rBiopaxParser Short Vignette

25th August 2012

Frank Kramer <dev@frankkramer.de>

The aim of this document is to help the user get accustomed with the package and to provide a step-by-step introduction on how to get started. For a deeper understanding of how BioPAX instances are composed, it is strongly encouraged to take a look at the BioPAX definition, especially the class inheritance tree and the list of properties for each class. The language definition, as well as further information on BioPAX, can be found at http://www.biopax.org.

This vignette contains installation instructions as well as a quick listing of working code to get started with the package right away.

Index

0 BioPAX

A plethora of databases offer a vast knowledge about biological signaling pathways. BioPAX is implemented in the Web Ontology Language OWL, an RDF/XML-based markup language. It allows the users to store and exchange pathway knowledge in a well-documented and standardized way. In simplified terms one can say, that the main class, the pathway, is build up from a list of interactions. Interactions themselves provide a link from one controlling molecule to one or more controlled molecules. Molecule instances, including their properties like names, sequences or external references are defined within the BioPAX model. This package will hopefully ease the task of working with BioPAX data within R.

1 Installing the package

1.1 Prerequisites

This package depends on package ***XML*** to parse the BioPAX .owl files.
This package suggests package ***RCurl*** to download BioPAX files from the web.
This package suggests package ***graph*** to build graphs/networks from the data.
This package suggests package ***Rgraphviz*** to visualize networks.
To install directly from github you need package ***devtools***.

Installation or running certain functions MIGHT fail if these prerequisites are not met. Please read through the following instructions.

1.1.1 Prerequisites for Linux users

This paragraph uses installation instructions fitting for Debian and Ubuntu derivatives. If you are on another Linux please use the corresponding functions of your distribution.

XML:
Make sure your Linux has library *libxml2* installed. This is almost always the case. Otherwise install libxml2:

```
sudo apt-get install libxml2
```

You will now be able to install R package XML, this should be automatically done when you install rBiopaxParser, or you can run within R:

```
install.packages("XML")
```

RCurl:

RCurl is only needed for a convenience function to download BioPAX files directly within R. You can skip this step if you already have the BioPAX data downloaded.
Make sure your Linux has library *libcurl* installed and curl-config in your path. Check out

```
locate libcurl
locate curl-config
```

If these are not found (usually the developer version is missing), most Linux users will be able to fix this by running

```
sudo apt-get install libcurl4-openssl-dev
```

You will now be able to install R package RCurl, this should be automatically done when you install rBiopaxParser, or you can run within R:

```
install.packages("RCurl")
```

If you encounter other problems check out
http://www.omegahat.org/RCurl/FAQ.html

graph:
Package graph has moved from CRAN to Bioconductor recently, you might encounter an error saying that package graph is not available for your distribution when calling install.packages("graph").
Check out http://bioconductor.org/packages/release/bioc/html/graph.html or call

```
source("http://bioconductor.org/biocLite.R")
biocLite("graph")
```

to install it right away.

Rgraphviz:
Rgraphviz is used to layout the graphs generated in this package. You can layout and plot these yourself if you want to.
Make sure your Linux has package **graphviz** installed.
If this is not the case, you can usually fix this by running

```
sudo apt-get install graphviz
```

You will now be able to install R package Rgraphviz using:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")
```

If you encounter more problems check out
http://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html

devtools:
Package devtools is available at CRAN. Run

```
install.packages("devtools")
```

to install it.

1.1.2 Prerequisites for Windows users

XML & RCurl:
These packages depend on Linux libraries. However, Brian Ripley has put together a repository to allow Windows users to run these packages.
Check out http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/ for these two packages for your R version.
Download first *XML.<yourRversion>.zip* and then *RCurl.<yourRversion>.zip* and install them locally on your machine.

graph:
Package graph has moved from CRAN to Bioconductor recently, you might encounter an error saying that package graph is not available for your distribution when calling install.packages("graph").
Check out http://bioconductor.org/packages/release/bioc/html/graph.html or run

```
source("http://bioconductor.org/biocLite.R")
biocLite("graph")
```

to install it.

Rgraphviz:
Rgraphviz is used to layout the graphs generated in this package. You can layout and plot these yourself if you want to.
Make sure your machine has **graphviz** installed, it can be found at:
http://www.graphviz.org
Click on Download -> Windows. After installing graphviz you will now be able to install R package Rgraphviz using:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")
```

