How to design synthetic gene circuits with composable parts and pools in the ProMoT framework

Mario Andrea Marchisio

ETHZ, Department of Biosystems Science and Engineering (D-BSSE)
Mattenstrasse 26, CH-4058 Basel
September 2008

Contents

1 Part generation **2**
- 1.1 Standard parts . . . 2
  - 1.1.1 Promoter . . . 2
  - 1.1.2 RBS . . . 4
  - 1.1.3 Coding part . . . 5
  - 1.1.4 Terminator . . . 5
  - 1.1.5 sRNA . . . 6
  - 1.1.6 The DNA-spacer . . . 6
- 1.2 Pool generation . . . 6
  - 1.2.1 Polymerase pool . . . 6
  - 1.2.2 Ribosome pool . . . 7
  - 1.2.3 Transcription factor pool . . . 7
  - 1.2.4 Signal pool . . . 7
  - 1.2.5 sRNA pool . . . 7
  - 1.2.6 Enzymatic reaction pool . . . 7

2 Circuit design **9**
- 2.1 Normal representation . . . 9
- 2.2 Compact representation . . . 14

Chapter 1

Part generation

This manual makes use of the notation adopted in [Marchisio, 2008] where a detailed description of every composable standard biological part and pool is provided.
Circuit design by means of standard biological parts and pools [Marchisio and Stelling, 2008] needs the software **ProMoT** [Ginkel et al., 2003] available at:
http://www.mpi-magdeburg.mpg.de/projects/promot .
All the necessary files here described to generate parts and pools are integrated inside ProMoT.

1.1 Standard parts

For each part, a perl script (*part_name_gen.pl*) together with an input file (*part_name.inp*) is available.
After downloading ProMoT, perl scripts and input files will be located into the directory:
/usr/local/diana/promot/kb/scripts/synthetic-biology/.
Create, then, a working directory and copy here all the input files (as well as the icon–*png* ones).
Using th ProMoT GUI, in order to generate the MDL file associated with a given part, select "Run Script" from the "File" menu: a new window will appear. Here you have to specify your working directory ("Directory containing input file"); the perl script you want to run ("Script File") and the related input file ("Input File"). No perl script for part and pool generation requires extra commands so the field "command line arguments" shall remain blank.
Click on "Edit Input File" to visualize the selected input file and to modify all the necessary parameter values. Please, mantain the originary input file format:

- the first two lines must not contain any parameter values;
- parameter values and names have to be separated by colon (:);
- parameter values do not have to be enclosed between quotation marks or apices.

Save any changements (click on "Model" and then on "Save") before closing this editor window.
Click on "Generate Model" to run the selected perl script: possible error messages or warnings will be displayed on the "Output Log" panel.
If everything goes well, a new MDL file will be created inside your working directory.
Repeat the above operations for each part and pool, then close the Run Script window.
In the following, an explanation of the entries of every input file is given.

1.1.1 Promoter

The first eleven input file entries are configuration parameters which have to be chosen carefully. Inconsistencies among them cause the end of the promoter generation.

Accepted configuration values are:

operator number : 0, 1 or 2;

transcription factor number : 1 or 2;

O1 transcription factor : Ra1, Ri1, Aa1 and Ai1;

O2 transcription factor : Ra1, Ri1, Aa1, Ai1, Ra2, Ri2, Aa2 and Ai2;

O1 signal : I1 and C1;

O2 signal : I1, C1, I2 and C2;

cooperativity : y or n;

synergistic activation : y or n;

incoming readthrough : y or n;

promoter flag : y or n

As for the transcription factors: the first letter (must be capital) represents its nature (R for repressors; A for activators) whereas the second one (small) refers to its "free" state (i. e. just after being transcribed: i, inactive; a, active). The number indicates the operator where the transcription factor binds to. However, if O2 hosts the same transcription factor as O1, the only transcription factor entering the promoter has to take always the number 1. This is valid also for the two possible environmental signals: I, inducers, and C, corepressors.
Note that it is forbidden the configuration where O1 is occupied by an activator and O2 by a repressor.
Cooperativity can arise only between transcription factors of the same type; synergistic activation concerns only activators.
Repressors can bind to the promoter without cooperativity; activators will show either cooperativity or synergistic activation.
Readthrough has to be set to *y* only if the promoter is preceded by a terminator, otherwise an extra, useless, terminal would be created (and then it would be necessary to plug it).
Promoter flag as to be set to *y* only in the particular case where the promoter is preceded by another promoter.
All the other entries are kinetic paramenter values. If some of them are not necessary (for instance k1sa, k_1sa and k2sa when synergistic activation is neglected) ignore them but do not take them off from the file. The numerical values can be cancelled, the parameter names not.
If a *required* parameter value has not been set, a default value will be assigned to it.

