search
Substructure search
Similarity search
MCS search
Search by species
Translation
Download
Documentation
Contact
Structure information for structure 235
Image of the structure
Species assignment for the structure
Entries in other databases for the structure
Known carbohydrate motifs
Expired annotations with entries in other databases
Known Aglyca
Encoding of the structure
GlycoCT [condensed]
Picture
(Click on image to get a full size image)
CFG image style
PNG
Get Image
Species
Show remote structure evidences
105351 Aspergillus awamori
5061 Aspergillus niger
9913 Bos taurus
5476 Candida albicans
10026 Cricetinae
3193 Embryophyta

GNAT also supports visualization of glycosylation networks in multiple compartments. No explicit option set-up is required in this case. Multiple compartments are defined in SBML file or **GlycanNetModel** object, or **Pathway** (Note: This command takes a few moments on slow computers). No facilities are available currently to automatically draw arrows between identical glycan species in different compartments.

```
options=displayset('NetFitToFrame',true,'NetLayout','CompactTree');
options=displayset(options,'NetFrameHeight',800,'NetFrameWidth',600);
glycanNetViewer(UB1997Model,options);
```

Output :



Click on the small box on the top left of individual compartments to close or open them. Use the control and mouse wheel to zoom in/out or use the view/layout menu to explore additional options.

Simulation of reaction network models and sample MATLAB scripts

Glycosylation reaction networks can be simulated using **glycanDSim**. The current version of glycanDSim uses ode15s to simulate the reaction network. This function does not yet support multi-compartment models. Thus in the example below, we use a custom function ('convertToSingleCmpt') to convert the multi-compartment model to a single compartment model. The mole fraction of bi- and tri- antennary glycans is the output (results shown in red below) from the MATLAB script (in green).

```
UB1997Model.glycanNet_sbmlmodel =
convertToSingleCompt(UB1997Model.glycanNet_sbmlmodel);
ubODEFileName = regexprep(UB1997Model.glycanNet_sbmlmodel.name,'.m',''); ubODEFileName =
[ubODEFileName '.m'];
if(exist(ubODEFileName,'file')~=2)         % generate a ODEFunction file using SBML
                                           % Toolbox if none exists previously
  WriteODEFunction(UB1997Model.glycanNet_sbmlmodel,ubODEFileName);
end
endTime = 100;                             % set up inputs
tsteps = 100;
options = odeset('RelTol',1e-12,'AbsTol',1e-6);  % run a simulation test based
                                      % on the parameter value defined in SBML
rehash toolboxcache;
[tspan, speciesConc, speciesnames] = glycanDSim(UB1997Model,ubODEFileName,...
endTime, tsteps);  binames={'M3Gn2_tgn';'M3Gn2G_tgn';'M3Gn2Gnb_tgn';'M3Gn2GnbG_tgn'};
biindex = findbyname(speciesnames,binames);
triprimenames={'M3Gn3prime_tgn';'M3Gn3primeG_tgn';'M3Gn3primeGnb_tgn';...
'M3Gn3primeGnbG_tgn'};
triindex =  findbyname(speciesnames,triprimenames);
[sumbiconc sumtriconc]=getBiTriConc(speciesConc,biindex,triindex);
fprintf(1,'the mole fraction of biantennary glycan structures is %f\n', sumbiconc);
fprintf(1,'the mole fraction of tri-prime-antennary glycan structures is %f\n',
sumtriconc);

Output:
 the mole fraction of biantennary glycan structures is 0.717286
 the mole fraction of tri-prime-antennary glycan structures is 0.239774
```

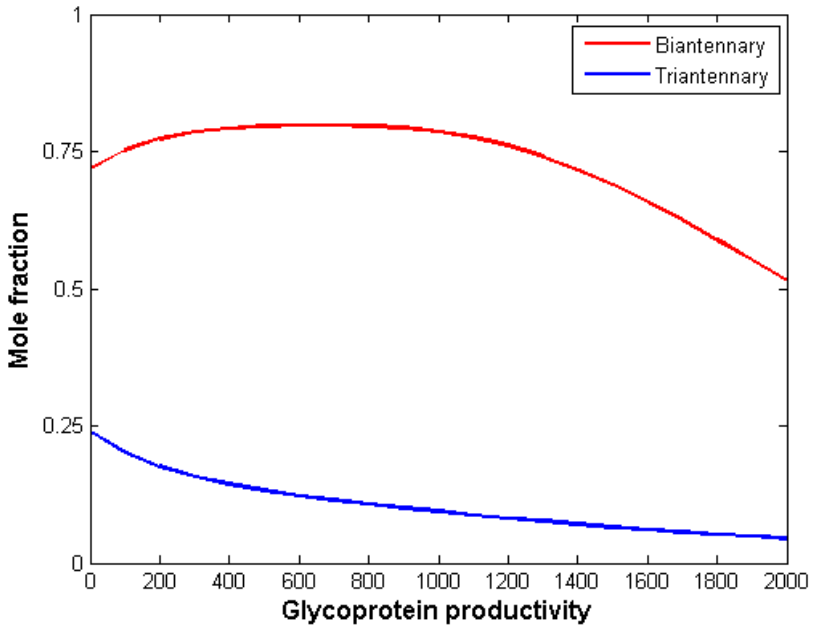
Writing MATLAB scripts with GNAT functions included

Small MATLAB scripts can be written to simulate the above glycosylation network for different values of protein productivity rate, the parameter qp.

```
qpname = 'qp';
qpArray=[0:100:2000];
qpArray(1,1)=1;
plotBi = zeros(1,length(qpArray));
plotTri =zeros(1,length(qpArray));
for i=1:length(qpArray);
 qp = qpArray(i);      % adjust m file
 adjustMFile(ubODEFileName, qpname, qp);
 options = odeset('RelTol',1e-12,'AbsTol',1e-6);
 [tspan, speciesConc, speciesnames] = glycanDSim(UB1997Model,ubODEFileName,...
 endTime,tsteps,options);
 [sumbiconc sumtriconc]=getBiTriConc(speciesConc,biindex, triindex);
 plotBi(i) =sumbiconc;
 plotTri(i)=sumtriconc;
```

```
     rehash toolboxcache;
    end
    adjustMFile(ubODEFileName, qpname, 1);
    figure;
    plot(qpArray,plotBi,'r',qpArray,plotTri,'b','Linewidth',2);
    axis([0 2000 0 1]);
    set(gca,'YTick',0:0.25:1);
    set(gca,'YTickLabel',{'0','0.25','0.5','0.75','1'});
    xlabel('Glycoprotein productivity','fontweight','b','fontsize',12); ylabel('Mole
fraction','fontweight','b','fontsize',12); legend('Biantennary','Triantennary');
```

Output:



The above simulation results are the same as Fig. 4 in Umana and Bailey 1997. This case study shows how simple MATLAB scripts can be used to augment the functionality available in GNAT!

13.2 Reconstruction of an O-linked glycosylation pathway

This case study provides a detailed description of how to construct a biosynthetic O-linked glycosylation pathway using the commands provided by GNAT2.0. The example shown is the model developed for O-linked reaction network leading to formation of glycan structures on PSGL-1 by Liu et al 2008. Details of enzyme properties and glycan structures used to construct the pathway can be seen in the main manuscript or Liu et al 2008 (Liu, Marathe et al. 2008).

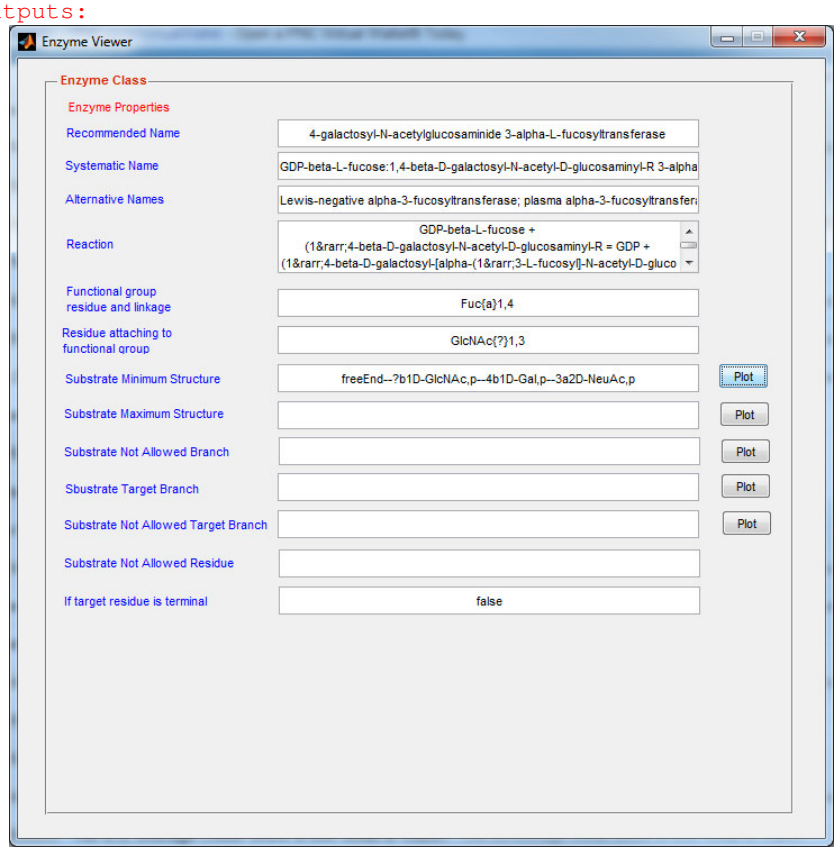
Definition of enzymes in O-linked core-2 structure extension and termination

First, five enzymes are considered for network reconstruction. These enzymes are β1,4GalT-IV, β1,3GlcNAc-T, α2,3ST3Gal-I/II, α2,3ST3Gal-IV and α1,3FT-VII. β1,4GalT-IV is a galactosyltransferase which transfers galactose to the terminal N-acetylgluocsamine of glycans in a β1,4-linkage. β1,3GlcNAcT adds GlcNAc to galactose residue in a β1,3 linkage, α2,3ST3Gal-I/II adds sialic acid to a β1,3 galactose residue in an α2,3 linkage, α2,3ST3Gal-IV adds sialic acid to a β1,4 galactose residue in an α2,3 linkage, α1,3FT-VII adds fucose to a β1,3 GlcNAc residue in a β1,3 linkage. These enzyme specificities are summarized in main manuscript. The enzymes are defined as follows:

```
Inputs:
galtiv = GTEnz([2;4;1;38]);
residueMap = load('residueTypes.mat');
galtiv.isTerminalTarget  = true;
galtiv.resfuncgroup = residueMap.allresidues('Gal');
glcnacResType = residueMap.allresidues('GlcNAc');
glcnacBond = GlycanBond('?','1');
galtiv.resAtt2FG = glcnacResType;
galtiv.linkresAtt2FG = struct('bond', glcnacBond,'anomer','b');
galtbond = GlycanBond('4','1');
galtiv.linkFG =  struct('anomer','b','bond',galtbond);
ignt = GTEnz([2;4;1;149]);
ignt.isTerminalTarget = true;
ignt.resfuncgroup = residueMap.allresidues('GlcNAc');
galResType = residueMap.allresidues('Gal');
galBond = GlycanBond('4','1');
ignt.resAtt2FG = galResType;
ignt.linkresAtt2FG = struct('bond', galBond,'anomer','b');
glcnacbond = GlycanBond('3','1');
ignt.linkFG = struct('anomer','b','bond',glcnacbond);
st3galI = GTEnz([2;4;99;4]);
st3galI.isTerminalTarget = true;
st3galI.resfuncgroup = residueMap.allresidues('NeuAc');
galResType = residueMap.allresidues('Gal');
galBond = GlycanBond('3','1');
st3galI.resAtt2FG = galResType;
st3galI.linkresAtt2FG = struct( 'bond', galBond,'anomer','b');
st3bond = GlycanBond('3','2');
st3galI.linkFG = struct('anomer','a','bond',st3bond);
st3galIV = GTEnz([2;4;99;6]);
st3galIV.isTerminalTarget = true;
st3galIV.resfuncgroup = residueMap.allresidues('NeuAc');
galResType = residueMap.allresidues('Gal');
galBond = GlycanBond('4','1');
st3bond = GlycanBond('3','2');
st3galIV.linkFG = struct('anomer','a','bond',st3bond);
st3galIV.resAtt2FG = galResType;
st3galIV.linkresAtt2FG = struct('bond', galBond,'anomer','b');
```

```
fucT7 = GTEnz([2;4;1;152]);
fucT7.isTerminalTarget = false;
fucT7.resfuncgroup = residueMap.allresidues('Fuc');
fuctbond = GlycanBond('4','1');
fucT7.linkFG = struct('anomer','a','bond',fuctbond);
glcnacResType = residueMap.allresidues('GlcNAc');
glcnacBond = GlycanBond('3','1');
fucT7.resAtt2FG = glcnacResType;
fucT7.linkresAtt2FG =  struct('bond', glcnacBond,'anomer','?');
fucT7.substNAResidue= residueMap.allresidues('Fuc');
fucT7.substMinStruct=glycanMLread('fucT7substmin.glycoct_xml');
enzViewer(fucT7)
```

Outputs:



Definition of glycan structures

Twelve glycan structures are specified using *Glycoworkench* toolbox and they are stored as Glycoct xml format. These files can be imported into GNAT as GlycanSpecies variables using *glycanMLread* command.

Input:

```
s1species = GlycanSpecies(glycanMLread('OG1.glycoct_xml'));
s2species = GlycanSpecies(glycanMLread('OG2.glycoct_xml'));
s4species = GlycanSpecies(glycanMLread('OG4.glycoct_xml'));
s5species = GlycanSpecies(glycanMLread('OG5.glycoct_xml'));
s7species = GlycanSpecies(glycanMLread('OG7.glycoct_xml'));
s10species = GlycanSpecies(glycanMLread('OG10.glycoct_xml'));
s11species = GlycanSpecies(glycanMLread('OG11.glycoct_xml'));
s13species = GlycanSpecies(glycanMLread('OG13.glycoct_xml'));
s14species = GlycanSpecies(glycanMLread('OG14.glycoct_xml'));
```

```
s17species = GlycanSpecies(glycanMLread('OG17.glycoct_xml'));
s18species = GlycanSpecies(glycanMLread('OG18.glycoct_xml'));
s19species = GlycanSpecies(glycanMLread('OG19.glycoct_xml'));
s20species = GlycanSpecies(glycanMLread('OG20.glycoct_xml'));
```

Inputs for glycosylation network reconstruction

Two input variables '*glycanArray*' and '*enzArray*' are created as *CellArrayList* variables and they are used to store a list of enzymes and glycan structure respectively.