If you encounter more problems check out
http://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html

devtools:
Package devtools is available at CRAN. For Windows this seems to depend on having ***Rtools*** for Windows installed. You can download and install this from:
http://cran.r-project.org/bin/windows/Rtools/
To install R package devtools call

```
install.packages("devtools")
```

1.2 Installation

If everything went well you will be able to install the rBiopaxParser package from:

https://github.com/frankkramer/rBiopaxParser

OR install directly using the devtools package:

```
library(devtools)
install_github(repo="rBiopaxParser", username="frankkramer")
```

2 Download BioPAX Data

Many online pathway databases offer an export in BioPAX format. This package gives the user a shortcut to download BioPAX exports directly from database providers from the web. A list of links to commonly used databases is stored internally and the user can select from which source and which export to download.
The data is stored in the working directory.
Currently only the NCI website is linked, with exports of the Pathway Interaction Database (PID), BioCarta and Reactome available.
The following command downloads the BioCarta export from the NCI website.

```
file = downloadBiopaxData("NCI","biocarta")
```

After the download is finished the on-screen output informs the user of success and name of the downloaded file.

3 Parse BioPAX Data

BioPAX data can be parsed into R using the rBiopaxParser. The *readBiopax* function reads in a BioPAX .owl file and generates the internal *data.frame* format used in this package. This function can take a while with large BioPAX files like NCIs Pathway Interaction Database or Reactome.

The following command reads in the BioPAX file which was previously downloaded into variable biopax.

```
biopax = readBiopax(file)
```

A summary is automatically generated and can be accessed calling

```
biopax$summary
```

4 Internal Data Model

The BioPAX ontology models biological pathway concepts and their relationships. Implemented in the Web Ontology Language OWL, an XML-based markup language, it allows the users to store and exchange pathway knowledge in a well-documented and standardized way. In simplified terms one can say, that the main class, the pathway, is build up from a list of interactions. Interactions themselves are linking a controlling molecule to a controlled molecule.

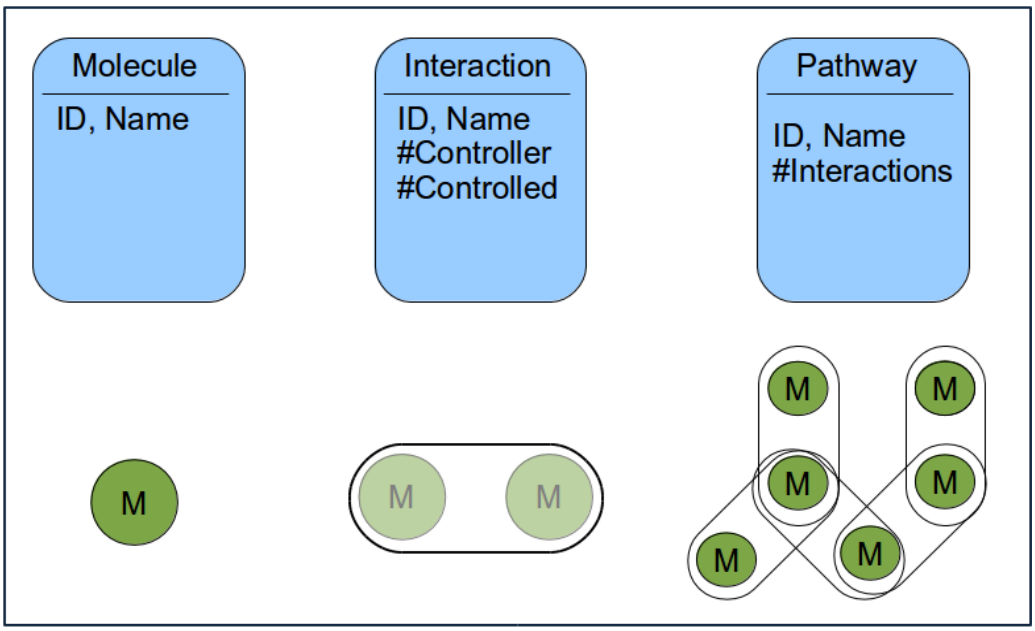


Figure 1: The building blocks for every BioPAX model: molecules, interactions and pathways.

