pyPolyBuilder
User's Guide and Tutorials

Vitor A.C. Horta, Mayk C. Ramos and Bruno A.C. Horta

May 31, 2017

Contents

1 Introduction **3**

2 Dendrimer module **3**

3 General module **5**

4 Tutorials **6**

4.1 PPI 7

4.2 PAMAM 8

4.3 PAMAM/PPI 9

4.4 Toy cyclic polymer 11

4.5 PAMAM/poly-etyleneglycol 15

1 Introduction

The pyPolyBuilder is a python code developed by LabMMol group of UFRJ's Chemistry Institute with the objective of facilitating the topology generation for molecular simularion of polymeric and dendritic structures.

Due to its code written in python, its execution is easy and straightforward. It's used through code line in any Linux distribution. The software is based in composing the repeating units of the interest molecule, called building blocks, to assemble the final molecule. Thus, the necessary inputs to execute the code are the topologies of the building blocks and the output will be the complet molecule topology.

PyPolyBuilder has two modules of application. The dendrimer module and the general. While the dendrimer module is optimized to generation of dendritics topologies, the general is broader and applicable to any structure which could be subdivided into building blocks.

2 Dendrimer module

As inputs to dendrimer module, there are required the building blocks topologies and a force field parameters list. On topologies files, are six available fields to pass information that define the building block structure as exemplified in tutorial files available in supplemental material. They are: *[moleculetype]*, *[atoms]*, *[bonds]*, *[angles]*, *[dihedrals]* and *[branches]*.

The field *[moleculetype]* is where the building block name should be written.

```
[ moleculetype ]
; Name   nrexcl
  C4     2
```

In the field *[atoms]*, each one of the atoms should to be listed with their index number, type, residue number, rusidue name, charge group number, coulombic charge and mass, in that order. The required values are:

```
[ atoms ]
; nr  type  resnr  resid  atom  cgnr  charge    mass
   1   CH3     1      1     C     1      0   15.0350
   2   CH2     1      1     C     1      0   14.0270
   3   CH2     1      1     C     1      0   14.0270
   4   CH3     1      1     C     1      0   15.0350
```

In *[bonds]* should have the information about which atoms are forming chemical bonds. We need to pass their index numbers and information about the force field parameters:

```
[ bonds ]
;  ai aj  funct  ff
    1  2   2    gb_21
```

Where ai and aj are the index number of atom i and j, respectively, involved in the chemical bond and *g_21* is the force field parameter as the input of the gromacs package.

The field *[angles]* is similar to *[bonds]* where should be a list of the set of three atoms having a angle among they:

```
[ angles ]
; ai   aj   ak   funct  angle
    1    2     3  2  ga_21
```

And in a similar way, the field *[dihedrals]* should have a list with sets of 4 atoms specifying the building block dihedrals

```
[ dihedrals ]
; ai    aj    ak    al  funct  died
    1     2     3     4  2  gd_21
```

Note that, as an energy minimization step is implemented, the fields *[angles]* and *[dihedrals]* must not be necessarily provided. But if the user provide these fields, they will be used instead of the energy minimization.

The last field that we need to provide is the *[branches]*, here are organized the atoms that are used to form a bond between two building blocks to propagate the monomer.

```
[ branches ]
;  donor   acceptor
     1        0
     0        4
```

Usually, we can define the donor as the atom that will be bonded in the new building block and the aceptor as the atom that will be bonded in the last building block.

The parameters list file need to have force field information about bonds, angles and dihedrals that exists in our topology. Some cases may require that the stereochemistry confomation be specified, in that cases the conformation field can be used. In the cases where the conformation field are not present, the final structure will have two of them. As example:

```
 # BONDS
 H        N        gb_2
 H        NT       gb_2

 # ANGLES
 H        NT       CH2      ga_11
 H        NT       H        ga_10

 #IMPROPERS
 CH2      C        O        N         gi_1
 C        N        H        CH2       gi_1

 # DIHEDRALS
 CH2      CH2      C        O         gd_40
 CH2      CH2      C        N         gd_40

 # CONFORMATION
 CH2      C        N        CH2      trans
 CH2      C        N        H        cis
```

As the fields are filled as described before, to invoke the software, the following command must be run in the command line:

```
pypolybuilder   --core=coreTopology.itp\
                --inter=interTopology.itp\
                --ter=terTopology.itp\
                --params=list_param.itp\
                --ngen=n\
                --name=topologyName\
                --output=outputName.itp\
                --gro=coordinatesFileName.gro\
                --optgeo
```

The arguments used are indicators of topology of each required building block and *–ngen* indicates the generation of the resulting dendrimer. Hence, this command will use the file *coreTopology.ipt* as the core building block topology, *interTopology.ipt* as the intermediary building block topology and *terTopology.ipt* as the terminal building block to generate a nth dendrimer topology with the name topologyName. The generated files will be outputName.itp with the dendrimer topology and coordinatesFileName.gro with the atoms spacial coordinates, in the case the –name, –output and –gro parameters is not used, the default values are, respectively, NEWTOP, default.itp and default.gro. And the command *optgeo* will make the pyPolyBuilder try to optimize the geometry.