Input:

```
glycanArray = CellArrayList;
glycanArray.add(s1species);
glycanArray.add(s2species);
glycanArray.add(s4species);
glycanArray.add(s5species);
glycanArray.add(s7species);
glycanArray.add(s10species);
glycanArray.add(s11species);
glycanArray.add(s13species);
glycanArray.add(s14species);
glycanArray.add(s17species);
glycanArray.add(s18species);
glycanArray.add(s19species);
glycanArray.add(s20species);
enzArray      = CellArrayList;
enzArray.add(ignt);
enzArray.add(galtiv);
enzArray.add(st3galI);
enzArray.add(st3galIV);
enzArray.add(fucT7);
```

Network reconstruction from inputs of glycans and enzymes

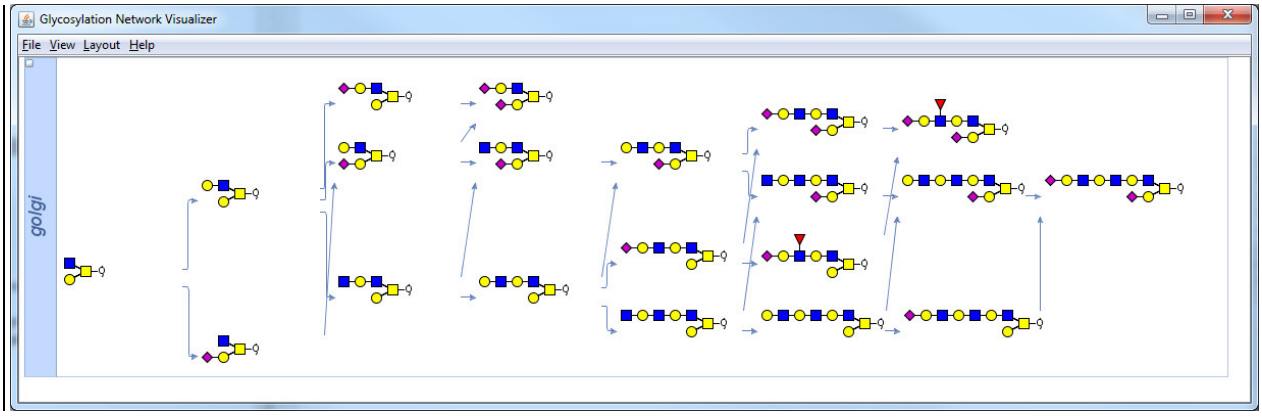
An O-linked glycosylation network can be constructed from 12 defined glycan structures and 5 enzymes using *inferGlyConnPath* command.

```
Input:
[isPath,oglycanpath]=inferGlyConnPath(glycanArray, enzArray);

Output:
the round: 1
number of total species in the pathway: 2
number of total reactions in the pathway: 1
...
```

Visualization of Reconstructed Network

The reconstructed network can be visualized using *glycanPathViewer* command and this network is

```
Input:
if(isPath)
      glycanPathViewer(oglycanpath);
end

Output:
```


Glycosylation Network Visualizer
File View Layout Help
golgi

13.3 Reconstruction of an N-linked glycosylation pathway

This example provides another demonstration about how to construct a biosynthetic N-linked glycosylation pathway using forward construction method. This exemplar network is predicted using enzymatic function and specificities. The network starts from a substrate M3Gn to form branched bi-, tri-, and tetra- antennary structures. Main manuscript provides details of enzyme properties and specificities in Table 1.

Enzyme definition

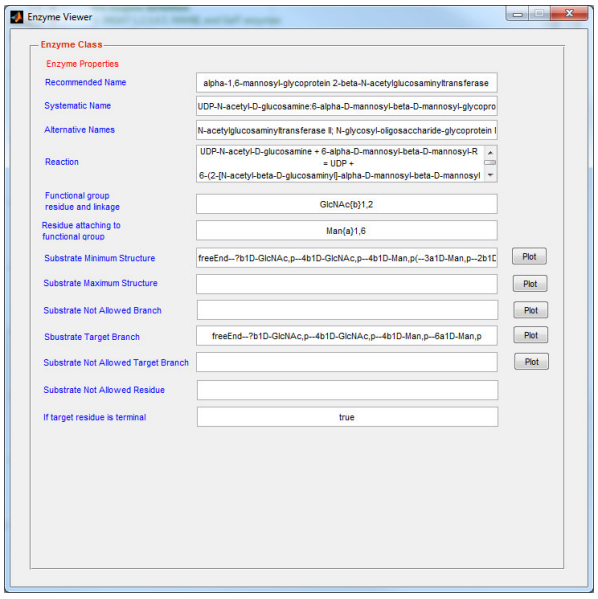
Enzymes defined in this exercise are MGAT2,3,4,5, ManII and GalT. Displays of enzymatic mechanism and their specificities are enabled using *enzViewer* command.

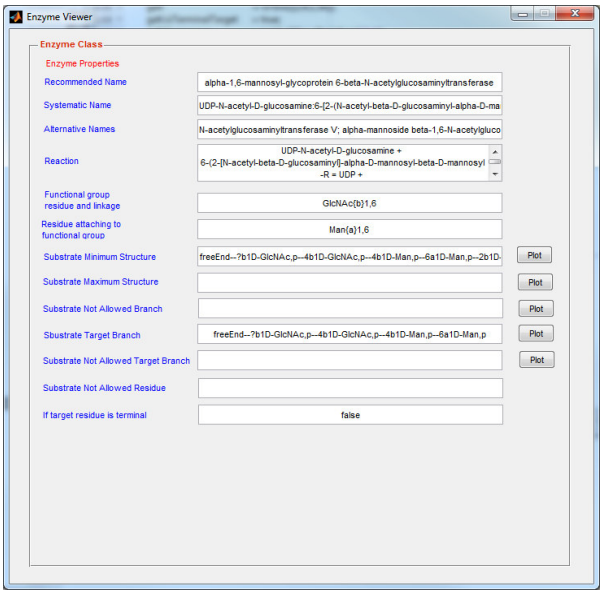
Input:

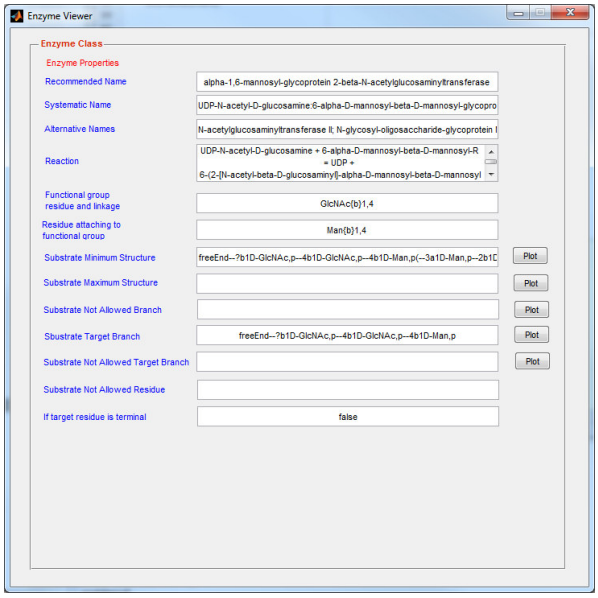
```
mgat2 = GTEnz([2;4;1;143]);
residueMap = load('residueTypes.mat');
mgat2.resfuncgroup = residueMap.allresidues('GlcNAc');
glcnacbond = GlycanBond('2','1');
mgat2.linkFG = struct('anomer','b','bond',glcnacbond);
manResType = residueMap.allresidues('Man');
manBond = GlycanBond('6','1');
mgat2.resAtt2FG = manResType;
mgat2.linkresAtt2FG = struct('bond', manBond,'anomer','a');
m3gn = glycanMLread('m3gn.glycoct_xml');
mgat2.isTerminalTarget = true;
mgat2.substMinStruct  = m3gn;
mgat2.targetBranch=glycanMLread('mgat2actingbranch.glycoct_xml');
mgat2.substNABranch = CellArrayList;
mgat2.substNABranch.add(...
 glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat2.substNABranch.add(...
 glycanMLread('mgat2substrateNAbranch.glycoct_xml'));
enzViewer(mgat2)

%% Definition and visualization of MANII enzyme
manii = GHEnz([3;2;1;114]);
manii.resfuncgroup = manResType;
manii.linkFG.anomer = 'a';
manBond(1,1)= GlycanBond('3','1');
manBond(2,1)= GlycanBond('6','1');
manii.linkFG.bond = manBond;
manii.resAtt2FG = manResType;
manii.prodMinStruct = glycanMLread('m3gn.glycoct_xml');
% mannosidase I can not work on M5 glycan
manii.substNABranch =glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');
enzViewer(manii)

%% Definition and visualization of MGAT3 enzyme
mgat3 = GTEnz([2;4;1;143]);
mgat3.resfuncgroup = residueMap.allresidues('GlcNAc');
manResType = residueMap.allresidues('Man');
manBond = GlycanBond('4','1');
mgat3.resAtt2FG = manResType;
mgat3.linkresAtt2FG = struct('bond', manBond,'anomer','b');
glcnacbond = GlycanBond('4','1');
mgat3.linkFG = struct('anomer','b','bond',glcnacbond);
m3gn = glycanMLread('m3gn.glycoct_xml');
mgat3.substMinStruct = m3gn;
mgat3.substNABranch = glycanMLread('NGlycanBisectGlcNAc.glycoct_xml');
```

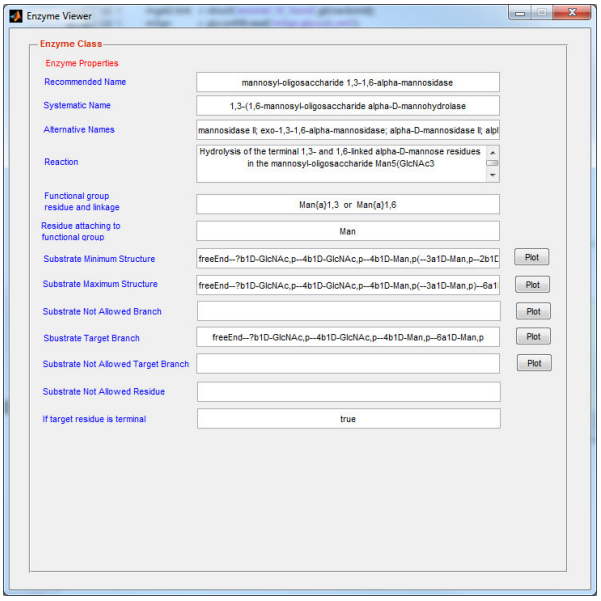
```
mgat3.targetBranch  = glycanMLread('mgat3targetbranch.glycoct_xml');
mgat3.substNAResidue = residueMap.allresidues('Gal');
enzViewer(mgat3)

%% Definition and visualization of MGAT4 enzyme
mgat4 = GTEnz([2;4;1;145]);
mgat4.resfuncgroup = residueMap.allresidues('GlcNAc');
manResType = residueMap.allresidues('Man');
manBond = GlycanBond('3','1');
mgat4.resAtt2FG = manResType;
mgat4.linkresAtt2FG = struct('bond', manBond,'anomer','a');
glcnacbond = GlycanBond('4','1');
mgat4.linkFG = struct('anomer','b','bond',glcnacbond);
m3gngn = glycanMLread('m3gngn.glycoct_xml');
mgat4.substMinStruct  = m3gngn;
mgat4.targetBranch = glycanMLread('mgat4targetbranch.glycoct_xml');
mgat4.substNABranch = CellArrayList;
mgat4.substNABranch.add(...
glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat4.substNABranch.add(...
glycanMLread('mgat4subsNAbranch.glycoct_xml'));
mgat4.substNAResidue = residueMap.allresidues('Gal');
enzViewer(mgat4)

%% Definition and visualization of MGAT5 enzyme
mgat5 = GTEnz([2;4;1;155]);
mgat5.resfuncgroup = residueMap.allresidues('GlcNAc');
manResType = residueMap.allresidues('Man');
manBond = GlycanBond('6','1');
mgat5.resAtt2FG = manResType;
mgat5.linkresAtt2FG = struct('bond', manBond,'anomer','a');
glcnacbond = GlycanBond('6','1');
mgat5.linkFG = struct('anomer','b','bond',glcnacbond);
m3gn = glycanMLread('m3gn.glycoct_xml');
%mgat5.substMinStruct  = m3gn;
mgat5.targetBranch = glycanMLread('mgat5targetbranch.glycoct_xml');
mgat5.substMinStruct = glycanMLread('mgat5substMinStruct.glycoct_xml');
mgat5.substNABranch = CellArrayList;
mgat5.substNABranch.add(...
glycanMLread('NGlycanBisectGlcNAc.glycoct_xml'));
mgat5.substNABranch.add(...
glycanMLread('mgat5substrateNAbranch.glycoct_xml'));
mgat5.substNAResidue = residueMap.allresidues('Gal');
enzViewer(mgat5)

%% Definition and visualization of GALT enzyme
galt = GTEnz([2;4;1;38]);
galt.isTerminalTarget = true;
galt.resfuncgroup = residueMap.allresidues('Gal');
glcnacResType = residueMap.allresidues('GlcNAc');
glcnacBond = GlycanBond('?','1');
galt.resAtt2FG = glcnacResType;
galt.linkresAtt2FG = struct('bond', glcnacBond,'anomer','b');
galtbond = GlycanBond('4','1');
galt.linkFG = struct('anomer','b','bond',galtbond);
galt.targetBranch = glycanMLread('galttargetbranch.glycoct_xml');
enzViewer(galt)
```