The BioPAX ontology models the domain of biological pathway knowledge. Classes like Protein, RNA, Interaction and Pathway, define the entities in this domain. Their respective properties, like NAME, SEQUENCE, CONTROLLER and PATHWAY-COMPONENT, define the characteristics of and the links between the instances of these classes. A detailed description of BioPAX can be found at www.biopax.org. The BioPAX ontology is constantly being revised and improved. The latest released version of the ontology is BioPAX Level 3. This package currently only supports BioPAX Level 2, which is still being used by most online databases. Support for BioPAX Level 3 is planned in a future release. An overview of the classes in BioPAX Level 2 is shown in the following figure:

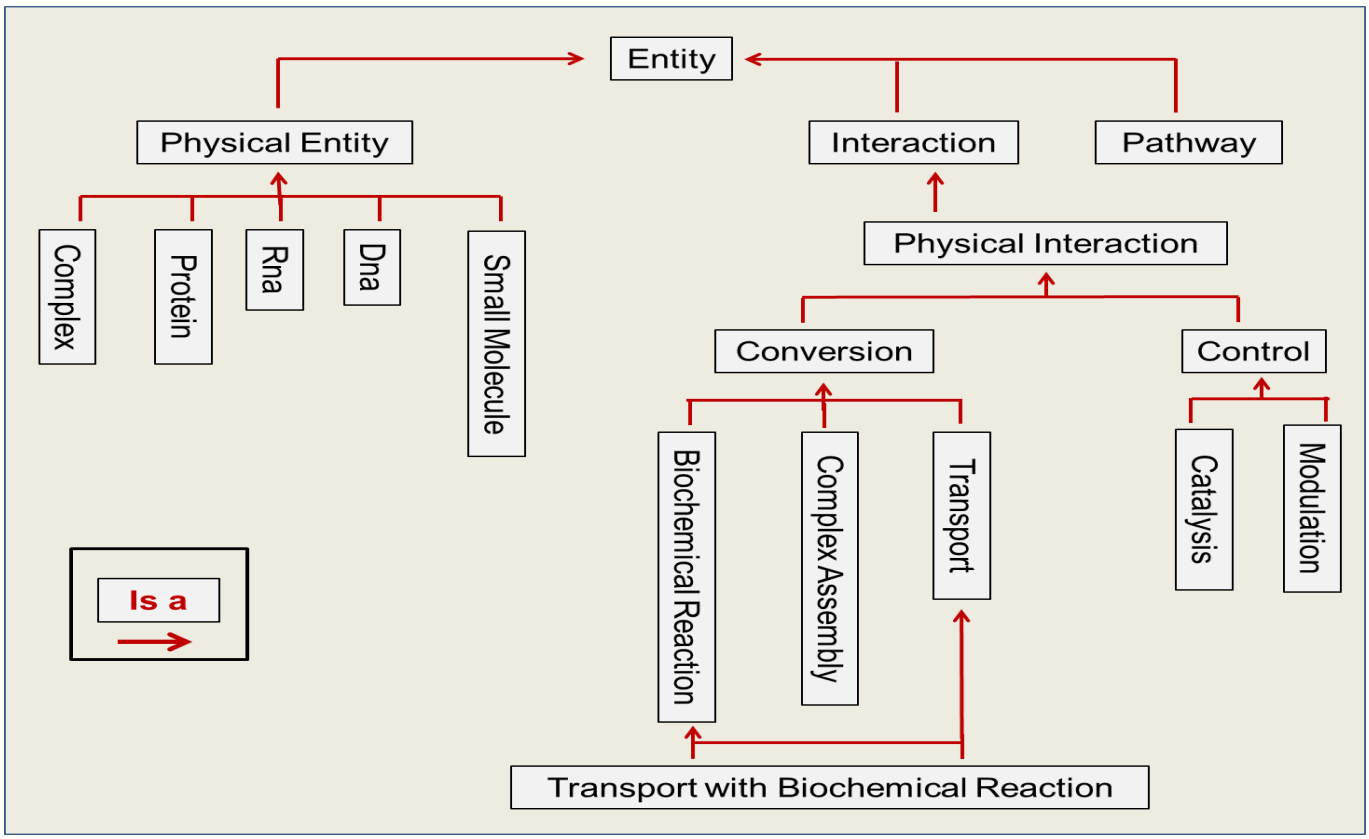


Figure 2: Class inheritance graph of the BioPAX ontology Level 2.

Mapping the XML/RDF representation of the BioPAX data from the OWL file to R is a work intensive task, especially considering the size of many complete exports of popular databases. The Pathway Interaction Database of the NCI consists of more than 50000 BioPAX instances, for example. Unfortunately mapping these instances to S3 or S4 classes within R and managing them within *lists* is not feasible, therefore the classes and their respective properties are internally mapped to a single R *matrix* and then converted to a *data.frame*. This allows for efficient indexing and selecting of subsets of this *data.frame*.