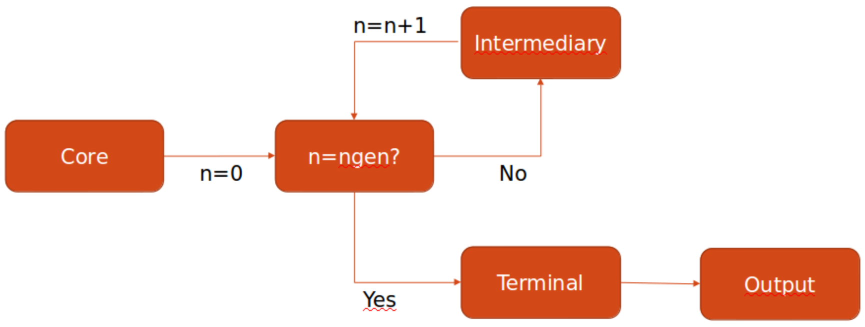


Figure 1: Dendrimer Module behavior

As can be seen in Figure 1, the dendrimer module starts from the core building block and increase the dendrimer generation until it reach ngen. Then add a terminal blocks level and generates the output files.

3 General module

Otherwise, input files of general module are the building blocks topologies, the force field parameters list and a connection file that will have the information about how the blocks are bonded. Unlike dendrimer module, input topologies to general module does not need to have the *[branches]* field, once this information will be contained in connection file. Hence, to the general module, the building blocks topologies need to have only the fields: *[moleculetype]*, *[atoms]* and *[bonds]*. Just as in the dendrimer module.

The connection file has two required fields, they are: *[BUILDING BLOCKS]* and *[CONNECTS]*. In the first field, we need to specify which building blocks will be used by writing their names. And in the second field, we need to tell how the blocks are connected by saying their index number and which two atoms are bonded. For example, if we have a file for each CHn topology:

```
    #[ BUILDING BLOCKS ]
    1  CH3
    2  CH2
    3  CH2
    4  CH3

    #[ CONNECTS ]
    1  2 1 1
    2  3 1 1
    3  4 1 1
```

Once we have all the files, we need to run the following command in the command line:

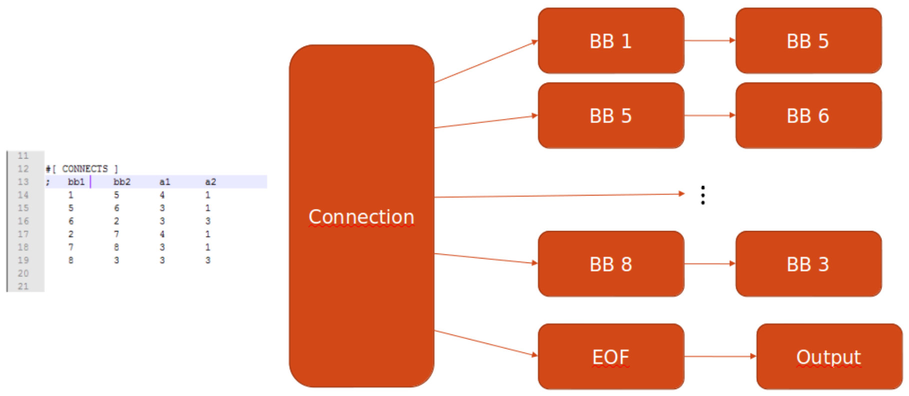
```
  pypolybuilder   --bbs=BB1.itp,BB2.itp,...,BBn.itp\
                  --params=list_param.itp\
                  --connect=connect.in\
                  --polymer\
                  --name=topologyName\
                  --output=outputName.itp\
                  --gro=coordinatesFileName.gro\
                  --polymer\
                  --optgeo
```

One more time the path *PATH* is the location where your pyPolyBuilder is locate in. This command will connect all the n building blocks as described in connect.in and the topology will be named as topologyName. There will be generated 2 files: outputName.itp with the molecule topology and coordinatesFileName.gro with the molecule atoms coordinates. Again the default values to the name, output and gro parameters are NEWTOP, default.itp and default.gro. The pyPolyBuilder default module is the dendrimer one, if we want to use the general module, we need to add a *–polymer* parameter in the command line. The *optget* command will make pyPolyBuilder try to optimize the molecule geometry.

Unlike Dendrimer module, the General module does not have a stop condition. It is made based on the connection list as ilustred in Figure 2.

4 Tutorials

This tutorial will deal with 5 cases, 3 of them in dendrimer module. We will build a PAMAM, a PPI and a hetero dendrimer which half is a PAMAM and the other half is a PPI with a PPI core. While in general module, we will build a fictional molecule where there are a poly-etyleneglycol ring with 4 arms bonded in it through nitrogens atoms. After that, we will use both modules to generate a molecule composed of two half PAMAM bonded by a poly-etyleneglycol. The dendrimers built here will be first generation and in the case which exists the poly-etyleneglycol, we will use 2 monomers.



```
#[ CONNECTS ]
;    bb1     bb2     a1      a2
     1       5       4       1
     5       6       3       1
     6       2       3       3
     2       7       4       1
     7       8       3       1
     8       3       3       3
```