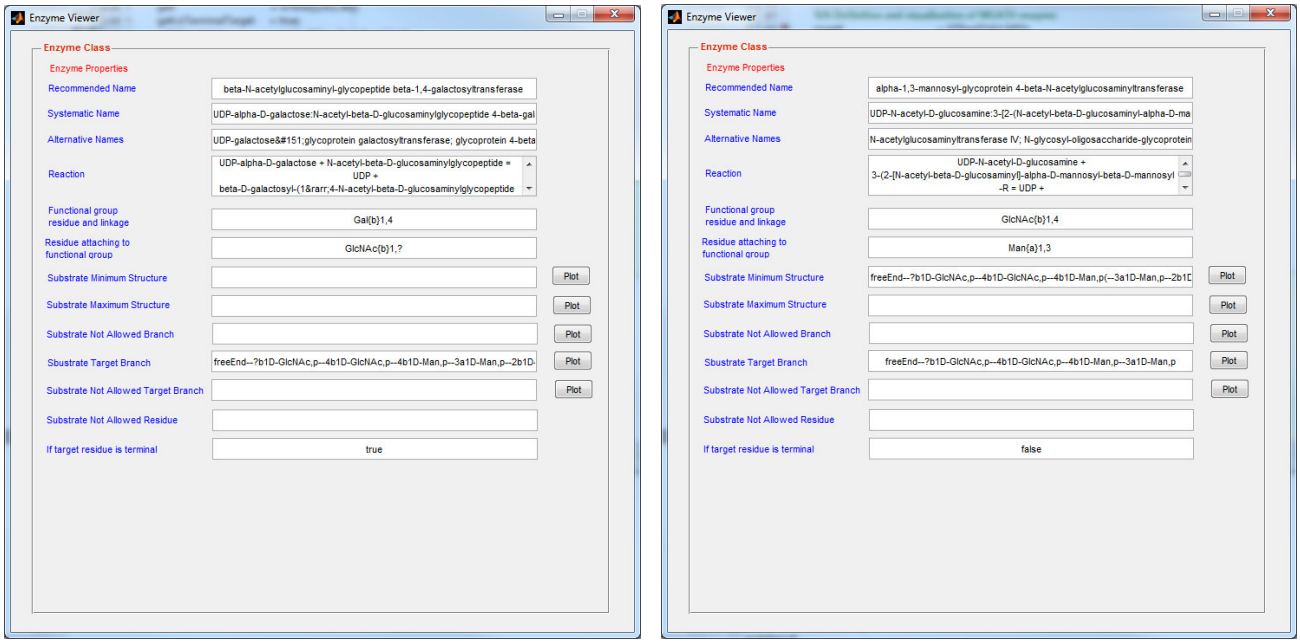
Output:


Enzyme Viewer
Enzyme Class
Enzyme Properties
Recommended Name
alpha-1,6-mannosyl-glycoprotein 2-beta-N-acetylglucosaminyltransferase
Systematic Name
UDP-N-acetyl-D-glucosamine:6-alpha-D-mannosyl-beta-D-mannosyl-glycopro
Alternative Names
N-acetylglucosaminyltransferase II; N-glycosyl-oligosaccharide-glycoprotein
Reaction
UDP-N-acetyl-D-glucosamine + 6-alpha-D-mannosyl-beta-D-mannosyl-R = UDP + 6-(2-[N-acetyl-beta-D-glucosaminyl]-alpha-D-mannosyl
Functional group residue and linkage
GlcNAc(b)1,2
Residue attaching to functional group
Man(a)1,6
Substrate Minimum Structure
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p(--3a1D-Man,p--2b1D
Substrate Maximum Structure
Substrate Not Allowed Branch
Sbustrate Target Branch
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p--6a1D-Man,p
Substrate Not Allowed Target Branch
Substrate Not Allowed Residue
If target residue is terminal
true
Plot


Enzyme Viewer
Enzyme Class
Enzyme Properties
Recommended Name
alpha-1,6-mannosyl-glycoprotein 6-beta-N-acetylglucosaminyltransferase
Systematic Name
UDP-N-acetyl-D-glucosamine:6-[2-(N-acetyl-beta-D-glucosaminyl-alpha-D-ma
Alternative Names
N-acetylglucosaminyltransferase V; alpha-mannoside beta-1,6-N-acetylgluco
Reaction
UDP-N-acetyl-D-glucosamine + 6-(2-[N-acetyl-beta-D-glucosaminyl]-alpha-D-mannosyl-beta-D-mannosyl -R = UDP +
Functional group residue and linkage
GlcNAc(b)1,6
Residue attaching to functional group
Man(a)1,6
Substrate Minimum Structure
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p--6a1D-Man,p--2b1D-
Substrate Maximum Structure
Substrate Not Allowed Branch
Sbustrate Target Branch
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p--6a1D-Man,p
Substrate Not Allowed Target Branch
Substrate Not Allowed Residue
If target residue is terminal
false
Plot


Enzyme Viewer
Enzyme Class
Enzyme Properties
Recommended Name
alpha-1,6-mannosyl-glycoprotein 2-beta-N-acetylglucosaminyltransferase
Systematic Name
UDP-N-acetyl-D-glucosamine:6-alpha-D-mannosyl-beta-D-mannosyl-glycopro
Alternative Names
N-acetylglucosaminyltransferase II; N-glycosyl-oligosaccharide-glycoprotein
Reaction
UDP-N-acetyl-D-glucosamine + 6-alpha-D-mannosyl-beta-D-mannosyl-R = UDP + 6-(2-[N-acetyl-beta-D-glucosaminyl]-alpha-D-mannosyl
Functional group residue and linkage
GlcNAc(b)1,4
Residue attaching to functional group
Man(b)1,4
Substrate Minimum Structure
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p(--3a1D-Man,p--2b1D
Substrate Maximum Structure
Substrate Not Allowed Branch
Sbustrate Target Branch
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p
Substrate Not Allowed Target Branch
Substrate Not Allowed Residue
If target residue is terminal
false
Plot

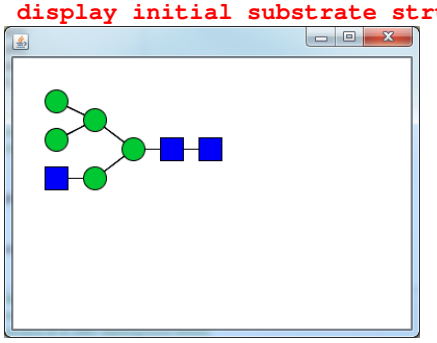

Enzyme Viewer
Enzyme Class
Enzyme Properties
Recommended Name
mannosyl-oligosaccharide 1,3-1,6-alpha-mannosidase
Systematic Name
1,3-(1,6-mannosyl-oligosaccharide alpha-D-mannohydrolase
Alternative Names
mannosidase II; exo-1,3-1,6-alpha-mannosidase; alpha-D-mannosidase II; alpl
Reaction
Hydrolysis of the terminal 1,3- and 1,6-linked alpha-D-mannose residues in the mannosyl-oligosaccharide Man5(GlcNAc3
Functional group residue and linkage
Man(a)1,3 or Man(a)1,6
Residue attaching to functional group
Man
Substrate Minimum Structure
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p(--3a1D-Man,p--2b1D
Substrate Maximum Structure
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p(--3a1D-Man,p)--6a1
Substrate Not Allowed Branch
Sbustrate Target Branch
freeEnd--?b1D-GlcNAc,p--4b1D-GlcNAc,p--4b1D-Man,p--6a1D-Man,p
Substrate Not Allowed Target Branch
Substrate Not Allowed Residue
If target residue is terminal
true
Plot



Initial substrate definition

Initial substrate is defined as M5Gn. Its structure can be visualized using glycanViewer command

```
Input:
m5gnspecies = GlycanSpecies(glycanMLread('m5gn.glycoct_xml')) ;
fprintf(1,'display initial substrate structure\n');
glycanViewer(m5gnspecies.glycanStruct);

Output:
 display initial substrate structure
```



Enzymes and glycans added to CellArrayList variables

'substrateArray' and 'enzArray' are created to store initial substrate and enzymes which might act on substrate.

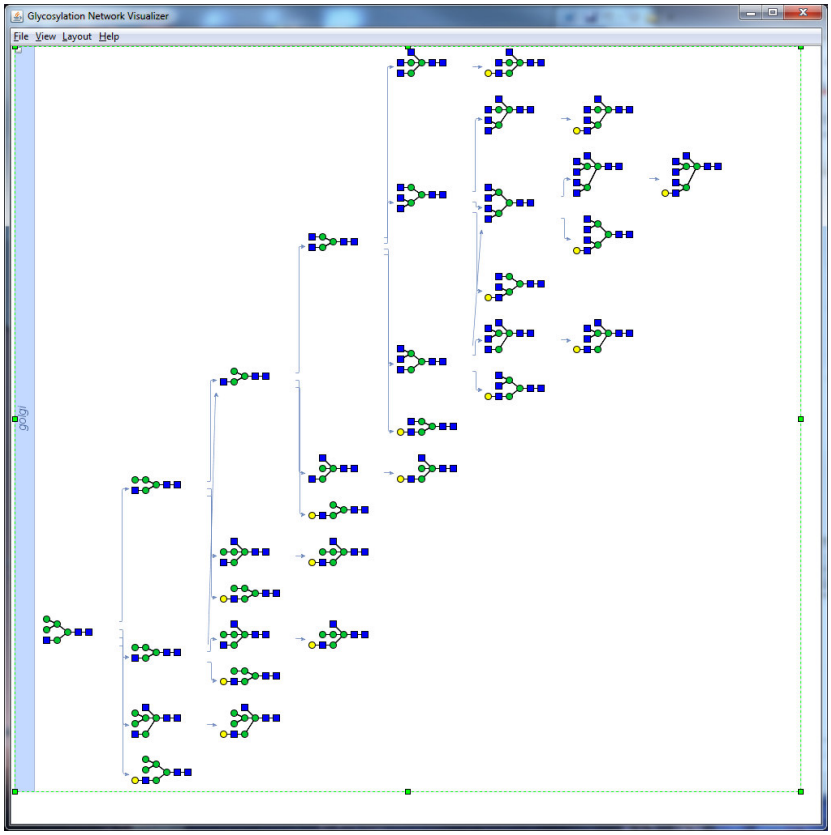
```
Input:
substrateArray = CellArrayList;
enzArray = CellArrayList;
substrateArray.add(m5gnspecies);
enzArray.add(manii);
enzArray.add(mgat2);
```

```
enzArray.add(mgat3);
enzArray.add(mgat4);
enzArray.add(mgat5);
enzArray.add(galt);
```

Theoretical prediction of pathway using forward construction

inferGlyForwPath command is used to construct a theoretical pathway and the pathway can be visualized using *glycanPathViewer* command.

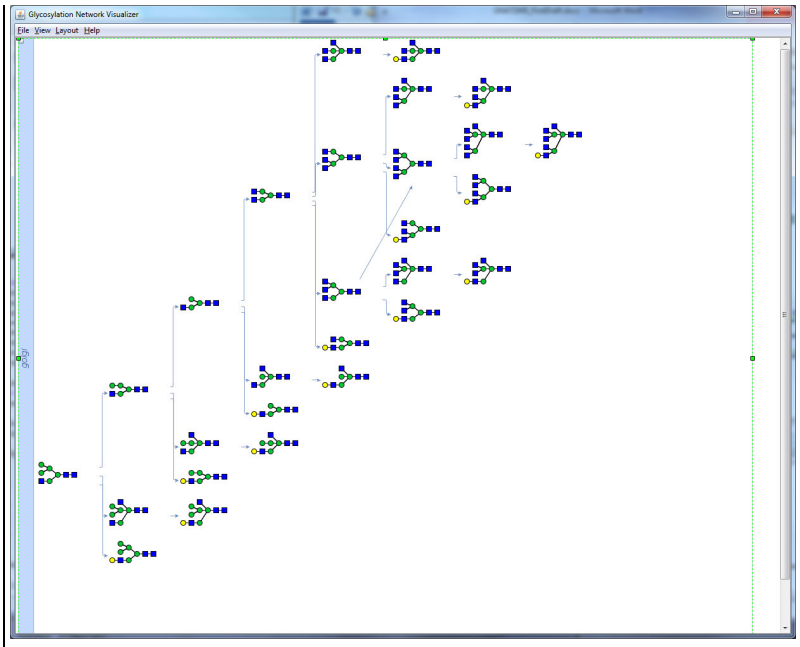
```
Input:
[isPath,nglycanpath]=inferGlyForwPath(substrateArray,enzArray);
if(isPath)
   glycanPathViewer(nglycanpath);
end
Output:
round:1
number of total species in the pathway: 5
number of total reactions in the pathway: 4
...
```



Removal of isomer structures

Removal of isomer structures from derived pathway leads to a new form of pathway.

```
Input:
nlinkedpath2 = removeIsomerStruct(nglycanpath);
glycanPathViewer(nlinkedpath2);

Output:
```


Glycosylation Network Visualizer
File View Layout Help

13.4 Pathway reconstruction from MS data

This user case constructs a glycosylation pathway using annotated MS data. These data were obtained upon profiling the N-glyans of Chinese Hamster Ovary (CHO) cells cultured in suspension (North, Huang et al. 2010). The glycan structures annotated for mass spectrometry peaks (m/z value between 1500 and 3250) can be seen at CFG website (http://www.functionalglycomics.org/fg/).

Definition of glycan inputs

Initial glycans (43 structures) are defined based on 20 experimental peaks and these are stored in substrateArray variable. Please note that glycan structure annotations were done by North et al 2010 using Glycoworkbench and Cartoonist software. Some of the glycan structure are isomers.