The mapping of BioPAX data is performed as revertible as possible, with one caveat, however. The XML structure of the data would allow for an infinite nesting of instance declarations. An example would be to instantiate an external publication reference within a protein instance, which could itself be instantiated in another instance. This is not desirable when attempting to map the data to a tabular format like *data.frame*. The trick here is to move these instances into the main XML tree and reference the specific instance with an rdf:resource attribute.

An excerpt of the internal *data.frame* of a biopax model, as created in the last section of this document “Modifying BioPAX”:

instancetype	instanceid	property	property_attr	property_attr_value	property_value
pathway	mypwid2	NAME	rdf:datatype	http://www.w3.org/2001/XMLSchema#string	pathway1
pathway	mypwid2	PATHWAY-COMPONENTS	rdf:resource	#control_1	
pathway	mypwid2	PATHWAY-COMPONENTS	rdf:resource	#control_2	
control	control_1	CONTROL-TYPE	rdf:datatype	http://www.w3.org/2001/XMLSchema#string	ACTIVATION
control	control_1	CONTROLLER	rdf:resource	#myPEPid_A	
control	control_1	CONTROLLED	rdf:resource	#myBCRid_B	
control	control_2	CONTROL-TYPE	rdf:datatype	http://www.w3.org/2001/XMLSchema#string	INHIBITION
control	control_2	CONTROLLER	rdf:resource	#myPEPid_A	
control	control_2	CONTROLLED	rdf:resource	#myBCRid_C	

This *data.frame* represents instances as a collection of their properties. The first column specifies the class and the second column specifies the id of the instance. The properties, for example "NAME", can either be of rdf:datatype, usually a string like "pathway1", or of type rdf:resource, which is a reference to another instance, like "#control_1".
This *data.frame* can be accessed directly via the slot "df" of the parsed object, e.g. by accessing

```
head(biopax$df)
```

5 Accessing the internal representation

Many convenience functions are available that will aid the user in selecting certain parts or instances of the biopax model. Generally, these functions will require the parsed biopax model as parameter as well as other parameters that differ from function to function.

The most basic functions to select distinct instance are *getBiopaxInstancesByID*, *getBiopaxInstancesByName*, *getBiopaxInstancesByType*. All of these do exactly as the name suggest and are also vectorized to allow the user to select multiple instances.
These functions return data.frames according to the internal data model.

The next type of selecting functions are (a) *getBiopaxInstancesList*, (b) *getPathwayList*, (c) *getPathwayComponentList* which return a vector of IDs and names of either all instances, all pathways or of all pathway components respectively.

The functions *getPathways* and *getPathway* return either all, or distinctive, pathways of a biopax model, whereas *getPathwayComponents* returns all pathway component ids of a pathway.

The functions *getReferencedIDs* and *getReferencedInstances*, which can optionally both be called recursively, are passed a biopax model and an instance ID.
The return value is either the IDs or the complete instances of all instances that are referenced by the instance supplied.

This example retrieves a list of all pathways within a BioPAX model, selects two of them and retrieves their data, their component lists and components.

```
pw_list = getPathwayList(biopax)
pw_complete = getPathways(biopax)

pwid1 = "pid_p_100002_wntpathway"     #wnt
pwid2 = "pid_p_100146_hespathway"     #segmentation clock

getInstanceProperty(biopax,pwid1,property="NAME")
getInstanceProperty(biopax,pwid2,property="NAME")

pw_1  = getPathway(biopax,pwid1)
pw_1_component_list = getPathwayComponentList(biopax,pwid1)
pw_1_components = getPathwayComponents(biopax,pwid1)
pw_2  = getPathway(biopax,pwid2)
pw_2_component_list = getPathwayComponentList(biopax,pwid2)
pw_2_components = getPathwayComponents(biopax,pwid2)
```

6 Visualization

These functions transform BioPAX pathways into regulatory graphs. However, there are some caveats. These graphs rely solely on the BioPAX information about activations and inhibitions, by classes of, or inheriting from, class “control”. Involved molecules, as nodes, are connected, via edges, depending on this information. Lack of this information will inevitably lead to disconnected or incomplete graphs. The splitComplexMolecules parameter is available to split all complexes into their most atomic members, all members will share the same in- and outgoing edges.