Kinetic parameters

p_free : initial free promoter concentration. Refers to P, O_1^f or $O_1^f O_2^f$ depending on the chosen operator number;

alpha1 : corresponds to α_1; to be set only for one-operator promoter;

beta1 : corresponds to β_1; to be set only for one-operator promoter;

alpha1f ÷ beta2t : parameters of the same names are found only in two-operator promoters;

lambda1 *and* **mu1** : correspond to λ_1 and μ_1 present both in one and two-operator promoters;

lambda2 *and* **mu2** : correspond to λ_2 and μ_2; they are present only in two-operator promoters;

k_d1 : decay rate constant associated with the first transcription factor. The value of this entry holds for all the following parameters: k_{D1}, k_{D12}, k_{D12p}, k_{D13}, k_{D14}, k_{D13p}, k_{D14p};

k_d2 : decay rate constant associated with the second transcription factor. It has to be set only in case of two-operator promoters and holds for the following parameters: k_{D2}, k_{D22}, k_{D22p}, k_{D24}, k_{D24p};

gamma1 : kinetic rate constant associated with the interaction of n_1 environmental signals with a transcription factor bound to the first operator. It holds for: γ_{12}, γ_{12p}, γ_{13}, γ_{13p}, γ_{14}, γ_{14p};

gamma2 : the same as gamma1, but the operator interested is the second one. It has to be taken into account only in case of two-operator promoters and holds for: γ_{22}, γ_{22p}, γ_{24}, γ_{24p};

k1 : Michaelis-Menten association rate constant. To be set in any kind of promoter, it holds for: k_1, k_{12}, k_{13};

k_1 : Michaelis-Menten dissociation rate constant. To be set in any kind of promoter, it holds for: k_{-1}, k_{-12}, k_{-13};

k2 : transcription initiation frequency. To be set in any kind of promoter, it holds for: k_2, k_{22}, k_{23};

k1sa : Michaelis-Menten association rate constant in case of synergistic activation. Corresponds to k_{14};

k_1sa : Michaelis-Menten dissociation rate constant in case of synergistic activation. Corresponds to k_{-14};

k2sa : transcription initiation frequency in case of synergistic activation. Corresponds to k_{24};

k2_lk : transcription initiation frequency due to leakage. To be set both in one and two-operator promoters. It holds for: k_{21}^{lk}, k_{22}^{lk}, k_{23}^{lk}, k_{24}^{lk}, k_q^{lk}.

1.1.2 RBS

An RBS can be of three **types**:

n : normal RBS, preceded by a promoter or a coding part and followed by another coding part;

c : preceded by a crRNA part;

s : followed by a coding part and a DNA-spacer in sequence i. e. belonging to a multicistronic region.

When **polymerase readthrough** is set to y a new terminal will be added. It will get the $PoPS^{rt}$ signal coming from the next terminator; **ribosome readthrough** has to be considered only when a multicistronic sequence is under study. It shall be set to y only when the RBS is placed after a spacer. An extra terminal to receive the $RiPS^{rt}$ signal will then be created. **repressed by sRNAs**, when set to y, means that the RBS will be connected to an sRNA pool. sRNAs will then compete with ribosomes to have access to the mRNA ribosome binding sites. **pseudo-cistronic** has to be set to y when an RBS is placed inside a multicistronic sequence without spacers and it is preceded by a coding part; **first pseudo-cistron** is on the contrary required to be set to y when an RBS belongs to the first pseudo-cistron of a pseudo-cistronic region, being preceded by a promoter.

Kinetic parameters

k_el : polymerase elongation rate inside RBS;

k_d : mRNA decay rate constant. Depending on the kind of RBS it holds for: k_d, k_{db}, k_{dlb}, k_{dkl}, k_{drt};

k1r, **k_1r** : Michaelis-Menten ribosomal associaton and dissociation rate constants;

k2r : translation initiation frequency. It holds both for k_{2r} and k_{2r}^{rt}.

The other parameters refer to the ones of the same name present either in "RBS for crRNAs" (**csi_k**, **csi_l**, **phi**, **chi**, **theta_k** and **theta_l**) or in "RBS repressed by sRNAs" (**csi_s** and **theta_s**).