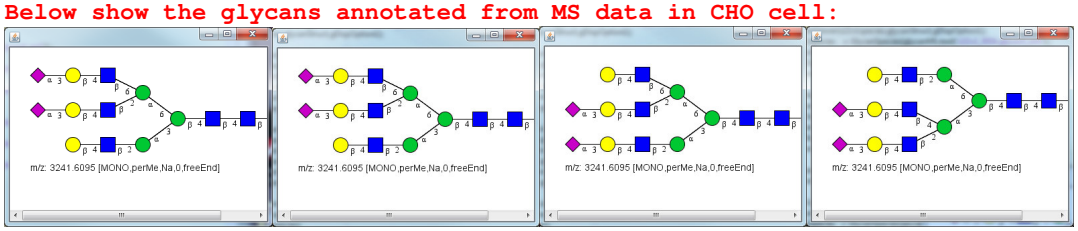
```
Input:
gDispOption1       = displayset('showmass',true,'showLinkage',true,...
      'showRedEnd',true);
s1species       = GlycanSpecies(glycanMLread('s1_1579.glycoct_xml')) ;
glycanViewer(s1species.glycanStruct,gDispOption1);
s2species       = GlycanSpecies(glycanMLread('s2_1590.glycoct_xml')) ;
glycanViewer(s2species.glycanStruct,gDispOption1);
s3v1species       = GlycanSpecies(glycanMLread('s3v1_1620.glycoct_xml')) ;
glycanViewer(s3v1species.glycanStruct,gDispOption1);
s3v2species       = GlycanSpecies(glycanMLread('s3v2_1620.glycoct_xml')) ;
glycanViewer(s3v2species.glycanStruct,gDispOption1);
s4species       = GlycanSpecies(glycanMLread('s4_1783.glycoct_xml')) ;
glycanViewer(s4species.glycanStruct,gDispOption1);
s5species       = GlycanSpecies(glycanMLread('s5_1835.glycoct_xml')) ;
glycanViewer(s5species.glycanStruct,gDispOption1);
s6v1species       = GlycanSpecies(glycanMLread('s6v1_1865.glycoct_xml')) ;
glycanViewer(s6v1species.glycanStruct,gDispOption1);
s6v2species       = GlycanSpecies(glycanMLread('s6v2_1865.glycoct_xml')) ;
glycanViewer(s6v2species.glycanStruct,gDispOption1);
s7species       = GlycanSpecies(glycanMLread('s7_1988.glycoct_xml')) ;
glycanViewer(s7species.glycanStruct,gDispOption1);
s8v1species       = GlycanSpecies(glycanMLread('s8v1_2040.glycoct_xml')) ;
glycanViewer(s8v1species.glycanStruct,gDispOption1);
s8v2species       = GlycanSpecies(glycanMLread('s8v2_2040.glycoct_xml')) ;
glycanViewer(s8v2species.glycanStruct,gDispOption1);
s9species       = GlycanSpecies(glycanMLread('s9_2070.glycoct_xml')) ;
glycanViewer(s9species.glycanStruct,gDispOption1);
s10species       = GlycanSpecies(glycanMLread('s10_2081.glycoct_xml'));
glycanViewer(s10species.glycanStruct,gDispOption1);
s11species      = GlycanSpecies(glycanMLread('s11_2192.glycoct_xml')) ;
glycanViewer(s11species.glycanStruct,gDispOption1);
s12species      = GlycanSpecies(glycanMLread('s12_2244.glycoct_xml')) ;
glycanViewer(s12species.glycanStruct,gDispOption1);
s13species       = GlycanSpecies(glycanMLread('s13_2396.glycoct_xml')) ;
glycanViewer(s13species.glycanStruct,gDispOption1);
s14v1species       = GlycanSpecies(glycanMLread('s14v1_2431.glycoct_xml')) ;
glycanViewer(s14v1species.glycanStruct,gDispOption1);
```

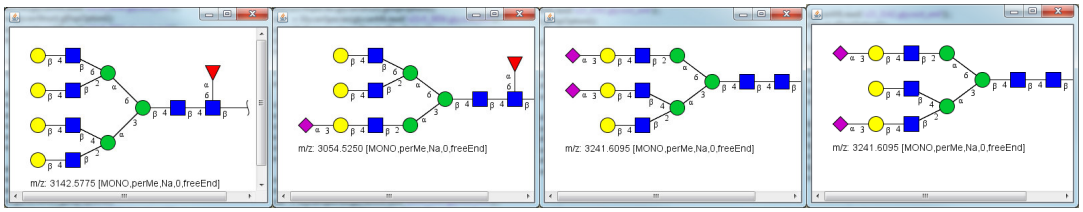
```
s14v2species       = GlycanSpecies(glycanMLread('s14v2_2431.glycoct_xml')) ;
glycanViewer(s14v2species.glycanStruct,gDispOption1);
s15species       = GlycanSpecies(glycanMLread('s15_2489.glycoct_xml')) ;
glycanViewer(s15species.glycanStruct,gDispOption1);
s16v1species       = GlycanSpecies(glycanMLread('s16v1_2519.glycoct_xml')) ;
glycanViewer(s16v1species.glycanStruct,gDispOption1);
s16v2species       = GlycanSpecies(glycanMLread('s16v2_2519.glycoct_xml')) ;
glycanViewer(s16v2species.glycanStruct,gDispOption1);
s17v1species       = GlycanSpecies(glycanMLread('s17v1_2605.glycoct_xml')) ;
glycanViewer(s17v1species.glycanStruct,gDispOption1);
s17v2species       = GlycanSpecies(glycanMLread('s17v2_2605.glycoct_xml')) ;
glycanViewer(s17v2species.glycanStruct,gDispOption1);
s18v1species              = GlycanSpecies(glycanMLread('s18v1_2693.glycoct_xml')) ;
glycanViewer(s18v1species.glycanStruct,gDispOption1);
s18v2species              = GlycanSpecies(glycanMLread('s18v2_2693.glycoct_xml')) ;
glycanViewer(s18v2species.glycanStruct,gDispOption1);
s19v1species       = GlycanSpecies(glycanMLread('s19v1_2880.glycoct_xml')) ;
glycanViewer(s19v1species.glycanStruct,gDispOption1);
s19v2species       = GlycanSpecies(glycanMLread('s19v2_2880.glycoct_xml')) ;
glycanViewer(s19v2species.glycanStruct,gDispOption1);
s19v3species       = GlycanSpecies(glycanMLread('s19v3_2880.glycoct_xml')) ;
glycanViewer(s19v3species.glycanStruct,gDispOption1);
s19v4species       = GlycanSpecies(glycanMLread('s19v4_2880.glycoct_xml')) ;
glycanViewer(s19v4species.glycanStruct,gDispOption1);
s19v5species       = GlycanSpecies(glycanMLread('s19v5_2880.glycoct_xml')) ;
glycanViewer(s19v5species.glycanStruct,gDispOption1);
s19v6species       = GlycanSpecies(glycanMLread('s19v6_2880.glycoct_xml')) ;
glycanViewer(s19v6species.glycanStruct,gDispOption1);
s20v1species              = GlycanSpecies(glycanMLread('s20v1_2938.glycoct_xml')) ;
glycanViewer(s20v1species.glycanStruct,gDispOption1);
s20v2species              = GlycanSpecies(glycanMLread('s20v2_2938.glycoct_xml')) ;
glycanViewer(s20v2species.glycanStruct,gDispOption1);
s21species              = GlycanSpecies(glycanMLread('s21v1_2968.glycoct_xml')) ;
glycanViewer(s21species.glycanStruct,gDispOption1);
s22v1species       = GlycanSpecies(glycanMLread('s22v1_3054.glycoct_xml')) ;
glycanViewer(s22v1species.glycanStruct,gDispOption1);
s22v2species       = GlycanSpecies(glycanMLread('s22v2_3054.glycoct_xml')) ;
glycanViewer(s22v2species.glycanStruct,gDispOption1);
s22v3species       = GlycanSpecies(glycanMLread('s22v3_3054.glycoct_xml')) ;
glycanViewer(s22v3species.glycanStruct,gDispOption1);
s22v4species       = GlycanSpecies(glycanMLread('s22v4_3054.glycoct_xml')) ;
glycanViewer(s22v4species.glycanStruct,gDispOption1);
s22v5species       = GlycanSpecies(glycanMLread('s22v5_3054.glycoct_xml')) ;
glycanViewer(s22v5species.glycanStruct,gDispOption1);
s22v6species       = GlycanSpecies(glycanMLread('s22v6_3054.glycoct_xml')) ;
glycanViewer(s22v6species.glycanStruct,gDispOption1);
s23species              = GlycanSpecies(glycanMLread('s23_3142.glycoct_xml')) ;
glycanViewer(s23species.glycanStruct,gDispOption1);
s24v1species     = GlycanSpecies(glycanMLread('s24v1_3241.glycoct_xml')) ;
glycanViewer(s24v1species.glycanStruct,gDispOption1);
s24v2species     = GlycanSpecies(glycanMLread('s24v2_3241.glycoct_xml')) ;
glycanViewer(s24v2species.glycanStruct,gDispOption1);
s24v3species     = GlycanSpecies(glycanMLread('s24v3_3241.glycoct_xml')) ;
glycanViewer(s24v3species.glycanStruct,gDispOption1);
```

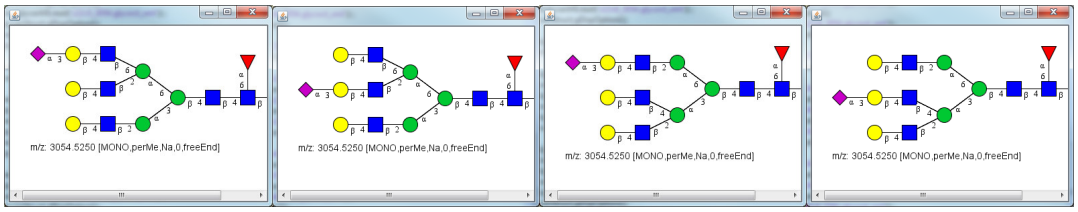
```
s24v4species      = GlycanSpecies(glycanMLread('s24v4_3241.glycoct_xml')) ;
glycanViewer(s24v4species.glycanStruct,gDispOption1);
s24v5species      = GlycanSpecies(glycanMLread('s24v5_3241.glycoct_xml')) ;
glycanViewer(s24v5species.glycanStruct,gDispOption1);
s24v6species      = GlycanSpecies(glycanMLread('s24v6_3241.glycoct_xml')) ;
glycanViewer(s24v6species.glycanStruct,gDispOption1);

substrateArray = CellArrayList;
substrateArray.add(s1species);
substrateArray.add(s2species);
substrateArray.add(s3v1species);
substrateArray.add(s3v2species);
substrateArray.add(s5species);
substrateArray.add(s6v1species);
substrateArray.add(s6v2species);
substrateArray.add(s8v1species);
substrateArray.add(s8v2species);
substrateArray.add(s9species);
substrateArray.add(s10species);
substrateArray.add(s12species);
substrateArray.add(s14v1species);
substrateArray.add(s14v2species);
substrateArray.add(s15species);
substrateArray.add(s16v1species);
substrateArray.add(s16v2species);
substrateArray.add(s17v1species);
substrateArray.add(s17v2species);
substrateArray.add(s18v1species);
substrateArray.add(s18v2species);
substrateArray.add(s19v1species);
substrateArray.add(s19v2species);
substrateArray.add(s19v3species);
substrateArray.add(s19v4species);
substrateArray.add(s19v5species);
substrateArray.add(s19v6species);
substrateArray.add(s20v1species);
substrateArray.add(s20v2species);
substrateArray.add(s21species);
substrateArray.add(s22v1species);
substrateArray.add(s22v2species);
substrateArray.add(s22v3species);
substrateArray.add(s22v4species);
substrateArray.add(s22v5species);
substrateArray.add(s22v6species);
substrateArray.add(s23species);
substrateArray.add(s24v1species);
substrateArray.add(s24v2species);
substrateArray.add(s24v3species);
substrateArray.add(s24v4species);
substrateArray.add(s24v5species);
substrateArray.add(s24v6species);
```

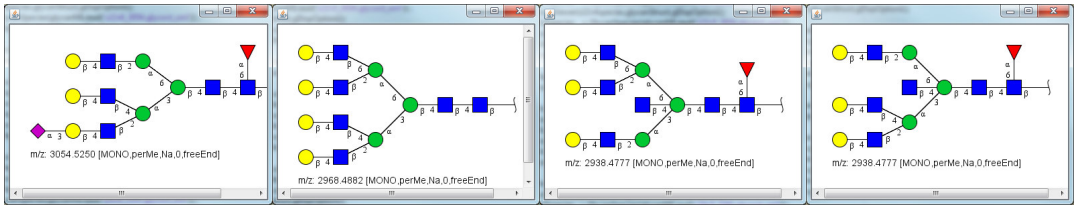
Output:

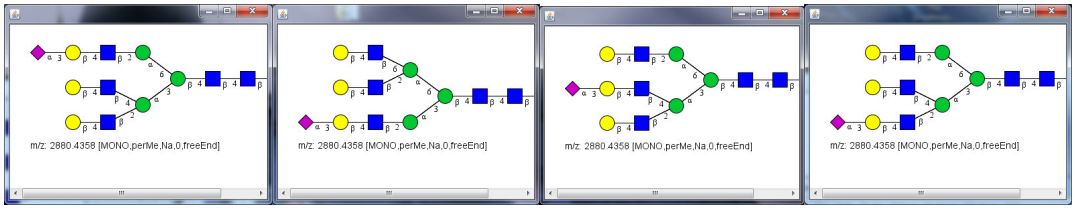
Below show the glycans annotated from MS data in CHO cell:

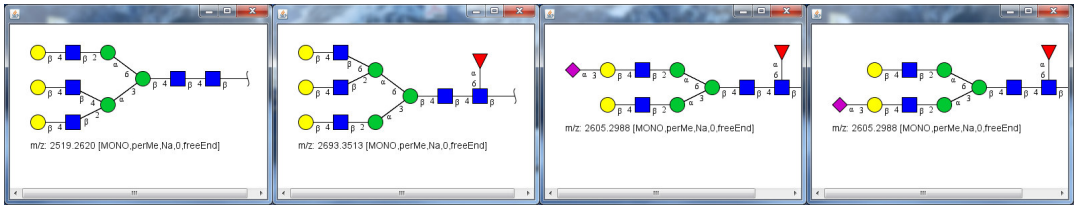


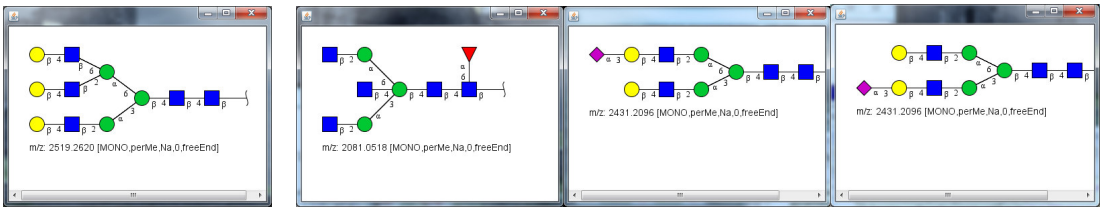












Enzyme definition

An array of known enzymes involved in N-linked glycosylation pathway is constructed. It includes 9 enzyme including MANII, MGAT 1, MGAT2, MGAT3, MGAT 4, MGAT 5, GalT, FucT, SiaT enzymes as listed below.