Transform pathways into a regulatory graph or an adjacency matrix:

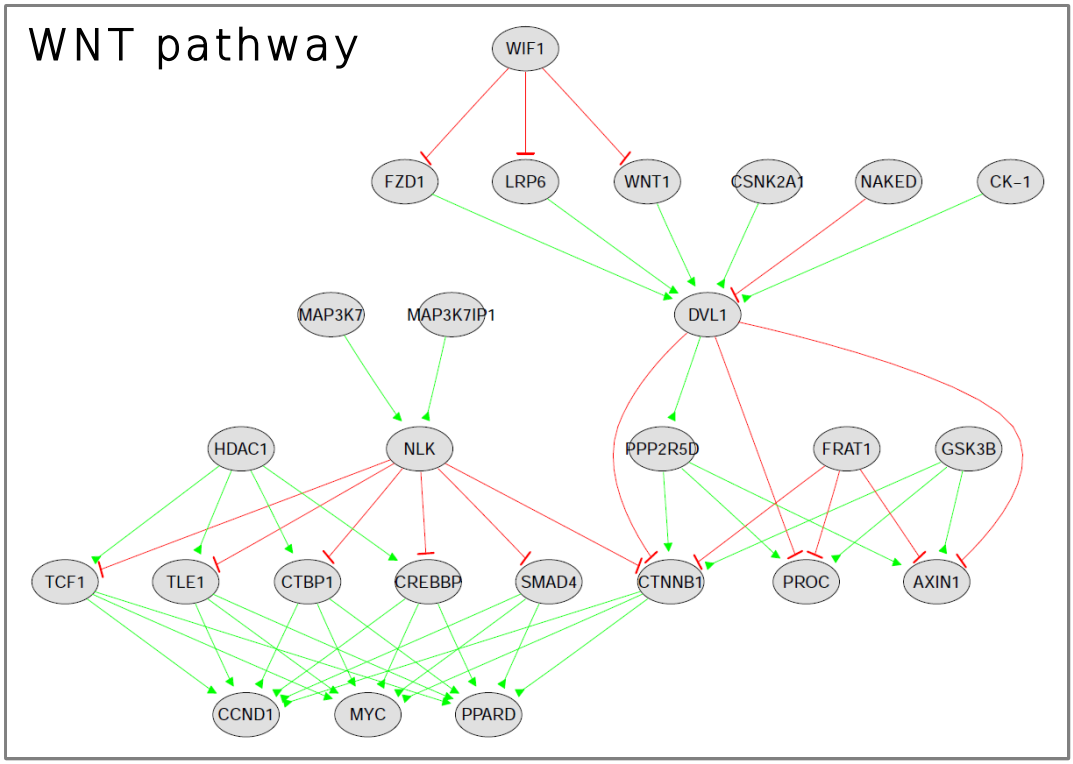
```
pw_1_adj   = pathway2AdjacancyMatrix(biopax, pwid1, expandSubpathways=TRUE,
         splitComplexMolecules=TRUE, verbose=TRUE)
pw_1_graph = pathway2RegulatoryGraph(biopax, pwid1,
         splitComplexMolecules=TRUE, verbose=TRUE)
pw_2_adj   = pathway2AdjacancyMatrix(biopax, pwid2, expandSubpathways=TRUE,
         splitComplexMolecules=TRUE, verbose=TRUE)
pw_2_graph = pathway2RegulatoryGraph(biopax, pwid2,
         splitComplexMolecules=TRUE, verbose=TRUE)
```