1.1.3 Coding part

Five different **types** of coding parts are available:

p : normal protein coding. It is preceded by an RBS and followed by a terminator or by another RBS (multicistronic sequence without spacers i. e. translational readthrough);

r : reporter protein coding. It is associated with the production of fluorescent proteins;

t : transcription factor coding. This part encodes for repressors or activators and is connected also to a transcription factor pool;

e : enzyme coding. The product of this part is an enzyme which is then sent to an appropriate pool;

c : cistronic coding part. It is always preceded by an RBS and followed by a spacer. Differently from the other coding parts it has only two terminals.

If the coding part lies on the first cistron of a multicistronic sequence containig spacers, **first cistron** has to be set to *y*.

Kinetic parameters

k_el : polymerase elongation rate constant inside a coding part;

k_el_r : ribosome elongation rate constant inside a coding part;

k_d : protein decay rate constant (corresponds to k_D);

zeta_r : ribosome dissociation rate constant from mRNA (ζ_r).

1.1.4 Terminator

When **forward readthrough** is set to *y* one more terminals will be generated. It will be connected to the promoter belonging to the next transcription unit; **backwards readthrough** (set to *y*) adds one more terminal to be connected to a previous RBS placed on the same trascription unit as the terminator. In both cases the transmitted signal is a $PoPS^{rt}$.
promoter flag has to be set to *y* just in the hypothetical case where a terminator is preceded by a promoter.
RNA flag has to be set to *y* only if the terminator is preceded by a taRNA part. A new terminal, connected to the sRNA pool, will be drawn.

Kinetic parameters

zeta : polymerase dissociationn rate constant (ζ);

eta : polymerase readthrough rate constant (η).

1.1.5 sRNA

sRNA can be of three different **types**:

c : crRNA, always connected to an RBS, where it sends *locks* to;

t : taRNA, encoding *keys*, for instance;

r : repressive sRNA. It encodes small RNAs which compete with ribosomes to bind to the RBS on mRNA.

Kinetic parameters

k_el : polymerase elongation rate constant inside sRNA parts;

k_d : free sRNA decay rate (k_d).

1.1.6 The DNA-spacer

The **readthrough** entry has to be set to *n* only when the spacer is followed by a terminator, where the ribosomes cannot enter. Otherwise, an outcoming flux of readthrough ribosomes is generated. If the spacer is placed on the first cistron of a multicistronic sequence, **first cistron** is required to be set to *y*.
tf_flag has to be set to *y* only when the spacer is preceded by a transcription factor coding part. In this case a new terminal connected to the transcription factor pool will be added; an analogous function has the **enzyme_flag** entry.

Kinetic parameters

k_d : released protein decay rate constant (k_D);

k_el : polymerase elongation rate constant inside the spacer;

zeta_r : ribosome-mRNA dissociation rate constant. It holds both for ζ_r and ζ_r^{rt};

eta_r : ribosome readthrough rate constant. It holds both for η_r and η_r^{rt}.

1.2 Pool generation

Two entries are common to every signal-carrier pool:

n_in : number of terminals which receive one (or more) positive signal-carrier flux from circuit parts;

n_b : number of terminal which receive a negative signal-carrier flux from circuit parts.

These are the only entries necessary to construct both a "polsum" and "ribsum" part.

1.2.1 Polymerase pool

The **n_in** entry shall be equal to the number of terminators present in the circuit whereas the **n_b** one should count for every promoter. **pol_free** represents the total conserved amount of RNA polymerases available in the circuit. At the beginning of a simulation all the RNA molecules are free and stored inside the pool.

1.2.2 Ribosome pool

The **n_in** entry shall be set equal to the number of circuit RBSs; the **n_b** one depends on the number of coding parts and spacers in the network. Note that a spacer (unless it occupies the first cistron) has two different terminals connected to the ribosome pool. **rib_free** is the total conserved amount of ribosomes present in the system.

1.2.3 Transcription factor pool

The **type** entry has to be set to *m* if the pool contains just monomers; to *d* if a dimerization reaction is forseen.
The **n_in** entry is associated with the number of transcription factor coding parts placed in the circuit whereas the **n_b** one depends on the number of operator-containing promoters.

Kinetic parameters

k_d : free transcription factor decay rate constant (k_D);

delta : dimerization rate constant (δ);

epsilon : dimer dissociation rate constant (ϵ);

tf_free : initial amount of free monomers stored in the pool.

1.2.4 Signal pool

The **n_in** entry is normally set to 0. However, when the signal is produced outside the pool, it can assume any integer values different from 0 and the **input_flag** entry has consequently to be set to *y*; **n_b** is equal to the number of promoters communicating with the pool.

Kinetic parameters

sig_free : initial environmental signal concentration;

k : signal production rate;

k_d : signal decay rate constant.