Figure 2: General Module behavior

4.1 PPI

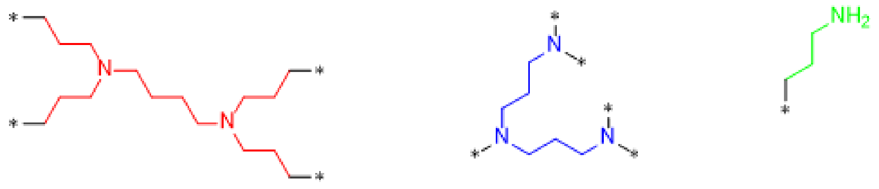


Figure 3: core(red), intermediary(blue) and terminal(green) building blocks used to create PPI dendrimer

In the PPI folder, exist 4 files: *core_PPI.itp*, *inter_PPI.itp*, *ter_PPI.itp* and *list_params.itp*, there are, respectively, core, intermediary and terminal building block topologies, as shown in Figure 3 and the list of parameters of the force field. To use pyPolyBuilder, we need to go to the tutorial folder:

```
/$PATH/tutorial/PPI/.
```

where *PATH* is the where your supplemental material is located. And execute the command:

```
pypolybuilder   --core=core_PPI.itp\
                --inter=inter_PPI.itp\
                --ter=ter_PPI.itp\
                --params=list_param.itp\
                --ngen=1\
                --name=PPI\
                --output=PPI.itp\
                --gro=PPI.gro\
                --optgeo
```

Here we are using the *–output* and the *–gro* options so the generated files will be named as PPI.itp and PPI.gro, there are information about atoms coor-

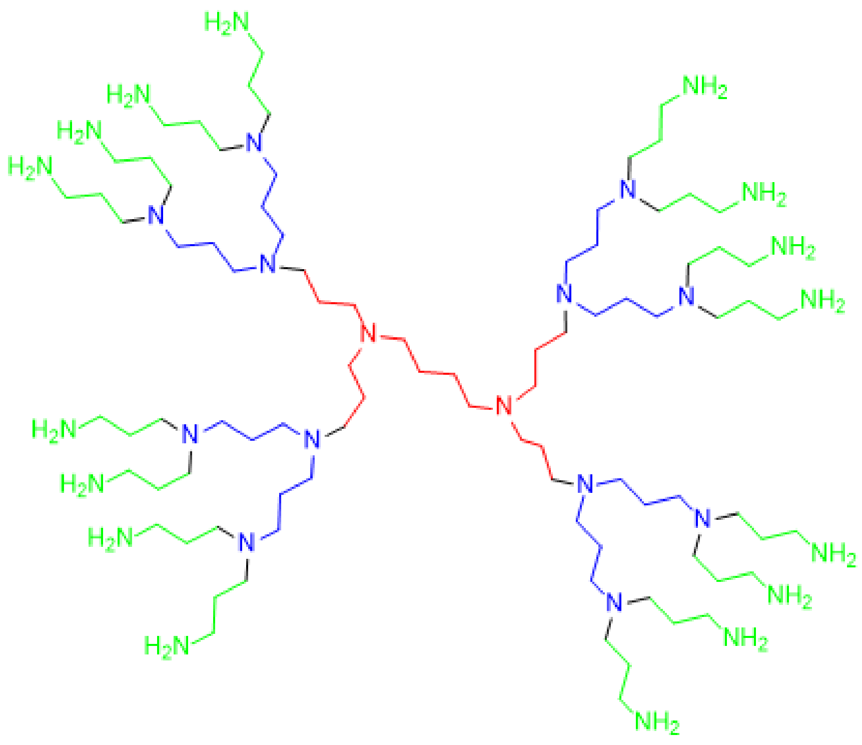


Figure 4: Complete generation one PPI dendrimer

dinates and created molecule topology, respectively. The generated topology is related to the molecule shown in Figure 4

4.2 PAMAM

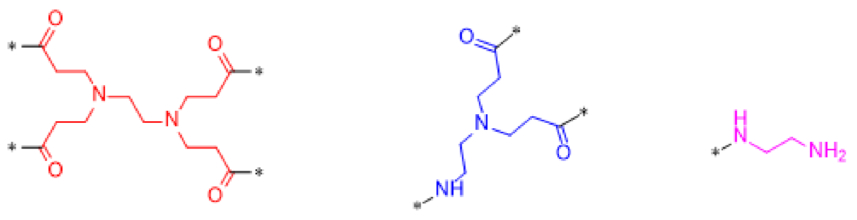


Figure 5: core(red), intermediary(blue) and terminal(pink) building blocks used to create PAMAM dendrimer

In a similar way of PPI creation methodology, we can create a PPI dendrimer. In the tutorial folder will be available files containing the PAMAM building blocks topology as shown in Figure 5. As PAMAM dendrimer has a amide functional group, we need to specify the stereochemistry of the dihedral, for that there are some differences in *list_params.itp*, actually there is a new field called *#CONFORMATION*. There we can say which dihedrals are trans or cis conformation.

```
# CONFORMATION
CH2     C       N       CH2     trans
CH2     C       N       H       cis
O       C       N       CH2     cis
O       C       N       H       trans
```