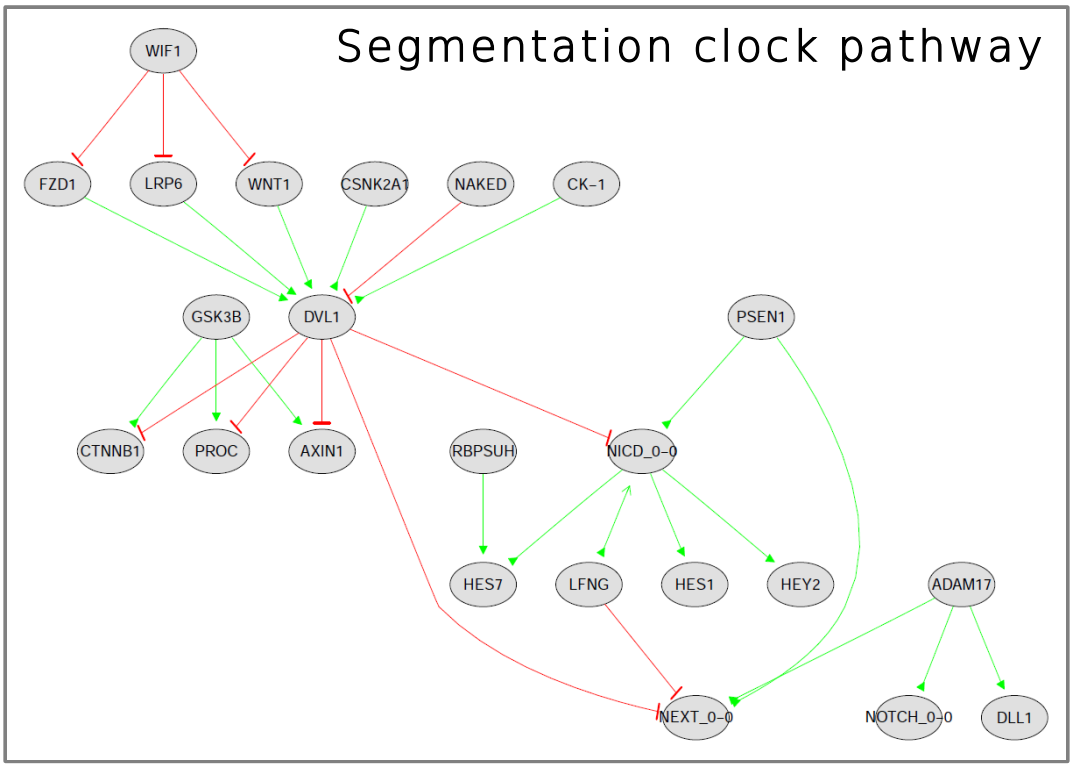
Layout the graphs using Rgraphviz:

```
pw_1_graph_laidout = layoutRegulatoryGraph(pw_1_graph)
pw_2_graph_laidout = layoutRegulatoryGraph(pw_2_graph)
```

Plot the graphs:

```
plotRegulatoryGraph(pw_1_graph)
plotRegulatoryGraph(pw_2_graph)
```