Input :

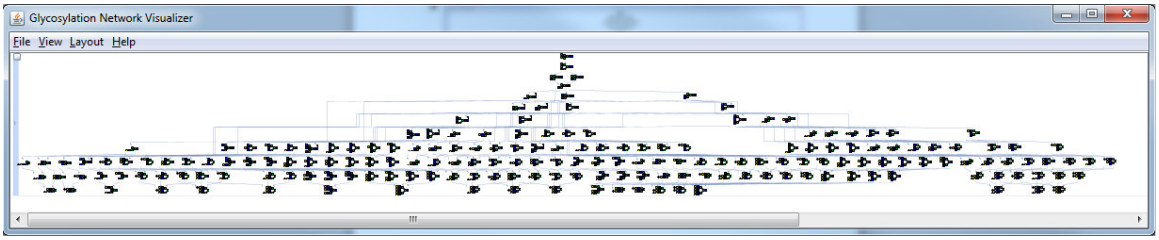
```
%MAN II, MGAT 1,2,3,4,5, GalT, FucT, SiaT enzymes
% are loaded from a pre-defined local database.
%
  enzdbmatfilename     = 'glyenzDB.mat';
  enzdb = enzdbmatLoad(enzdbmatfilename);
  mgat1 = enzdb('mgat1');
  mgat2 = enzdb('mgat2');
  mgat3 = enzdb('mgat3');
  mgat4 = enzdb('mgat4');
  mgat5 = enzdb('mgat5');
  manii   = enzdb('manii');
  galt       = enzdb('galt');
  siaT       = enzdb('siaT');
  fucT8 = enzdb('fucT8');
% Store enzymes in CellArrayList variables
% enzArray are created to store enzymes which might act on substrates.
```

```
enzArray = CellArrayList;
enzArray.add(manii);
enzArray.add(mgat1);
enzArray.add(mgat2);
enzArray.add(mgat3);
enzArray.add(mgat4);
enzArray.add(mgat5);
enzArray.add(galt);
enzArray.add(fucT8);
enzArray.add(siaT);

Output:
```

Pathway reconstruction using connection inference

inferGlyConnPath command is used to construct a pathway and the pathway can be visualized using ***glycanPathViewer*** command. The reconstructed network has 360 reactions and 188 species. This construction process takes about 5 mins to finish on an Intel i7 4-Core PC.

```
Input:

[isPath,nglycanpath]=inferGlyConnPath(substrateArray, enzArray);

if(isPath)
    glycanPathViewer(nglycanpath);
    fprintf(1,'the generated network has: \n');
    fprintf(1,' %i  reactions\n',nglycanpath.getNReactions);
    fprintf(1,' %i  species\n',   nglycanpath.getNSpecies);
else
    fprintf(1,'no path is found\n');
end

Output:
the round: 1
number of total species in the pathway: 6
number of total reactions in the pathway: 6
the round: 2
.
.
.
.
the generated network has:
    360  reactions
    188  species
```



14. Installation Instructions

The following sections provide step-by-step instructions for installation of the software in Windows, Linux (Ubuntu) and Mac OS. A detailed three-step procedure is provided for Windows, while instructions are more succinct for other platforms. Whereas these instructions are for 64-bit OS, installation for 32-bit versions follows similarly.

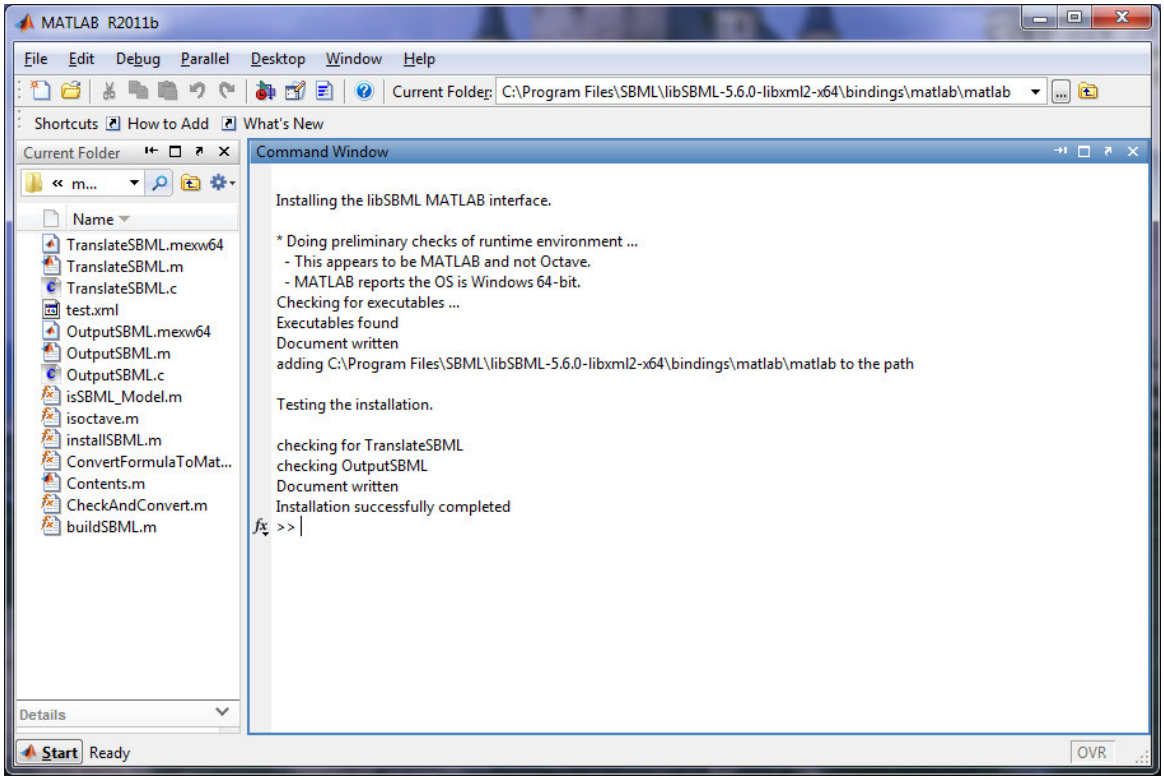
14.1 Fresh Installation of GNAT 2.0

14.1.1 Windows

STEP 1 Install libSBML (Bornstein, Keating et al. 2008) in MATLAB, if not already installed. Detailed instruction can be found at http://sbml.org/Software/libSBML. Briefly:

1) Download libSBML5.6.0 package at the link here or copy the following address into web-browser: http://sourceforge.net/projects/sbml/files/libsbml/5.6.0/stable/Windows/64-bit/libSBML-5.6.0-win-matlab-x64.exe/download

2) Double-click binary package "libSBML-5.6.0-win-matlab-x64.exe" to auto-install. This will perform all steps including the installation of libsbml interface in MATLAB (see below).



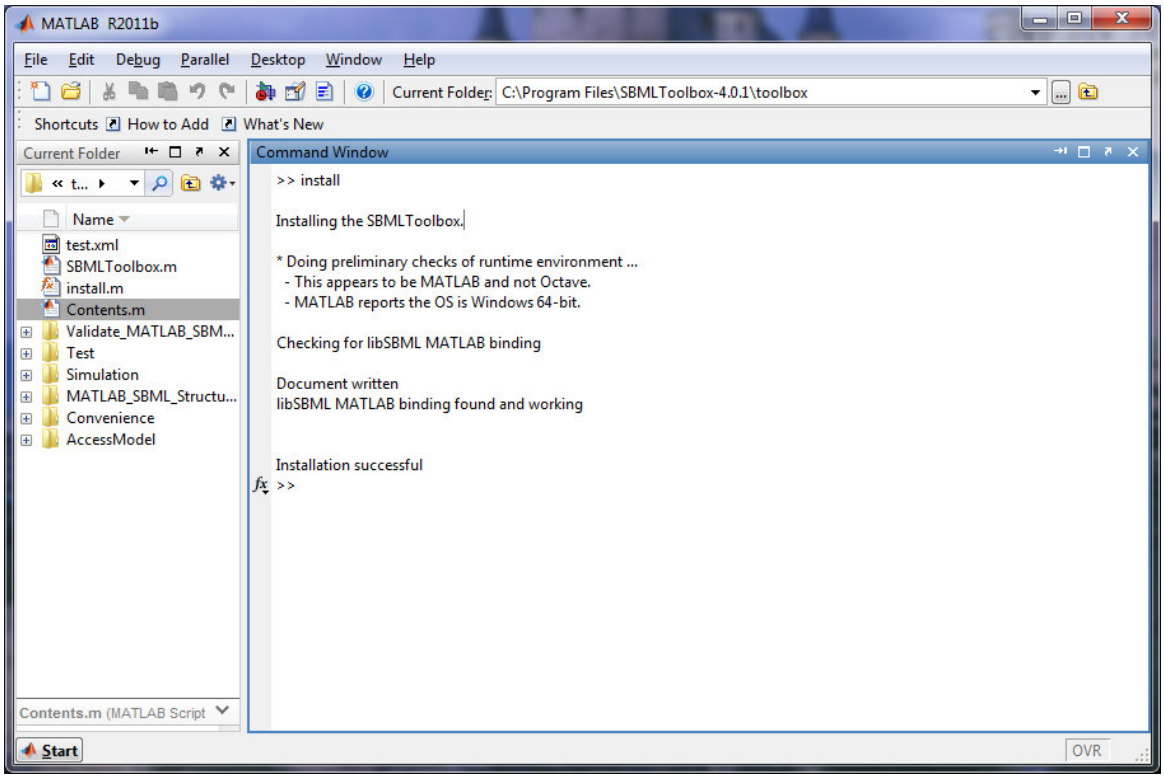
MATLAB window when libSBML is successfully installed.

STEP 2 Install SBML Toolbox (Keating, Bornstein et al. 2006) in MATLAB, if not already installed. Detailed instructions are at http://sbml.org/software/sbmltoolbox. Briefly:

1) Download SBML toolbox ("SBMLToolbox-4.1.0.zip") from site below and unzip files into user specified directory (for example: "C:\SBMLToolbox"):

   a. http://sourceforge.net/projects/sbml/files/SBMLToolbox/4.1.0/SBMLToolbox-4.1.0.zip/download

2) In MATLAB environment, change path to "<SBMLToolboxDirectory>\SBMLToolbox -4.1.0\toolbox". Type "install" in the command window and press enter to start installation. This step adds the SBML directory to the MATLAB search path and checks libSBML binding package using an example SBML file. Once the installation succeeds, the "Install successful" message will display in the MATLAB command window as shown below.



MATLAB window when SBML Toolbox is successfully installed.

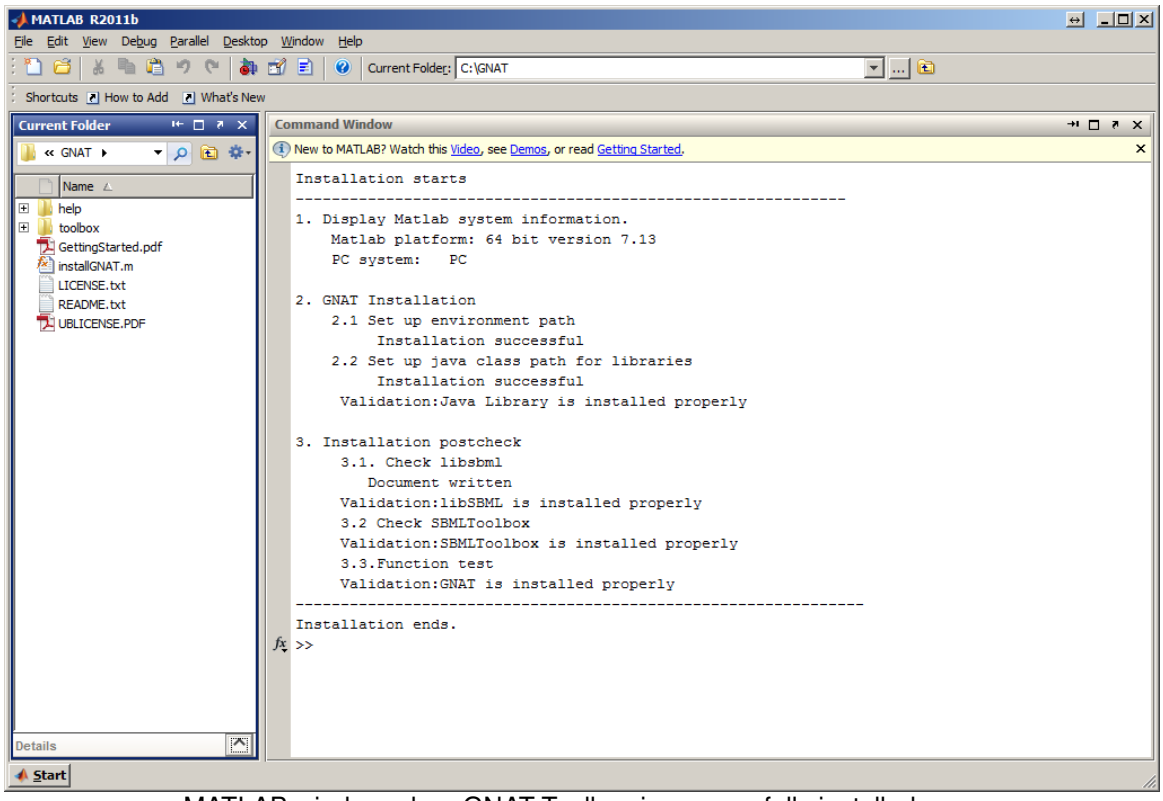
STEP 3 Installation of GNAT in MATLAB

1) Download "gnat.zip" from site below and unzip files in desired application directory (say C:\gnat): http://sourceforge.net/projects/gnatmatlab/files/. A folder ("toolbox") and additional files including the "GettingStarted.pdf", and "installGNAT.m" file will appear. "toolbox" folder contains source code, example files and java libraries.

2) Start MATLAB as administrator. MATLAB version 7.13 (R2011b) or higher is recommended. Administrator access is helpful so that the "pathdef.m" file can be edited. This operation adds new paths to MATLAB search path. To open MATLAB as

administrator, right click MATLAB icon on desktop or start menu, and choose “run as administrator”.

3) In MATLAB command window change current path to the GNAT directory (say: C:\gnat). Then, type “installGNAT” and press enter to start installation. This: i) clears all variable from the MATLAB workspace, cleans command window and displays MATLAB version and system information; ii) checks for libSBML and SBML Toolbox installation; iii) adds “toolbox” folder and its sub-directories to MATLAB search path; iv) adds GNAT java library path to MATLAB dynamic java class path; v) checks for GNAT installation directory and class path. The procedure ends with an installation status report (see below).



MATLAB window when GNAT Toolbox is successfully installed.

4) Now the user can start to use all GNAT commands in command window or write user-defined functions.

5) While GNAT is fully functional at the time of initial installation, loading of Java library in subsequent startups is not automatic unless one of two alternate steps outlined below is carried out:

Change of MATLAB java static path:

To do this, the Java library path names need to be appended to the “classpath.txt” file. In order to perform this operation:

- Type “**`edit classpath.txt`**” in MATLAB command window.