PAMAM is one of the molecules that need to be specified the stereochemistry conformation for the amide functional group. So, in this force field parameters file, will have a *#CONFORMATION* field as discussed before. We can go to the tutorial folder with the files *core_PAMAM.itp*, *inter_PAMAM.itp*, *ter_PAMAM.itp* and *list_params.itp*:

```
/$PATH/tutorial/PAMAM/.
```

here, again, *PATH* is the where your supplemental material is located. And use the command line to call pyPolyBuilder passing these files as parameters:

```
pypolybuilder   --core=core_PAMAM.itp\
                --inter=inter_PAMAM.itp\
                --ter=ter_PAMAM.itp\
                --params=list_param.itp\
                --ngen=1\
                --name=PAMAM
                --output=PAMAM.itp\
                --gro=PAMAM.gro\
                --optgeo
```

And as in the PPI case, two files containing the atoms coordinates and the created dendrimer topology, they are: PAMAM.gro and PAMAM.itp, respectively. And a topology to the molecule shown in Figure 6 will be created.

4.3 PAMAM/PPI

In the case that we want to create a hetero dendrimer, we need to do some intermediary steps. As discused previously, the camp *[branches]* in any of each topologies is responsable to control which atoms from the building blocks will form chemical bonds. If the pyPolyBuilder is executed as it is configured in the previously tutorials, it will form chemical bonds in all the four donors atoms of the core. To prevent this to happening, we need to open the *core_PPI.itp* file and remove the two atoms which we do not want the chemical bonds to be formed (the atoms 15 and 18). Hence, we will be mimicking the terminal protection process. After deleting the lines for atoms 15 and 18 in *core_PPI.itp [branches]* field, save it and execute the software once:

```
pypolybuilder --core=core_PPI.itp\
              --inter=inter_PPI.itp\
              --ter=ter_PPI.itp\
```

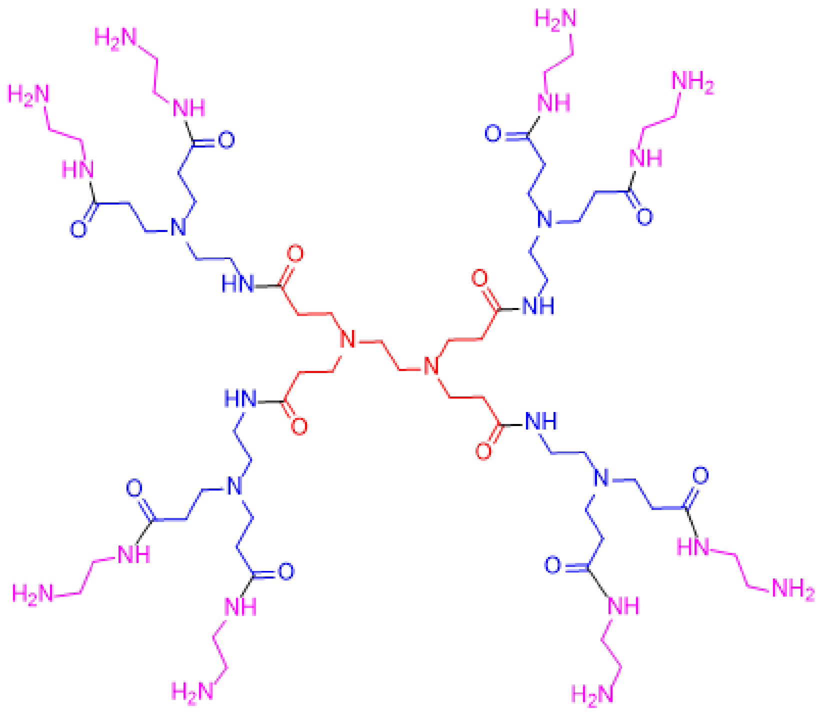


Figure 6: Complete generation one PAMAM dendrimer

```
                            --params=list_param.itp\
                            --ngen=1\
                            --name=PPIhalf\
                            --output=PPIhalf.itp
```

Note that we do not use the command *–optgeo* because, as it is an intermediary step, we do not need to optimize this geometry. When it ends, the PPI dendrimer will only be branched in the core atoms which was in the *[branches]* field. We will have a half branched dendrimer with PPI intermediary and terminal as can ben seen in Figure 8. We now use this half dendrimer as the core to do the ramifications of PAMAM half. We need to open the file PPI.itp that was created, and write the field *[branches]* in the end to specify in which atoms the pyPolyBuilder have to form chemical bonds now. Note that as we have a more complex molecule, the atoms list is longer and the index number of intermediary and terminal blocks was probably changed. But as the core block is allocated first, their index number remains the same. So, we can write the *[branches]* field where the atoms 9 and 12 will be aceptors and save it.

```
[ branches ]
;  donor    acceptor
     0     9
```