1.2.5 sRNA pool

The **n_in** entry depends on the number of terminator releasing taRNAs and the **n_b** one on the number of RBS accepting them.

Kinetic parameters

srna_free : initial concentration of free sRNAs;

k_dk : free signal (keys) decay rate constant (k_{dk}).

1.2.6 Enzymatic reaction pool

For this pool, which does not contain signal carriers, two different entries have to be set: **n_in_e**, that represents the number of parts (enzyme codings or sRNAs) producing enzymes; **n_in_s**, that reflects the number of parts (again enzyme codings or sRNAs) producing the substrate. The former is always greater than zero whenever the latter has to be set to a positive value only if **Substrate-like input flux** has been set to *y*.
Other configuration entries are:

Enzyme-like input flux type : enps or rnaps;

Substrate-like input flux type : enps or rnaps;

Output flux type : pops, rips, faps, sips or rnaps.

Kinetic parameters

su_free : initial concentration of free substrate;

k1e : enzyme-substrate Michaelis-Menten association rate constant;

k_1e : enzyme-substrate Michaelis-Menten dissociation rate constant;

k2e : production rate constant;

k_de : free enzyme decay rate constant;

k_ds : free substrate decay rate constant;

k_des : enzyme-substrate complex decay rate constant.

Chapter 2

Circuit design

2.1 Normal representation

Let us design a one-step cascade circuit made of eight standard biological parts (divided into two transcription units) and four pools. They are:

p_basic : basic promoter (two terminals: exc_pol; out_pol);

rbs_rt : RBS able to receive a $PoPS^{rt}$ signal (four terminals: in_pol; exc_r; out_cod; in_rt_pol);

tf_cod : transcription coding part (four terminals: in_rbs; out_r; out_pol; out_tf);

ter_rt : terminator able to send a $PoPS^{rt}$ signal (four terminals: in_pol; out_pol; out_rbs; out_rt);

p_riri : inducible two-operator promoter (five terminals: in_pol, exc_sig_1; exc_tf_1; exc_pol; out_pol);

rbs_n : normal RBS i. e. without any incoming readthrough signal (three terminals: in_pol; exc_r; out_cod);

rep_cod : reporter coding part (three terminals: in_rbs; out_r; out_pol);

ter_n : normal (without readthrough production) terminator (two terminals: in_pol; out_pol);

polpool_2_2 : polymerase pool (four terminals: in_1; in_2; exc_1; exc_2);

ribpool_2_2 : ribosome pool (four terminals: in_1; in_2; exc_1; exc_2);

tf_poo_1_1 : transcription factor pool (two terminals: in_1_tf; exc_1_tf);

sig_pool_1 : environmental signal pool (one terminal: exc_1).

Each part and pool represents a ProMoT module. After loading them, we have to create a new module associated with the whole circuit: select (with the left button of the mouse) the entry "Add Subclass" in the dropdown menu of "module" and call the circuit module **onestep** ("class name").
Now select the new module "onestep" and click the right button: a new dropdown menu appears. Select "Visual Edit" and the canvas will be visualized.
At the beginning, the Visual Editor column "Modules" will be empty. To visualize parts and pools it is sufficient to click on their name in the ProMoT User Interface and drag the appearing icon into the Visual Editor.
It could be useful to vary the canvas size (click "Settings" then "Set Module Size") before starting the circuit design.

You have to drag the icons from the colum "Modules" into the adjacent canvas and drop them here. Then, corresponding terminals have to be connected by means of "wires".
Let us start from the **p_basic** part: connect the *out_pol* terminal with the *in_pol* one on **rbs_rt** and *exc_pol* with *exc_1* which belongs to the polymerase pool. In this way we are implicitly assigning the number 1 to the transcription unit we are building (see Fig. 2.1).

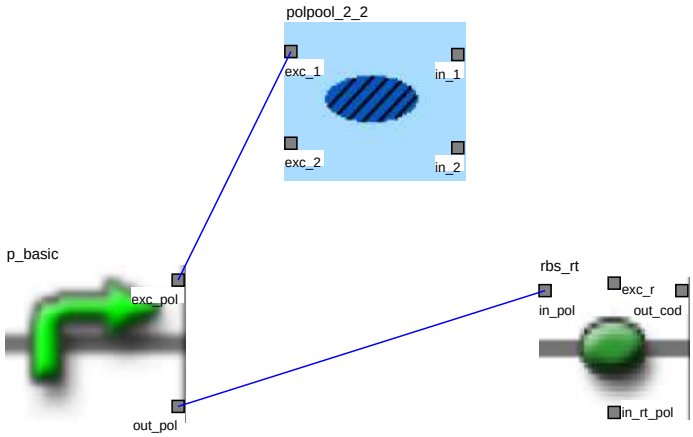

Figure 2.1: **p_basic**, **rbs_rt** and **polpool_2_2** links.