- Append the following path names to this file. Here <GNATInstallationDirectory> stands for the absolute directory path (e.g. C:\gnat).

```
<GNATInstallationDirectory>\toolbox\javalib\xmlgraphics-commons-1.0.jar
<GNATInstallationDirectory>\toolbox\javalib\resourcesdb_interfaces.jar
<GNATInstallationDirectory>\toolbox\javalib\mysql-connector-java-5.1.18-bin.jar
<GNATInstallationDirectory>\toolbox\javalib\molecular_framework.jar
<GNATInstallationDirectory>\toolbox\javalib\log4j-1.2.16.jar
<GNATInstallationDirectory>\toolbox\javalib\jsbml-0.8-rc1-with-dependencies.jar
<GNATInstallationDirectory>\toolbox\javalib\jgraphx.jar
<GNATInstallationDirectory>\toolbox\javalib\jdom-1.g0.jar
<GNATInstallationDirectory>\toolbox\javalib\jdesktop.jar
<GNATInstallationDirectory>\toolbox\javalib\flamingo.jar
<GNATInstallationDirectory>\toolbox\javalib\commons-logging-api.jar
<GNATInstallationDirectory>\toolbox\javalib\commons-logging-1.0.4.jar
<GNATInstallationDirectory>\toolbox\javalib\commons-io-1.1.jar
<GNATInstallationDirectory>\toolbox\javalib\commons-codec-1.3.jar
<GNATInstallationDirectory>\toolbox\javalib\batik.jar
<GNATInstallationDirectory>\toolbox\javalib\batik-all-1.6.jar
<GNATInstallationDirectory>\toolbox\javalib\avalon-framework-4.2.0.jar
<GNATInstallationDirectory>\toolbox\javalib\GlycanBuilder.jar
<GNATInstallationDirectory>\toolbox\javalib\GNVJAVA.jar
<GNATInstallationDirectory>\toolbox\javalib
--------------------------------------------------------------------------------
```

Change of MATLAB java static path:

Change the ‘startup.m’ file so that the dynamic Java class path is loaded at the time of MATLAB start up. This involves:

- Opening the ‘startup.m’ file located in the GNAT “\toolbox\startup” folder. Edit <GNATInstallationDIR> in the tenth line of this file {**`gnatpath='<GNATInstallationDIR>'`**} to reflect the absolute directory path (e.g. if the absolute directory path is ‘C:\gnat’; replace this line with **`gnatpath='C:\gnat'`**).
- Locate the MATLAB startup folder by typing “userpath” in the MATLAB command window, or alternatively use “userpath(newdirectory)” to setup a new MATLAB startup folder location. Copy ‘startup.m’ into the MATLAB startup folder. If a “startup.m” file already exists in this location, append the contents of the GNAT “startup.m” file into the existing “startup.m” file.

14.1.2 Linux (Ubuntu)

Installation of GNAT in Linux has been tested using Ubuntu 10.10. Ubuntu versions higher than 11.04 are **NOT** recommended since the MATLAB compiler has not been fully tested in Ubuntu 11.04 or newer versions. This is because the current MATLAB compiler supports GNU gcc/g++ compiler versions up to 4.4. Ubuntu 11.04 and higher versions ship with gcc/g++ compiler version 4.5 or higher.

Step 1: Installation of libSBML in MATLAB

In Ubuntu, it is necessary to install libSBML from source distribution i.e. a. libSBML source code needs to be compiled to a binary package. During this step, proper configuration flags need to be applied as briefly described below; b. library path must be setup. A brief stepwise procedure for libSBML installation in Ubuntu is described below. More information can be obtained from http://sbml.org/Software/libSBML/docs/cpp-api/libsbml-installation.html.

1) Download libSBML source code "libsbml-5.6.0-src.tar.gz" from: http://sourceforge.net/projects/sbml/files/libsbml/5.6.0/stable/libsbml-5.6.0-src.tar.gz/download.

   a. Open a terminal and extract files to user-specified directory (say: /usr/local/libsbml) using the command "**tar –xzvf libsbml-5.6.0-src.tar.gz –C /user/desired/path**". Enter the directory using "**cd /user/desired/path**" command. Run the command below to build the package preconfigured with MATLAB:

2) "**`./configure --with-matlab=<MatlabInstallationPath> --prefix=/usr/local/ --enable-m64`**"

   a. Execute three more commands at the terminal prompt:

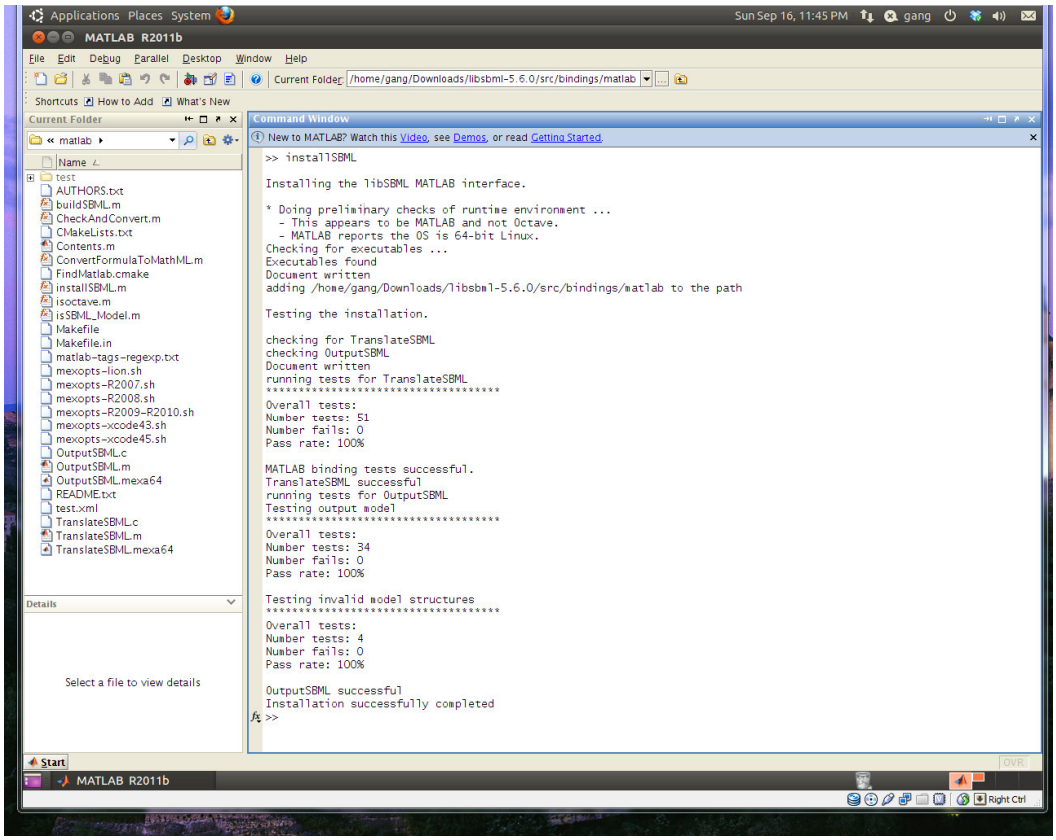
   I. "**make**"

   II. "**sudo make install**"

   III. "**sudo ldconfig**"

3) For the purpose of binding libSBML to MATLAB, it is necessary to set LD_LIBRARY_PATH in the system. In Ubuntu, this can be implemented by adding "**export LD_LIBRARY_PATH=/usr/lib64:/usr/local/lib**" to .bash_profile in the home folder (/home/<usrname>).

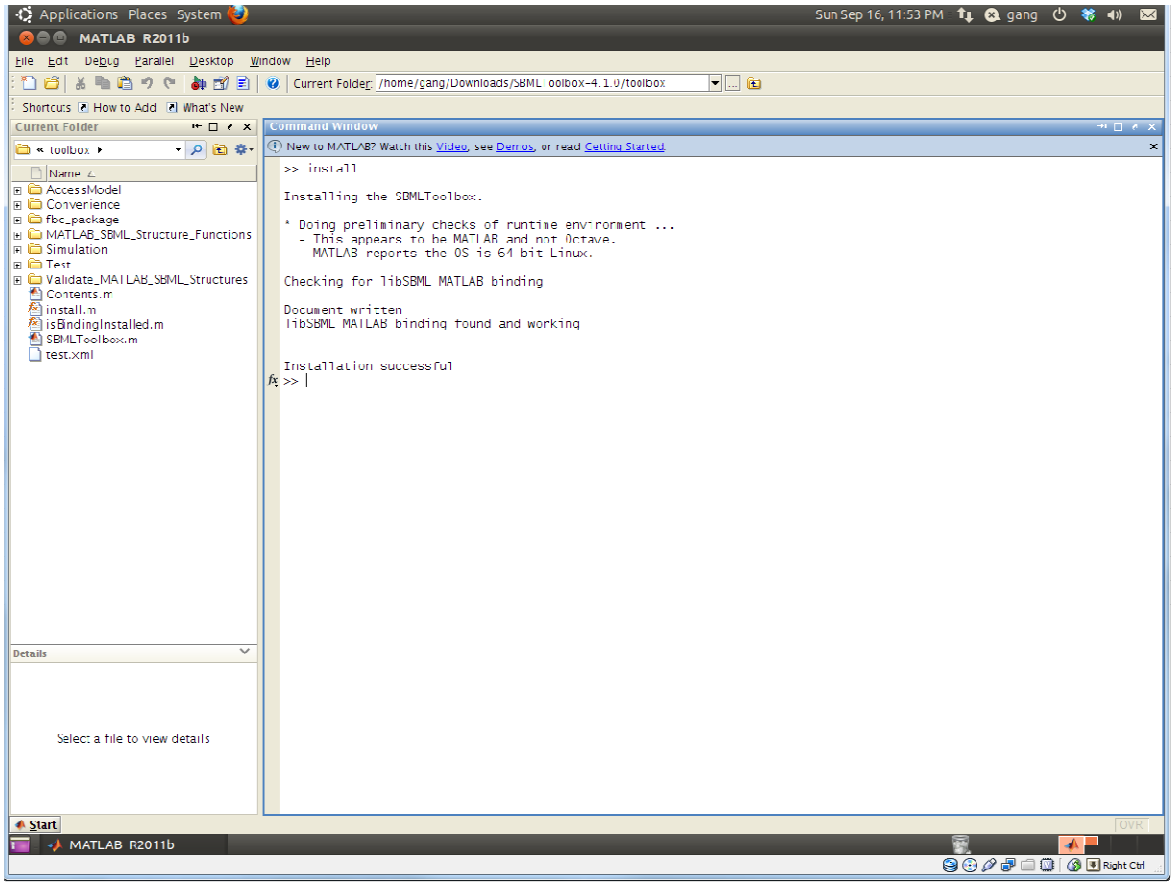
4) Navigate to <libsbmlInstallDirectory>/src/bindings/matlab in MATLAB, and type installSBML.m to create the interface between libSBML and MATLAB. After this run, libSBML is installed successfully and the command window will display installation status as shown below.



MATLAB window when libSBML is successfully installed in Ubuntu

D. STEP 2 Install SBML Toolbox in MATLAB.

This installation is similar to Windows, i.e., first download SBML Toolbox package and then install it in MATLAB. Since SBML Toolbox is platform-independent, the package to download is same as in Windows ("SBMLToolbox-4.1.0.zip"). After running command "**install**" in MATLAB, the installation status will show up in command window as below.

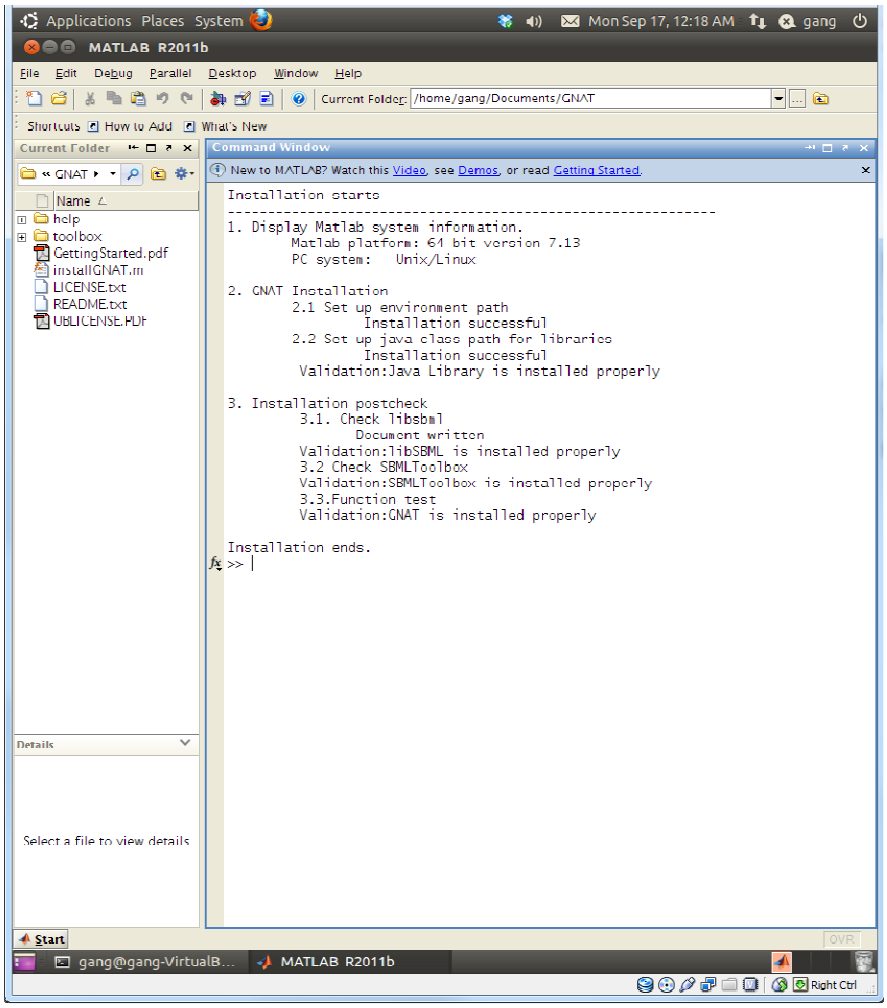


MATLAB window when SBML Toolbox is successfully installed in Ubuntu.

Step 3: Installation of GNAT in MATLAB

1) Start MATLAB. To do this in Ubuntu, open a terminal, enter the directory where MATLAB executable file is installed, and type "sudo ./matlab" to start MATLAB.

2) GNAT installation steps are similar to Windows. Download GNAT platform-independent package ("gnat.zip") and install it in MATLAB. After running the "**installGNAT**" command in MATLAB, the installation status will appear in command window (see below). After installation, follow similar steps as in Windows to set up Java class path. Please note that the path separator in Linux should be "/", rather than "\" in Windows.



MATLAB window when GNAT is successfully installed in Ubuntu.