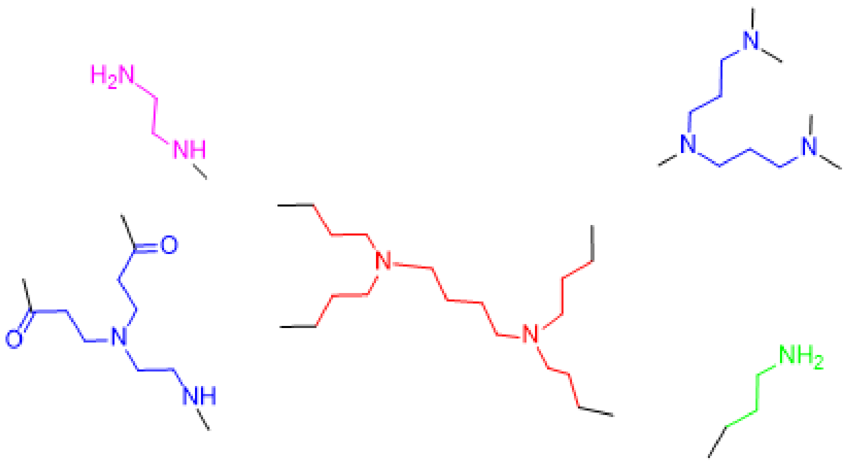


Figure 7: PPI core(red), PPI intermediary(blue at right), PPI terminal (green), PAMAM intermediary (blue at left) and PAMAM terminal (pink) building blocks used to create PAMAM_PPI dendrimer

```
       0    12
;        0      15
;        0      18

[ exclusions ]
;  ai    aj
```

Now we run the command:

```
pypolybuilder --core=PPIhalf.itp\
                --inter=inter_PAMAM.itp\
                --ter=ter_PAMAM.itp\
                --params=list_param.itp\
                --ngen=1\
                --name=PAMAM_PPI\
                --output=PAMAM_PPI.itp\
                --gro=PAMAM_PPI.gro\
                --optgeo
```

And then, will be used the half PPI dendrimer molecule as the core and the PAMAM intermediary and terminal building blocks will be used to generate a dendrimer as the Figura 9. The molecule information will be stored in *PPI_PAMAM.itp* and *PPI_PAMAM.gro* files.

4.4 Toy cyclic polymer

In this tutorial, we will use the general module. There are many ways that we can make the proposed molecule using the general module. Two of them are described above. The first one is to use the smallests building blocks and a

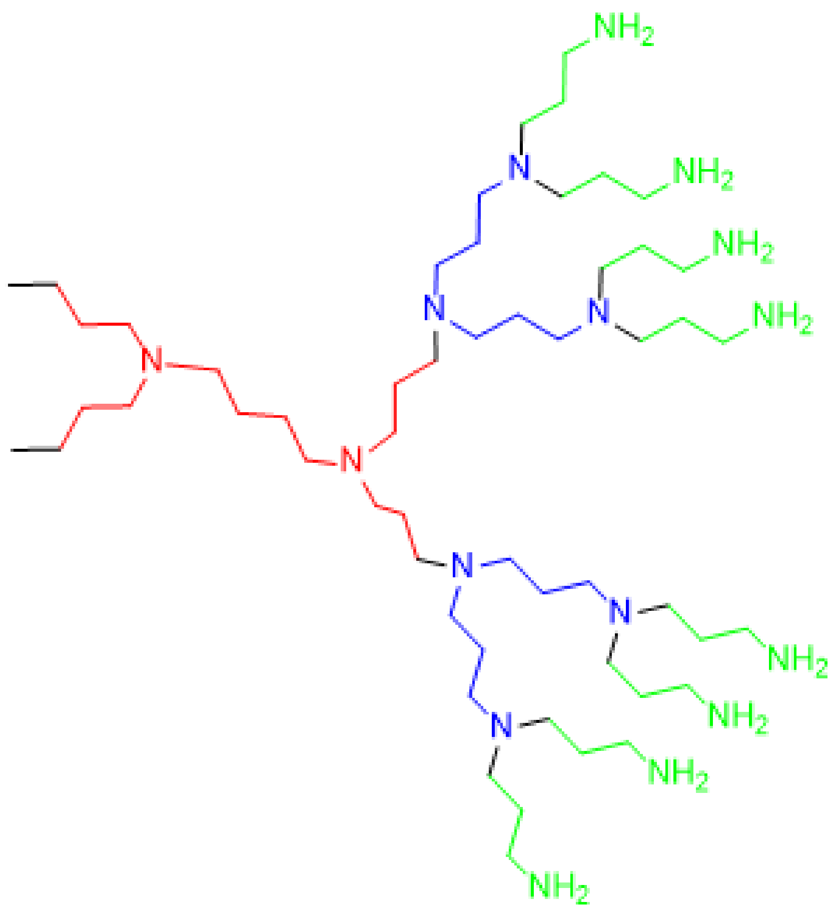


Figure 8: Complete generation one PPI half dendrimer

connection file to create the final molecule directly. The second one is to write a connection file to create, firstly, the central ring. Then, create the side arms. And finally write a connection file to unite all of them.