Moving to **rbs_rt**, link *out_cod* and *exc_r* to *in_rbs* on **tf_cod** and to *exc_1* on **ribpool_2_2** respectively. On **tf_cod** you also will connect *out_tf* to *in_1_tf* placed on **tf_pool_1_1**, the transcription factor pool (see Fig. 2.2).
The last terminal belonging to **tf_cod** (*out_pol*) has to be connected to *in_pol* on the terminator **ter_rt**. On **ter_rt** there is also a terminal named *out_pol*: its counterpart lies on the polymerase pool (*in_1*). Connect now the **ter_rt** readthrough terminal *out_rbs* to *in_rt_pol* (obviously on **rbs_rt**, see Fig. 2.3).
Let us place on the canvas the inducible promoter **p_riri** and start building the 2^{nd} transcription unit.
The promoter **p_riri** has several terminals. Connect *in_pol* to *out_rt* on **ter_rt** i. e. to a part belonging to the the 1^{st} transcription unit: in this way you drawn the wire where readthrough polymerases can flow. Terminal *exc_tf_1* has its counterpart on **tf_pool_1_1** (*exc_1_tf*): the transcription factor pool becomes a bridge between the two transcription units where repressors go over to reach **p_riri** after leaving **tf_cod**. Link then *exc_pol* with the polymerase pool (*exc_2*) and *exc_sig_1* with the signal pool **sig_pool_1** through the terminal *exc_1*. The last terminal on **p_riri** (*out_pol*) will then be connected to *in_pol* on **rbs_n** (see Fig. 2.4).
The remaining parts belonging to the second transcription unit (**rbs_n**; **rep_cod** and **ter_n**) do not present any particular terminals so that they can be connencted to each other and to the pools by following the same instructions as for the first transcription unit.
Finally, the one-step-cascade circuit looks like in Fig. 2.5.

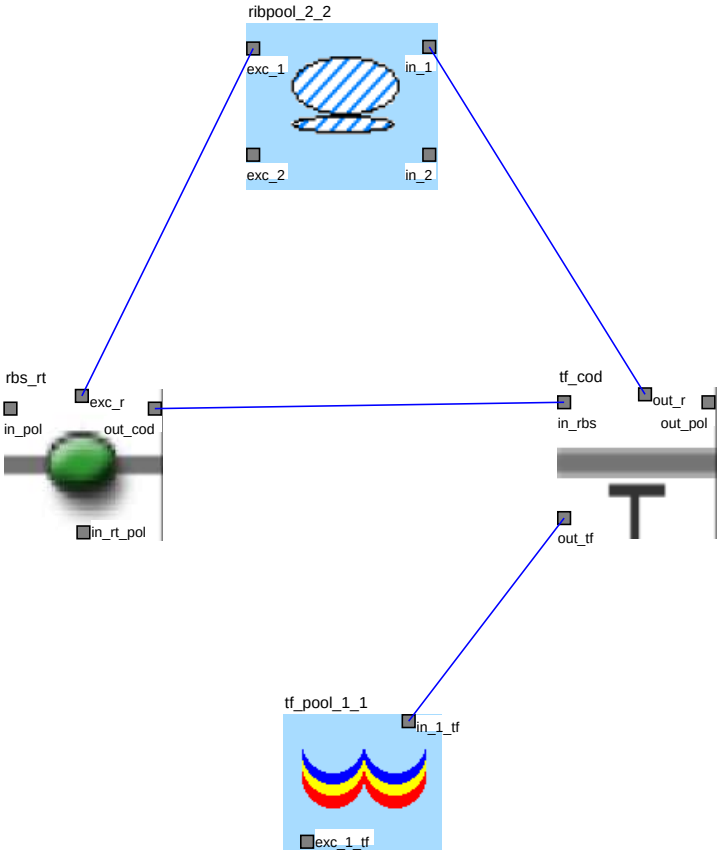

Figure 2.2: **rbs_rt**, **ribpool_2_2**, **tf_cod** and **tf_pool_1_1** links.