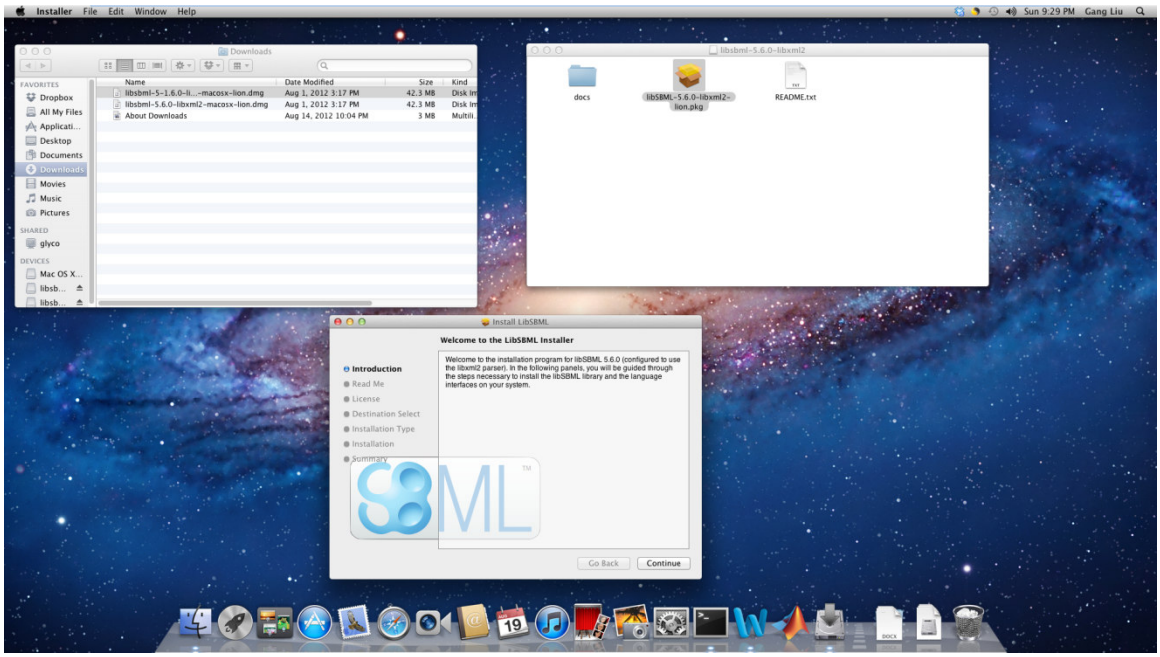
14.1.3 Mac OS

Mac OS installation of GNAT follows similar procedures as Windows. We tested the package in Mac OS X Lion (10.7).

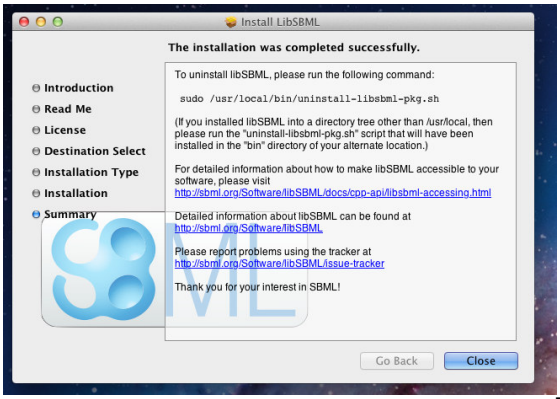
STEP 1 Install libSBML in MATLAB.

1) Download the dmg (Apple Disk Image) file "libsbml-5.6.0-macosx-lion.dmg" from http://sourceforge.net/projects/sbml/files/libsbml/5.6.0/stable/Mac%20OS%20X/.

2) Double click the "libsbml-5.6.0-macosx-lion.dmg" file. The package file will show up in a new window. Double clicking the .pkg file to install libSBML. The default libSBML installation directory is /usr/local/. The user can change this directory during installation. Screen shots of the installation windows are shown below.



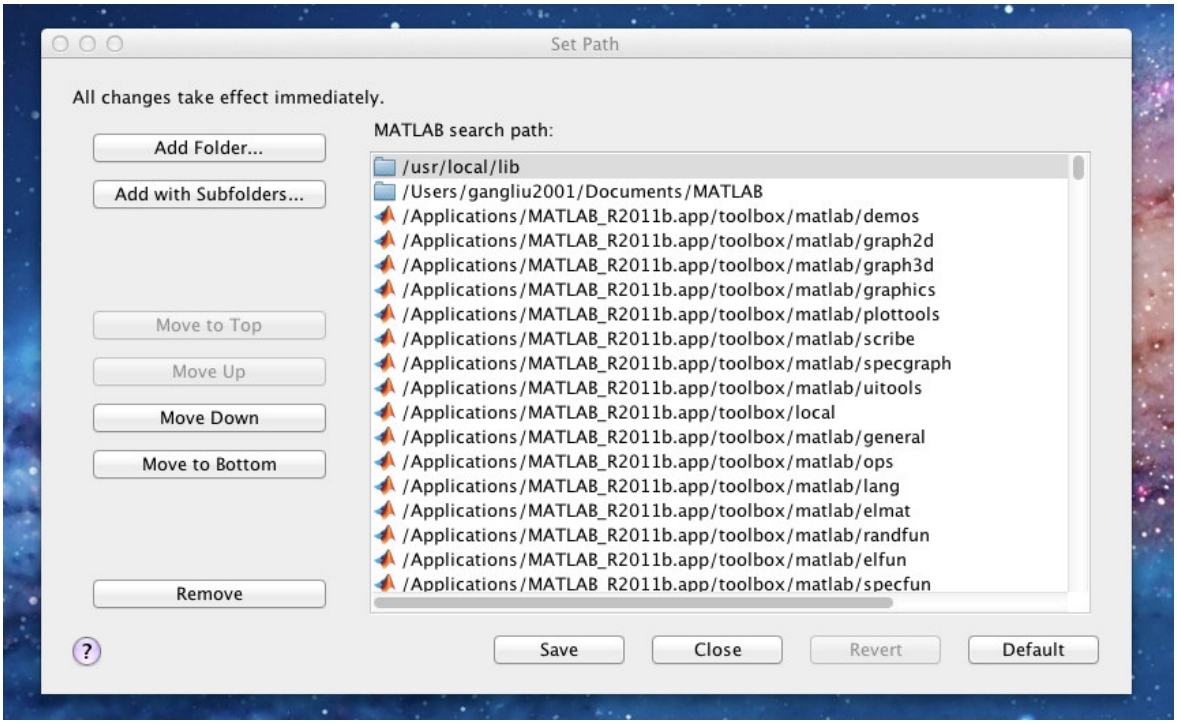
Installation window when libSBML is starting to be installed in Mac OS X Lion



Installation window when libSBML is installed successfully in Mac OS X Lion

3) To setup the interface between libSBML and MATLAB, user should add the libSBML library path (default: /usr/local/lib) to MATLAB search path. To do this:

1) Open MATLAB and choose "Set Path" window either from the 'File' pull-down menu (for MATLAB release prior to 2012b) or "Home" tab (MATLAB 2012b onwards). Select the "Add folder" option and browse to select the libsbml library directory (e.g. /usr/local/lib). Save the new path/settings, if possible.

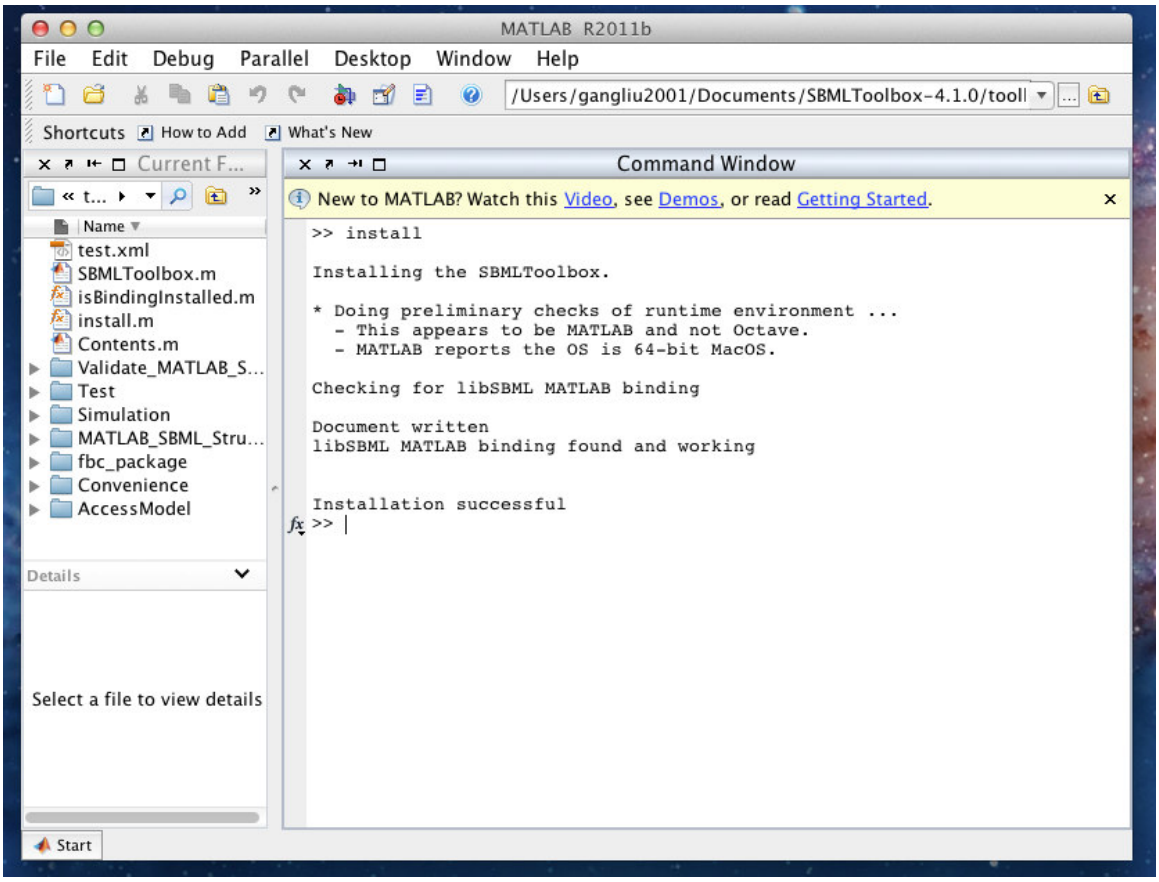


2) Alternatively use MATLAB command line operation. Type the following command in command window and then press enter.

```
addpath('/usr/local/lib'); savepath;
```

STEP 2 Install SBMLToolbox in MATLAB.

1) Installation instruction is similar to Windows. Extract "SBMLToolbox-4.1.0.zip" to specified directory. Go to this installation directory and then type "**install**". Installation message will be shown in MATLAB command window as shown below.



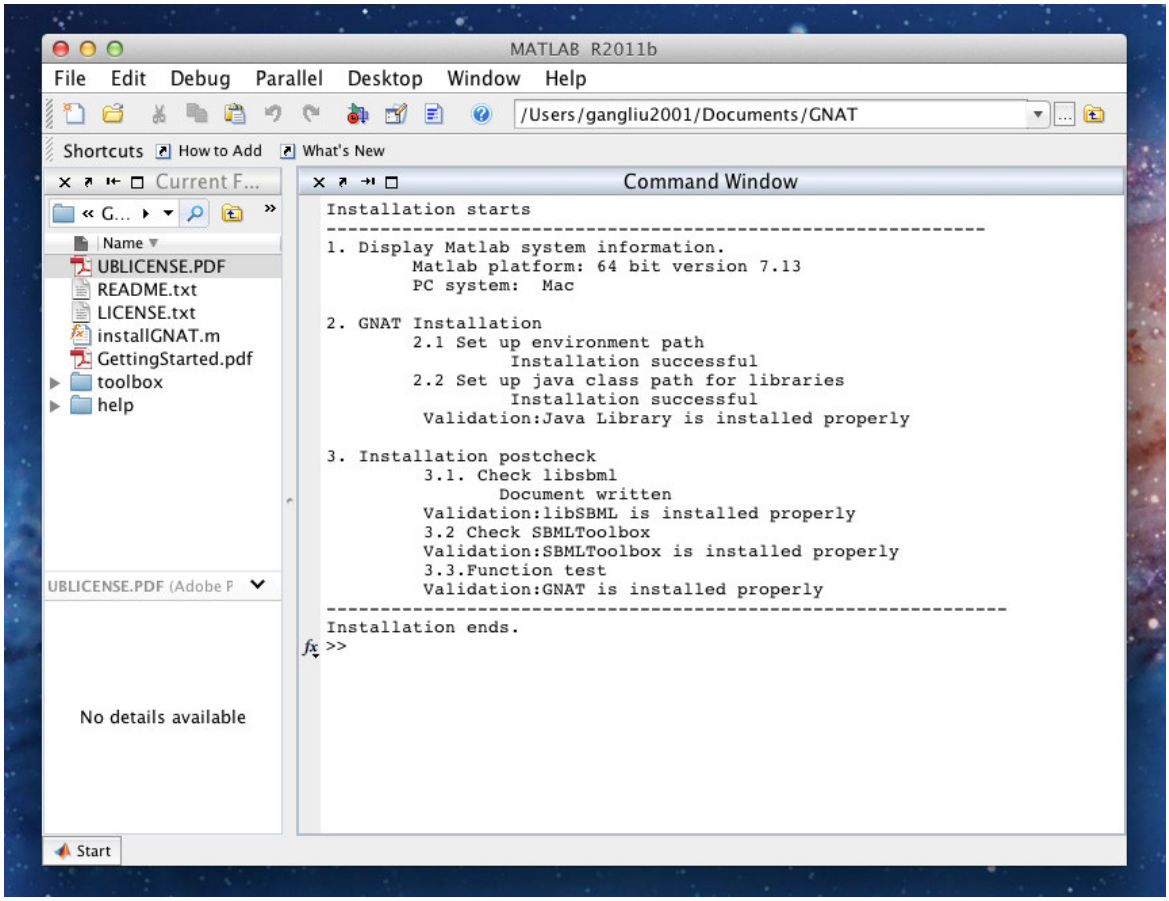
Installation window when SBML Toolbox is installed successfully in Mac OS X Lion

STEP 3 Install GNAT in MATLAB

1) Start MATLAB. Click "Launchpad" in Dock and double click the "MATLAB" icon. Alternatively, use "Spotlight" to find MATLAB application.

2) Installation instruction is similar to Windows. Extract "gnat.zip" to user-specified directory, go to the installation directory and then run "**`installGNAT`**" command. The installation message will be shown in MATLAB command window as below.