In the first approach, there are the files *nc3.itp*, *ter-etyleneglycol.itp*, *etyleneglycol.itp* and *connect.in* in this case folder in the supplemental material. Itp files contains the building blocks topologies and the connect.in explain how they have to be connected to each other in way to make the final molecule. We can run the command line:

```
pypolybuilder --bbs=nc3.itp,etyleneglycol.itp,ter-
    etyleneglycol.itp\
                  --in=connect.in\
                  --params=list_param.itp\
```

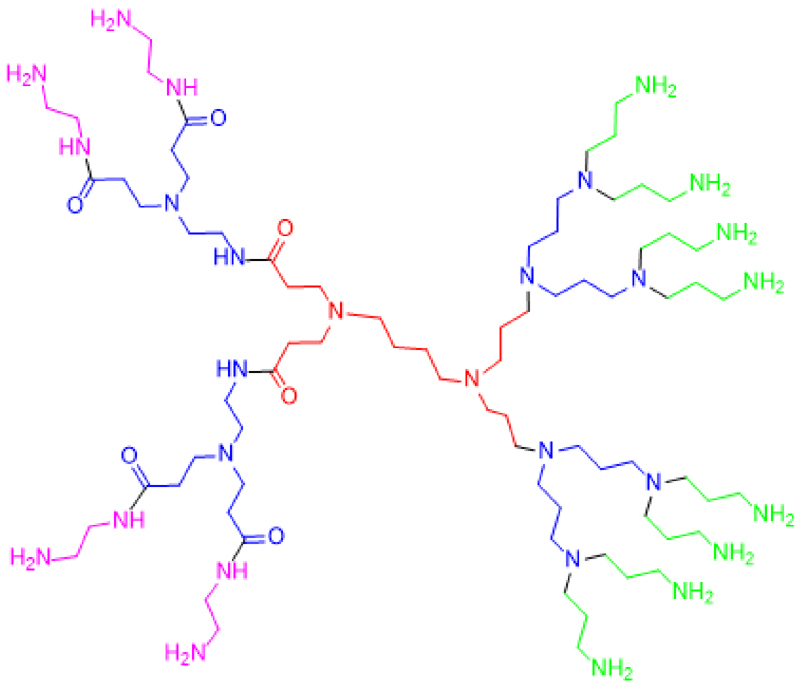


Figure 9: Complete generation one PAMAM_PPI hetero dendrimer

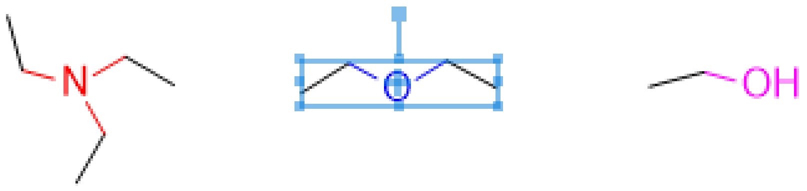


Figure 10: The three used building blocks to polymer molecule generation.

```
                   --name=polymer\
                   --output=polymer.itp\
                   --gro=polymer.gro\
                   --optgeo
```

So the molecule will be created and the topology and atoms coordinates saved in polymer.itp and polymer.gro files.

Otherwise, in second approach, we need to make each molecule part independently. To create the central ring topology, run the command line:

```
pypolybuilder --bbs=nc3.itp,etyleneglycol.itp\
                --in=connect_ring.in\
                --params=list_param.itp\
                --name=ring\
                --output=ring.itp\
                --polymer
```

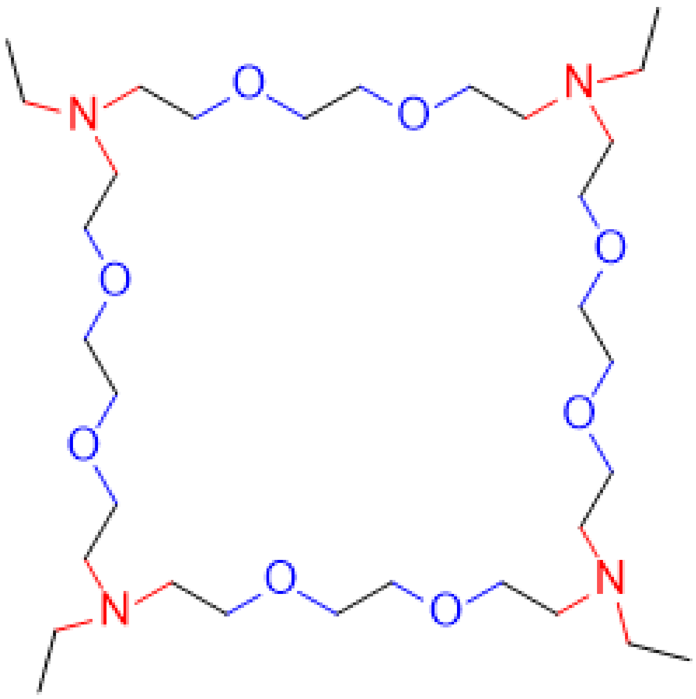


Figure 11: Created ring topology

And then, let's create the side arms topology:

```
pypolybuilder --bbs=etyleneglycol.itp,ter-
    etyleneglycol.itp\
                --in=connect_arm.in\
                --params=list_param.itp\
                --name=arm\
                --itp=arm.itp\
                --polymer
```

Finally, we can unite them using the *connect_BBs.itp* file:

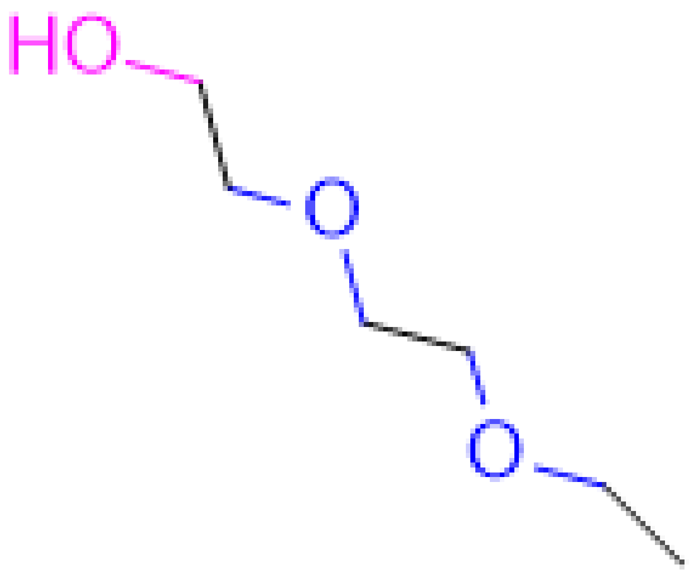


Figure 12: Created arm topology

```
pypolybuilder --bbs=ring.itp,arm.itp\
                --in=connect_BBs.in\
                --params=list_param.itp\
                --name=polymer\
                --itp=polymer.itp\
                --gro=polymer.gro\
                --polymer\
                --optgeo
```

Both approach will generate the same final molecule. But in different ways. It is up to each user to decide which is the best procedure for their study.

4.5 PAMAM/poly-etyleneglycol

In our last tutorial, we will use both of the pyPolyBuilder modules to create the wanted molecule. In fact, we will need to create an half PAMAM dendrimer with the dendrimer module first and then use the general module to connect two of these halfs through a poly-etyleneglycol.

The first step is use the adapted PAMAM core topology to create the dendrimer part. The adaptation was break the core topology at their middle by deleting the atoms that we no more longer wanted (Red building block in Figure 14). Was necessary that the bonds, angles, dihedrals and branches were adapted to the new topology. The resulting topology file is available in the case

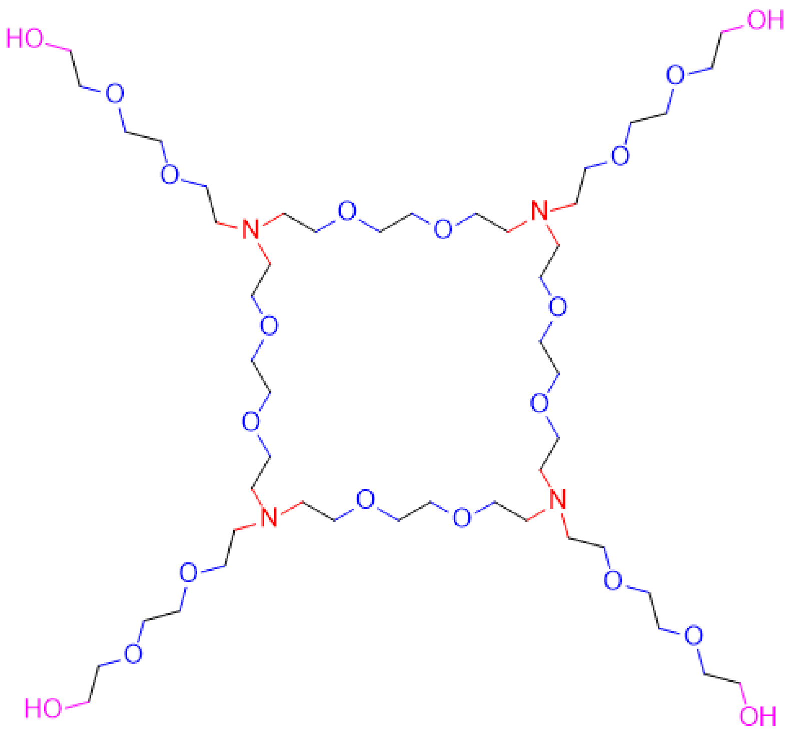


Figure 13: Complete final toy polymer molecule

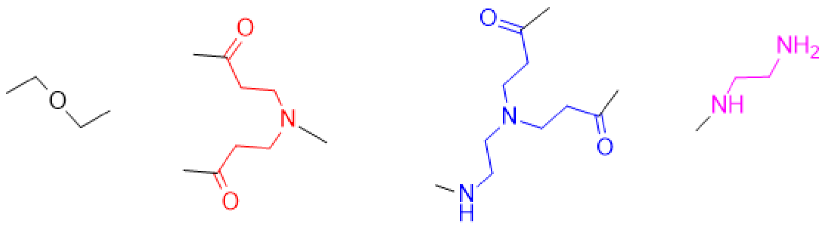


Figure 14: The adapted PAMAM core (red), the PAMAM intermediary (blue), the PAMAM terminal (pink) and the etyleneglycol building blocks topologies that were used to create the PAMAM_PolyEtyleneglycol topologies

supplemental material folder.