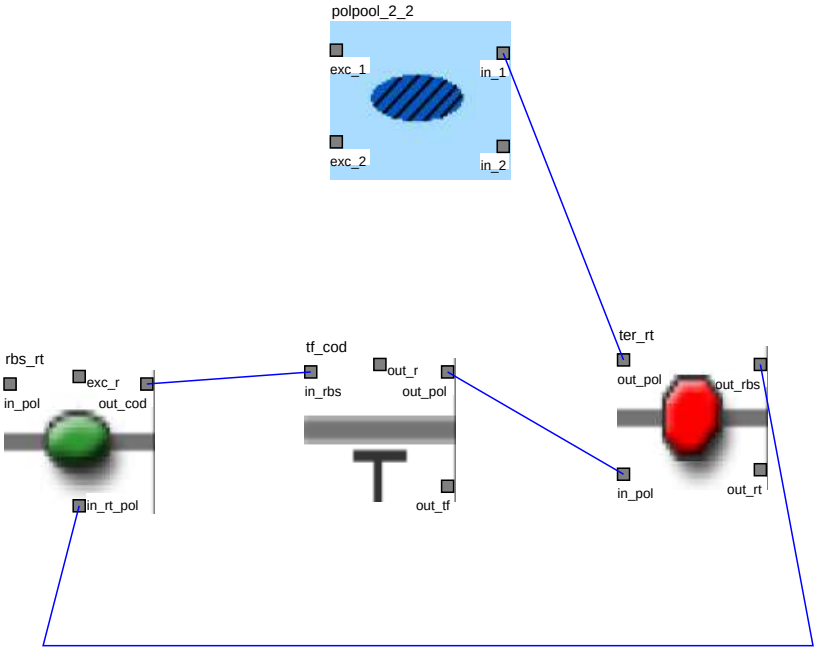

Figure 2.3: **rbs_rt**, **polpool_2_2**, **tf_cod** and **ter_rt** links.

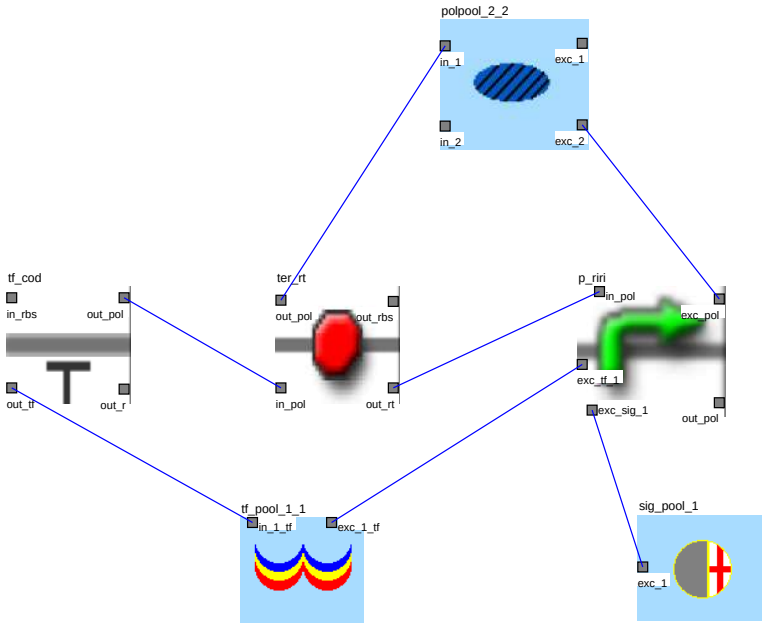

Figure 2.4: **p_riri**, **tf_cod**, **ter_rt**, **polpool_2_2** and **sig_pool_1** links.