3) After installation, follow similar steps as in Windows to set up Java class path. Please note that the path separator in Mac OS should be "/", rather than "\" in Windows.



Installation window when GNAT is installed successfully in Mac OS X Lion

14.1.4 Installation of local database (optional)

Two of the database-associated functions of GNAT require implementation of the **`dbLocalConnect`** and **`dbExactStructSearch`** commands in MATLAB. This allows access between MATLAB and various Glycomics databases. Implementation of these two commands requires installation of the *MATLAB Database Toolbox* and a *local database server*:

1. MATLAB Database Toolbox Installation

a. The user can follow the instructions provided by MathWorks. The toolbox may have to be purchased separately in some cases.

2. Building a local web server and loading GlycomeDB database

a. Detailed stepwise instructions for the installation of the PostgresSQL server in the local machine are provided at http://wiki.postgresql.org/wiki/Detailed_installation_guides. The user should follow these instructions to install PostgresSQL. This is a free program.

b. The user should download a GlycomeDB database dump file from the GlycomeDB website and load the database file into the local web server. The detailed instruction can be seen at http://www.glycome-db.org/getDownloadPage.action?page=glycomedb.

14.2 Upgrading from GNAT1.0 to 2.0

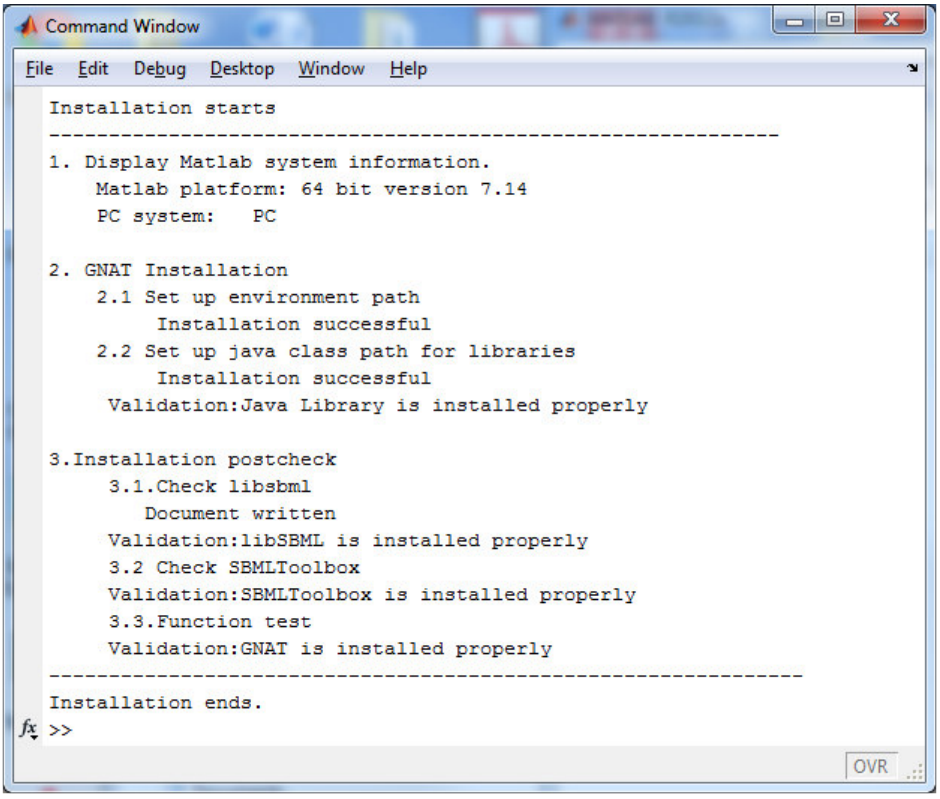
In order to remove a previous version of GNAT (i.e. GNAT1.0) and to replace it with the new version (i.e. GNAT2.0), perform the following:

1) Delete the contents of the previous GNAT installation folder, say C:\GNAT.

2) Unzip the new version of GNAT (GNAT2.0) at this location, i.e. place the installGNAT.m file and toolbox folder of GNAT2.0 at this location.

3) Open MATLAB, change the current directory to C:\GNAT, and at the command prompt type "installGNAT".

The following window shows the MATLAB output during installation (for PC). The installation assumes that libSBML and SBMLToolbox are already installed on the computer, and the user initiates MATLAB "as administrator". Please follow more detailed installation instructions in section 14.1, if necessary.



15. FAQ regarding GNAT

What versions of libSBML and SBMLToolbox are compatible with GNAT?

GNAT has been tested using libSBML5.6.0 and SBMLToolbox 4.1.0. Other versions of the toolboxes may be downloaded from http://sbml.org/Software/libSBML and http://sbml.org/software/sbmltoolbox. Software compatibility has not been tested with all versions of libSBML and SBML Toolbox.

Is GNAT compatible with 32-bit systems?

Our programs have been developed with the viewpoint that most of the newer computers are 64-bit systems. Thus, the 64-bit Java library has been provided in GNAT.

Limited testing has been performed on 32-bit computers and issues were not noted. However, in this case, it is better to use the 32-bit Java compiler for older computers. GNAT developers can assist with this as necessary.

What versions of MATLAB support GNAT?

MATLAB version 7.13 (R2011b) or higher is recommended.

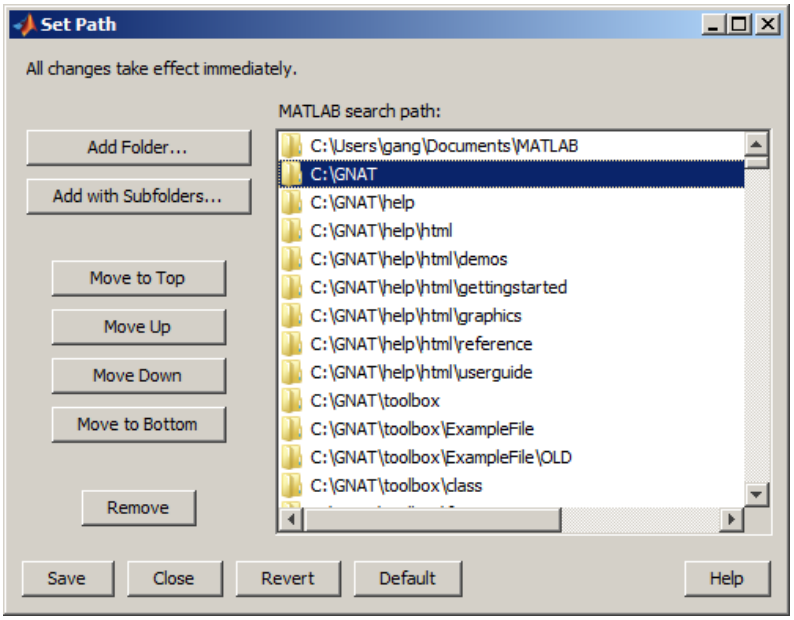
What will happen if I am not computer administrator when GNAT is installed?

GNAT will install properly at the time of first installation. However, problems may be encountered upon restarting MATLAB since MATLAB search path has not been saved. Errors may thus appear stating that "file cannot be found" or "function cannot be found".

Two alternative methods can be used to solve the issue:

A. Go to GNAT installation directory and rerun "installGNAT" command again.

B. Choose "Set Path" window from the 'File' pull-down menu in MATLAB (see below). Select the "Add with Subfolders…" option and browse to select the <User-defined directory> directory (e.g. C:\GNAT"). Save the new path/settings, if possible.



What do I do if Ubuntu libraries are missing during installation of libSBML?

Some libraries required to build libSBML may be missing from Ubuntu libraries (e.g. xml2-config). In this case, it may be necessary to download the missing libraries. This is done using the command "**sudo apt-get install <packagename>** (e.g., libxml2.dev)".

What do I do if I get an out of memory error?

Depending on the computer memory and the number of variables stored, it is possible to run out of RAM space. In this case run the commands:

```
clear;     % to cleanup MATLAB workspace
clear java% to cleanup JAVA variables in the JAVE heap space

Also got to 'Java Heap Memory Preferences' under 'MATLAB Preferences' and increase value
to avoid "OutOfMemoryError: Jave heap space" errors.
```

If the above does not work then follow the link for more information or restart MATLAB:
To increase java heap space:
http://www.mathworks.com/support/solutions/en/data/1-18I2C/
To address other out of memory problems:
http://www.mathworks.com/help/matlab/matlab_prog/resolving-out-of-memory-errors.html

Why are some of the fields in the Enz class not populated from the IUBMB database?

This typically happens because the computer is not connected to the internet, or the internet connectivity is too slow or the EC number does not have a match. Errors associated with this step have been suppressed in GNAT.

16 Acknowledgements

We gratefully acknowledge the SBML team and the groups from University of Hertfordshire, Hatfield, UK (Keating S.M. and colleagues), for graciously providing libSBML and SBML Toolbox for the benefit of the community.

We also thank the groups from the Imperial College London for providing the robust Java library GlycanBuilder (Ceroni, Dell et al. 2007) for glycan visualization.

We thank JGraph Ltd. for providing “mxGraph”, a Java library for high-quality visualization of networks (http://www.jgraph.com/), for non-commercial use.

17. References

Apte, A. and N. S. Meitei (2010). "Bioinformatics in glycomics: glycan characterization with mass spectrometric data using SimGlycan." Methods Mol Biol 600: 269-281.

Bohne-Lang, A., E. Lang, et al. (2001). "LINUCS: linear notation for unique description of carbohydrate sequences." Carbohydr Res 336(1): 1-11.

Bornstein, B. J., S. M. Keating, et al. (2008). "LibSBML: an API library for SBML." Bioinformatics 24(6): 880-881.

Ceroni, A., A. Dell, et al. (2007). "The GlycanBuilder: a fast, intuitive and flexible software tool for building and displaying glycan structures." Source Code Biol Med 2: 3.

Ceroni, A., K. Maass, et al. (2008). "GlycoWorkbench: a tool for the computer-assisted annotation of mass spectra of glycans." J Proteome Res 7(4): 1650-1659.

Herget, S., R. Ranzinger, et al. (2008). "GlycoCT-a unifying sequence format for carbohydrates." Carbohydr Res 343(12): 2162-2171.

Hoops, S., S. Sahle, et al. (2006). "COPASI--a COmplex PAthway SImulator." Bioinformatics 22(24): 3067-3074.

Hucka, M., A. Finney, et al. (2003). "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models." Bioinformatics 19(4): 524-531.

Keating, S. M., B. J. Bornstein, et al. (2006). "SBMLToolbox: an SBML toolbox for MATLAB users." Bioinformatics 22(10): 1275-1277.

Raman, R., M. Venkataraman, et al. (2006). "Advancing glycomics: implementation strategies at the consortium for functional glycomics." Glycobiology 16(5): 82R-90R.

Ranzinger, R., S. Herget, et al. (2011). "GlycomeDB--a unified database for carbohydrate structures." Nucleic Acids Res 39(Database issue): D373-376.

Sahoo, S. S., C. Thomas, et al. (2005). "GLYDE-an expressive XML standard for the representation of glycan structure." Carbohydr Res 340(18): 2802-2807.

Takahashi, K., N. Ishikawa, et al. (2003). "E-Cell 2: multi-platform E-Cell simulation system." Bioinformatics 19(13): 1727-1729.

Umana, P. and J. E. Bailey (1997). "A mathematical model of N-linked glycoform biosynthesis." Biotechnol Bioeng 55(6): 890-908.

Apte, A. and N. S. Meitei (2010). "Bioinformatics in glycomics: glycan characterization with mass spectrometric data using SimGlycan." Methods Mol Biol 600: 269-281.

Bohne-Lang, A., E. Lang, et al. (2001). "LINUCS: linear notation for unique description of carbohydrate sequences." Carbohydr Res 336(1): 1-11.

Bornstein, B. J., S. M. Keating, et al. (2008). "LibSBML: an API library for SBML." Bioinformatics 24(6): 880-881.

Ceroni, A., A. Dell, et al. (2007). "The GlycanBuilder: a fast, intuitive and flexible software tool for building and displaying glycan structures." Source Code Biol Med 2: 3.

Ceroni, A., K. Maass, et al. (2008). "GlycoWorkbench: a tool for the computer-assisted annotation of mass spectra of glycans." J Proteome Res 7(4): 1650-1659.

Herget, S., R. Ranzinger, et al. (2008). "GlycoCT-a unifying sequence format for carbohydrates." Carbohydr Res 343(12): 2162-2171.

Hoops, S., S. Sahle, et al. (2006). "COPASI--a COmplex PAthway SImulator." Bioinformatics 22(24): 3067-3074.

Hucka, M., A. Finney, et al. (2003). "The systems biology markup language (SBML): a

medium for representation and exchange of biochemical network models." Bioinformatics 19(4): 524-531.
Keating, S. M., B. J. Bornstein, et al. (2006). "SBMLToolbox: an SBML toolbox for MATLAB users." Bioinformatics 22(10): 1275-1277.
Liu, G., D. D. Marathe, et al. (2008). "Systems-level modeling of cellular glycosylation reaction networks: O-linked glycan formation on natural selectin ligands." Bioinformatics 24(23): 2740-2747.
Liu, G., A. Puri, et al. (2013). "Glycosylation Network Analysis Toolbox: a MATLAB-based environment for systems glycobiology." Bioinformatics 29(3): 404-406.
Neelamegham, S. and G. Liu (2011). "Systems glycobiology: biochemical reaction networks regulating glycan structure and function." Glycobiology 21(12): 1541-1553.
North, S. J., H. H. Huang, et al. (2010). "Glycomics profiling of Chinese hamster ovary cell glycosylation mutants reveals N-glycans of a novel size and complexity." J Biol Chem 285(8): 5759-5775.
Raman, R., M. Venkataraman, et al. (2006). "Advancing glycomics: implementation strategies at the consortium for functional glycomics." Glycobiology 16(5): 82R-90R.
Ranzinger, R., S. Herget, et al. (2011). "GlycomeDB--a unified database for carbohydrate structures." Nucleic Acids Res 39(Database issue): D373-376.
Sahoo, S. S., C. Thomas, et al. (2005). "GLYDE-an expressive XML standard for the representation of glycan structure." Carbohydr Res 340(18): 2802-2807.
Takahashi, K., N. Ishikawa, et al. (2003). "E-Cell 2: multi-platform E-Cell simulation system." Bioinformatics 19(13): 1727-1729.
Umana, P. and J. E. Bailey (1997). "A mathematical model of N-linked glycoform biosynthesis." Biotechnol Bioeng 55(6): 890-908.