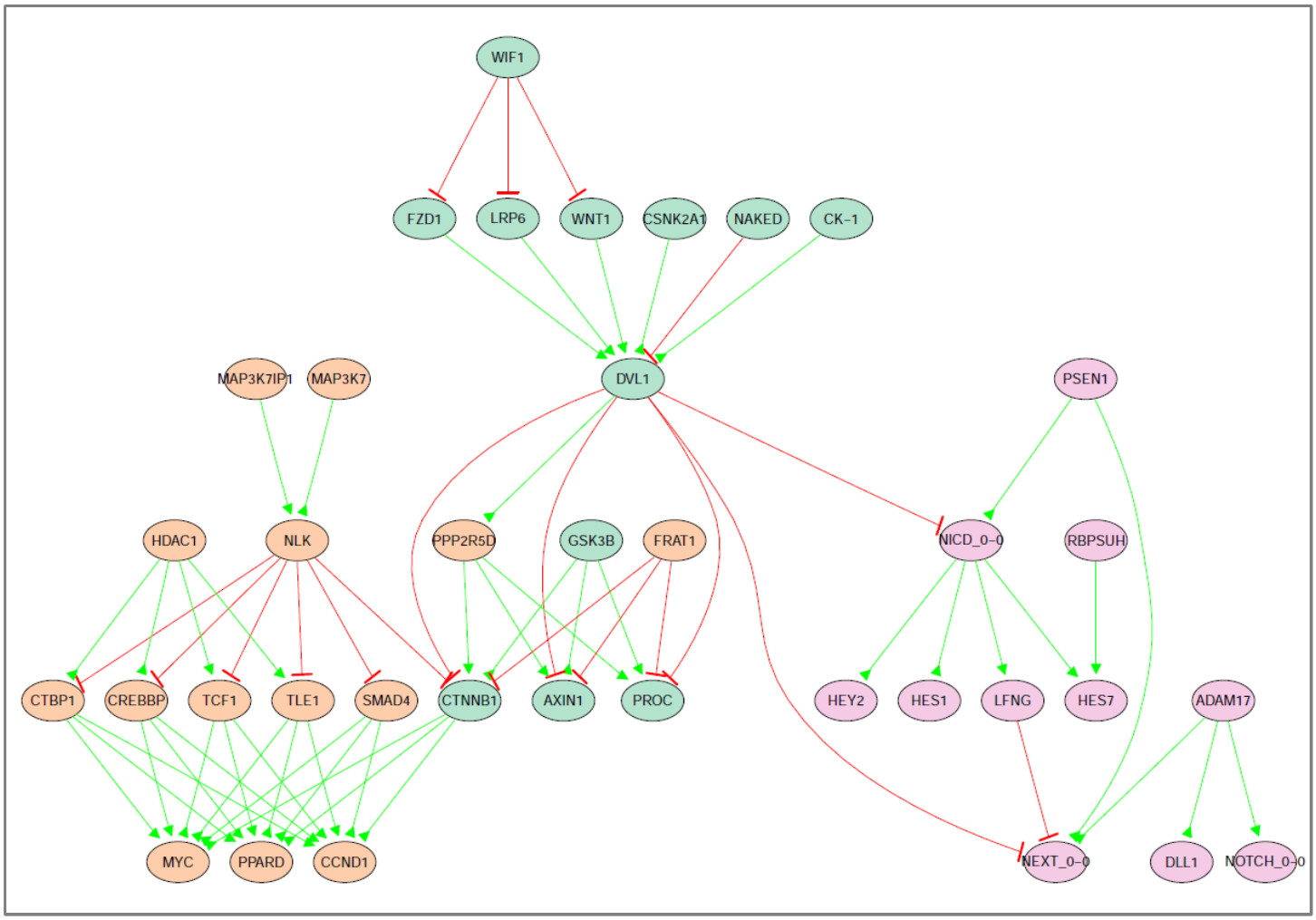




A number of functions can be applied to these regulatory graphs, for example, merge, diff or intersect.

Merge graphs and render them (this time disable re-layouting for the plot function):

```
merged_graph = uniteGraphs(pw_1_graph_laidout,pw_2_graph_laidout)
plotRegulatoryGraph(merged_graph, layoutGraph=FALSE)
```



If you want to make your graphs more beautiful a good start would be to look at Rgraphviz parameters that can be set via *nodeRenderInfo*. For example, try out:

```
nodeRenderInfo(merged_graph)$cex = 1
nodeRenderInfo(merged_graph)$textCol = "red"
nodeRenderInfo(merged_graph)$fill = "green"
plotRegulatoryGraph(merged_graph, layoutGraph=FALSE)
```

7 Modifying BioPAX

Instead of merging the regulatory graph representations it is also possible to merge the biopax pathways directly and add this new, merged pathway directly into the biopax model.

```
biopax = mergePathways(biopax, pwid1, pwid2, NAME="mergedpw1",
        ID="mergedpwid1")
mergedpw_graph = pathway2RegulatoryGraph(biopax, "mergedpwid1",
        splitComplexMolecules=TRUE, verbose=TRUE)
plotRegulatoryGraph(layoutRegulatoryGraph(mergedpw_graph))
```

Although it is possible to directly edit the parsed BioPAX data by accessing *biopax$df*, there are quite a few convenience functions to make life easier. In the following code block a new BioPAX model will be created from scratch using *createBiopax*. Functions *addPhysicalEntity*, *addPhysicalEntityParticipant*, *addBiochemicalReaction*, *addControl* and *addPathway* will be used to build 2 pathways with 2 controls between 3 proteins each.

Start out with adding 5 proteins (Protein_A-E), their corresponding physicalEntityParticipant instances and a biochemical reaction where they do something to themselves.

```
biopax = createBiopax()
for(i in LETTERS[1:5]) {
        biopax = addPhysicalEntity(biopax, class="protein",
                NAME=paste("protein",i,sep="_"), ID=paste("proteinid",i,sep="_"))
        biopax = addPhysicalEntityParticipant(biopax,
                referencedPhysicalEntityID=paste("proteinid",i,sep="_"),
                ID=paste("PEPid",i,sep="_"))
        biopax = addBiochemicalReaction(biopax, LEFT=paste("PEPid",i,sep="_"),
          RIGHT=paste("PEPid",i,sep="_"), ID=paste("BCRid",i,sep="_"))
}
```

Now we add some controls (A-B,A-C,C-D,C-E) between those proteins.

```
biopax = addControl(biopax, CONTROL_TYPE="ACTIVATION",
        CONTROLLER="PEPid_A", CONTROLLED=c("BCRid_B"),ID="control_1")
biopax = addControl(biopax, CONTROL_TYPE="INHIBITION",
        CONTROLLER="PEPid_A", CONTROLLED=c("BCRid_C"),ID="control_2")
biopax = addControl(biopax, CONTROL_TYPE="ACTIVATION",
        CONTROLLER="PEPid_C", CONTROLLED=c("BCRid_D"),ID="control_3")
biopax = addControl(biopax, CONTROL_TYPE="INHIBITION",
        CONTROLLER="PEPid_C", CONTROLLED=c("BCRid_E"), ID="control_4")
```