Let's run the command line that will create this part of the final molecule:

```
pypolybuilder --core=core_PAMAM.itp\
                            --inter=inter_PAMAM.itp\
                            --ter=ter_PAMAM.itp\
                            --params=list_param.itp\
                            --ngen=1\
                            --name=PAMAMhalf\
                            --output=PAMAMhalf.itp\
```

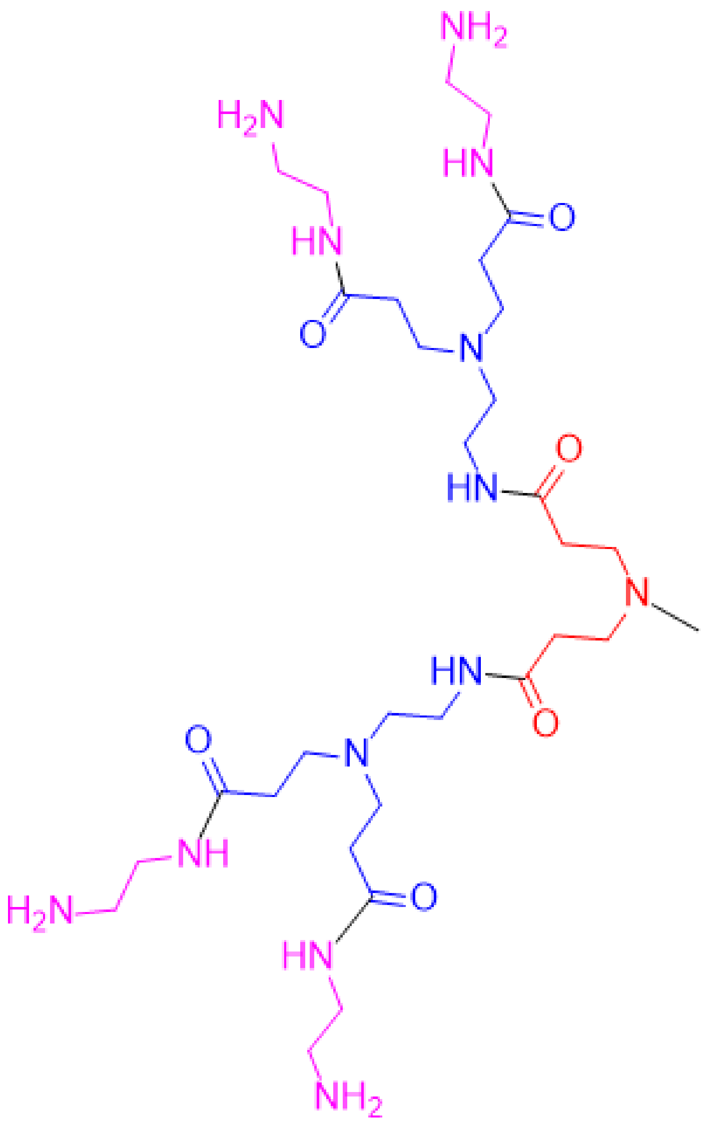


Figure 15: The adapted PAMAM core topology.

Where we do not use the *–optgeo* command because we do not want to spen time optimizing a geometry that is not the final one. Once we have the PAMAM half topology, we can use the *connect.in* file to connect the building blocks with the general module:

```
pypolybuilder   --bbs=PAMAMhalf.itp,etyleneglycol.
    itp\
                --in=connect.in\
                --params=list_param.itp\
                --name=PAMAM_PolyEtyleneglycol\
                --itp=PAMAM_PolyEtyleneglycol.itp\
                --gro=PAMAM_PolyEtyleneglycol.gro\
                --polymer\
                --optgeo
```

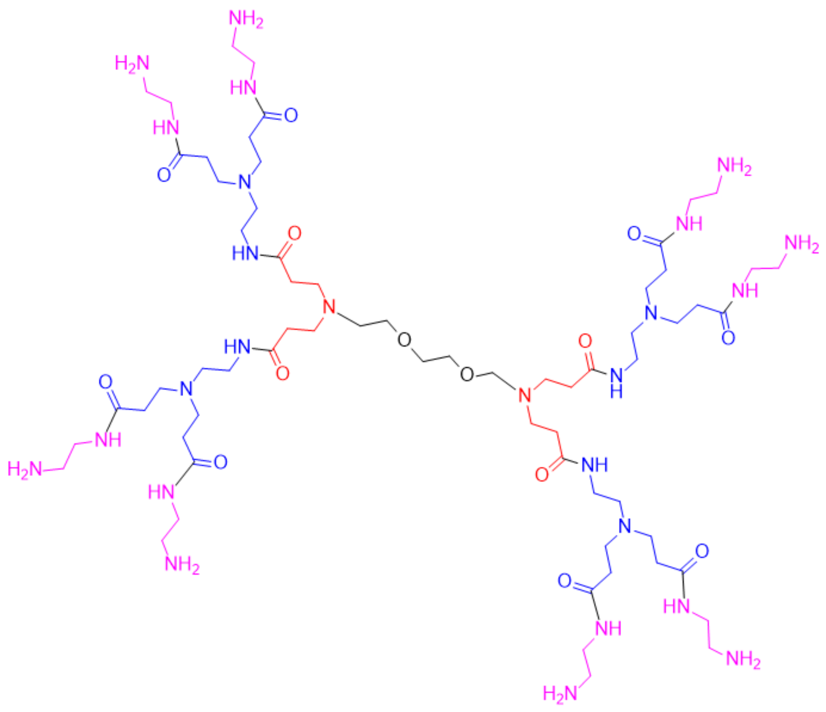


Figure 16: The complete PAMAM_PolyEtyleneglycol molecule.