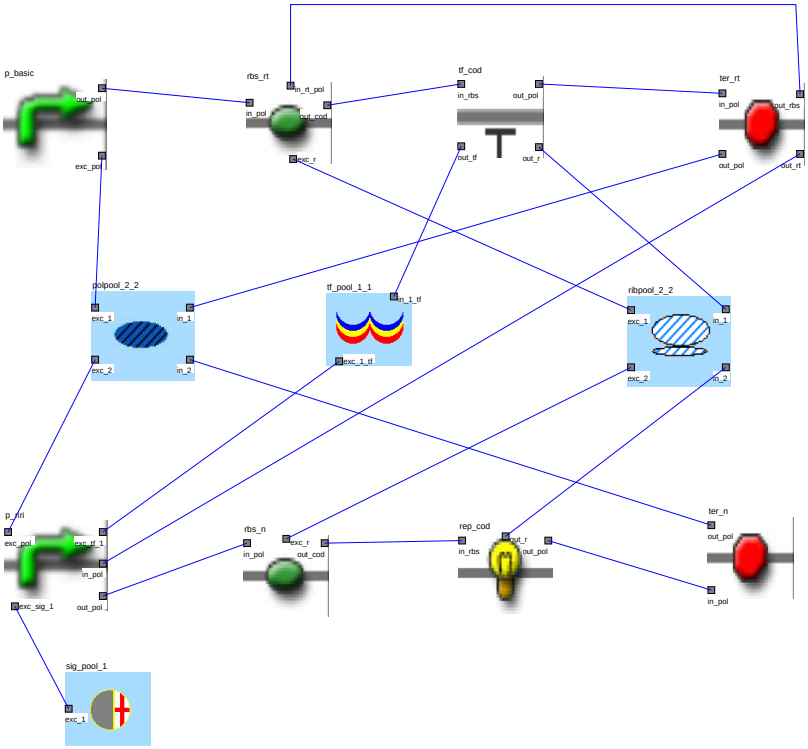

Figure 2.5: The one-step-cascade circuit. All the parts and the pools are shown.

Save the "onestep" module into an MDL file (by clicking on it and then choosing "Save Selected Classes" from the "File" dropdown menu). It could be a good idea also to save all the parts present inside "onestep" into a separate file named, for instance, "parts_and_pools".

2.2 Compact representation

The network design can be simplified by representing each transcription unit as a *device*.
Let us create a new module for each transcription unit, following the procedure described above for the "onestep" module. Let us call them "tu1" and "tu2".
To contruct the first transcription unit device, select "tu1" and open the Visual Editor. Place on the canvas only **p_basic**, **rbs_rt**, **tf_cod** and **ter_rt** then drawn all the possible links among them, as in Fig. 2.6.

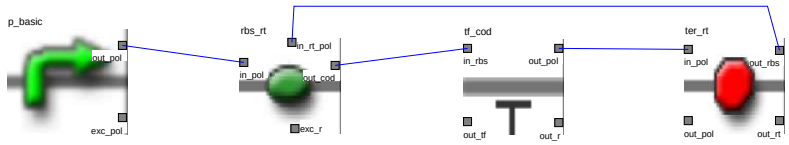

Figure 2.6: Construction of tu1 device: the first step.

All the free terminals will become the device terminals. For instance, *p_basic* has just one free terminal: *exc_pol*, which will be the terminal through which the "tu1" device will exchange polymerases with the corresponding pool. To turn *exc_pol* into a device terminal, select first *p_basic* directly on the canvas and open a new menu just by clicking the right button of the mouse. Select "Terminal Connections": "Propagate exc_pol" will appear. Click on it and the device terminal will be created. Its name will remain *exc_pol*. Repeat this procedure for all the other free terminals and the "tu1" device will be completed (Fig. 2.7). Save then the new module "tu1" into a corresponding MDL file.

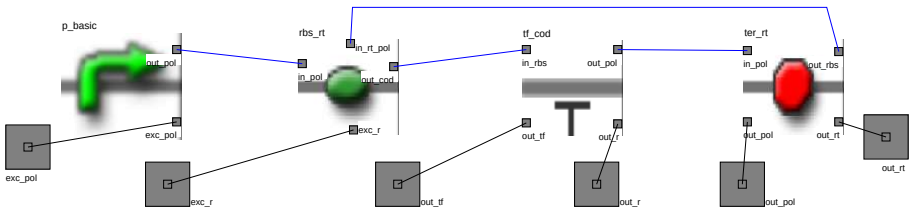

Figure 2.7: Aspect of the completed tu1 device.

The second device "tu2" can be realized in the same way (Fig. 2.8).
The new devices are respectively a protein and a reporter *generator* and have to be associated with the corresponding icons. Select "tu1", click the right button of the mouse and select "Change Icon" from the appearing dropdown menu. In the new window, choose *protein.png* as a desired icon, then click on "Open": now "tu1" has got the right icon. Redo the same for "tu2", just choosing this time *reporter.png*.
Now we can redraw the one-step-cascade network just by means of two devices and four pools.

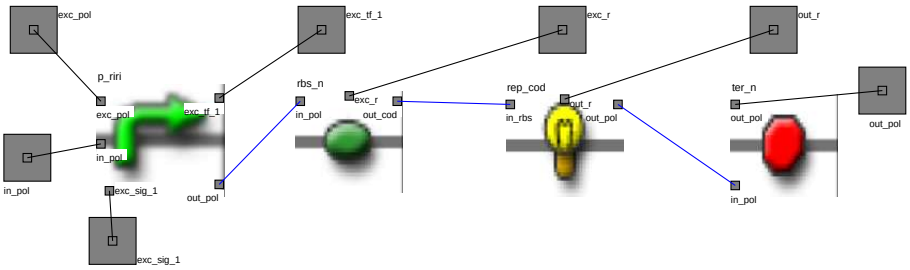

Figure 2.8: Aspect of the completed tu2 device.