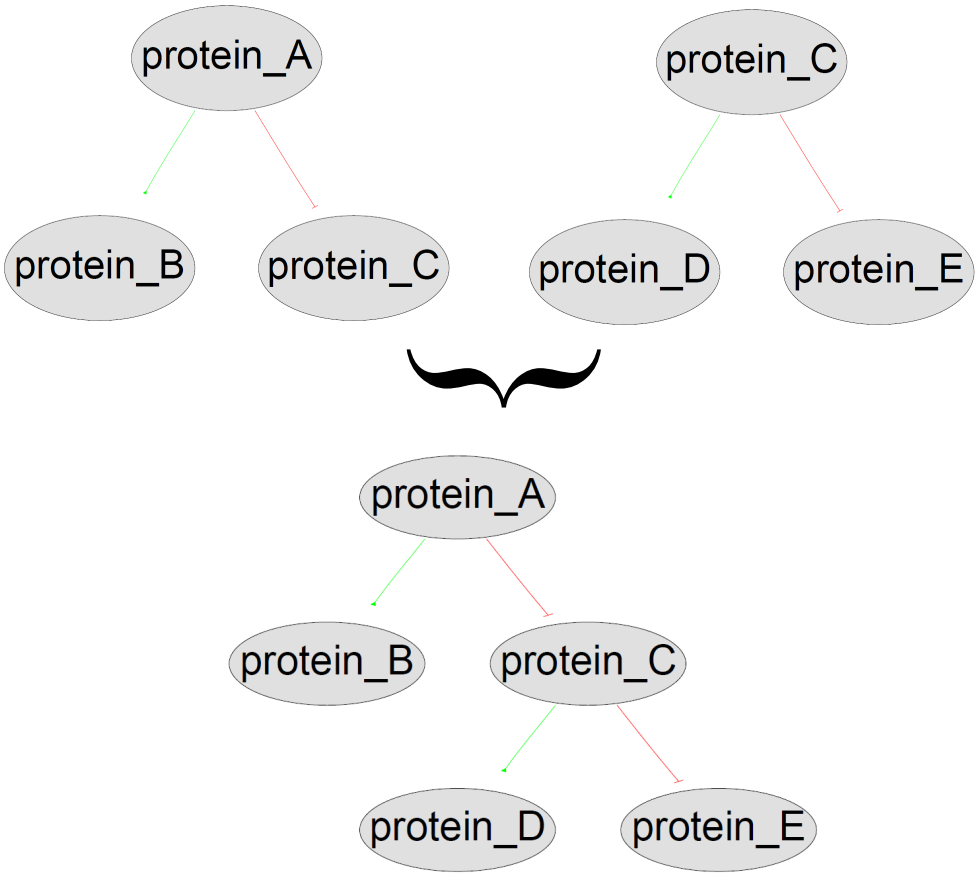
These interactions will be used as pathway components for new pathways by calling *addPathway*.

```
biopax = addPathway(biopax, NAME="pw1",
        PATHWAY_COMPONENTS=c("control_1","control_2"), ID="pwid1")
biopax = addPathway(biopax, NAME="pw2",
        PATHWAY_COMPONENTS=c("control_3","control_4"), ID="pwid2")
biopax = mergePathways(biopax, "pwid1", "pwid2", NAME="pw3", ID="pwid3")
```

Now these new pathways are ready to be viewed!

```
pw1_graph = pathway2RegulatoryGraph(biopax, "pwid1",
        splitComplexMolecules=TRUE, verbose=TRUE)
pw2_graph = pathway2RegulatoryGraph(biopax, "pwid2",
        splitComplexMolecules=TRUE, verbose=TRUE)
pw3_graph = pathway2RegulatoryGraph(biopax, "pwid3",
        splitComplexMolecules=TRUE, verbose=TRUE)

plotRegulatoryGraph(layoutRegulatoryGraph(pw1_graph))
plotRegulatoryGraph(layoutRegulatoryGraph(pw2_graph))
plotRegulatoryGraph(layoutRegulatoryGraph(pw3_graph))
```



Finally, properties as well as complete instances can be removed from the current BioPAX model by calling:

```
temp = biopax
temp = removeProperties(temp,instanceid="newpwid2",
          properties="PATHWAY-COMPONENTS")
temp = removeInstance(temp,instanceid="newpwid3")
```

8 Writing out in BioPAX Format

Writing out an internal BioPAX model into a valid .owl file is very easy.
Simply call:

```
writeBiopax(biopax, file="test.writeBiopax.owl")
```