Starting from the beginning (no modules are supposed to be present in the ProMoT User Interface), load "parts_and_pools.mdl", tu1.mdl and tu2.mdl then create the new module "newonestep". Select it and go to the canvas. Place here only the pools and the two new devices (Fig. 2.9) then connect the corresponding terminals following the procedure descibed above. Note that the name of the terminals has not been changed (Fig. 2.10)
After saving "newonestep" into an MDL file (not really necessary, but it is better to have a copy of it), it is possible to export the circuit into a format suitable for simulations, like the Matlab or SBML ones.
Select "newonestep" in the "ProMoT User Interface" and then "Export" from the "File" dropdown menu. Another menu will appear: choose "Output to Matlab" or "Export SBML".
If you choose the SBML format (level 2, version 1 by default), the file generated by ProMoT will contain only rate rules and no species or reactions so that it has to be converted into a more "readable" format. In order to do that, you can make use of another perl script: either *parser_lev1.pl* or *parser_lev2.pl* depending on the kind of SBML you want to get. Both the programs take as an input the name of the SBML file generated by ProMoT (without the xml extension).
To run these parsers, open (as above) the Run Script window then choose as the "Directory containing input file" the one where your SBML file (let us call it "newonestep.xml") is stored. Choose as "Script File" one of the two parsers; no "Input File" (i. e. no file with the *inp* extension) is required. As "command line arguments" write (after a blank space) "newonestep". Click on "Generate Model" and then close the window.
Note that *parser_lev1.pl* generates a level 1-version 1 SBML file whereas *parser_lev2.pl* a level 2-version 1 one.
The level 1 version 1 SBML has been tested on Dizzy [Ramsey et al., 2005] and has been written to run stochatic simulations with this software properly.
The level 2 version 1 SBML has been tested on COPASI [Hoops et al., 2006]. It can be used both for stochastic and deterministic simulations and should be readable to any software able to import this version of SBML. The only compartment present in the file has been named "Cell" and its volume (*size*) has been set to $1.6 \cdot 10^{-15} l$.

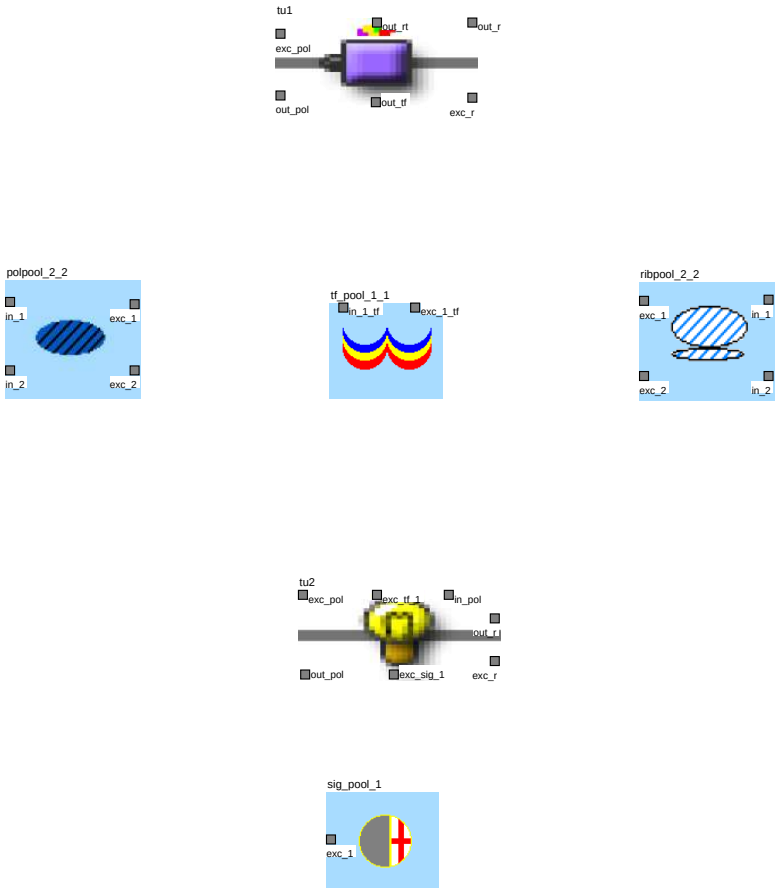

Figure 2.9: Pools and devices necessary to build the one-step cascade.

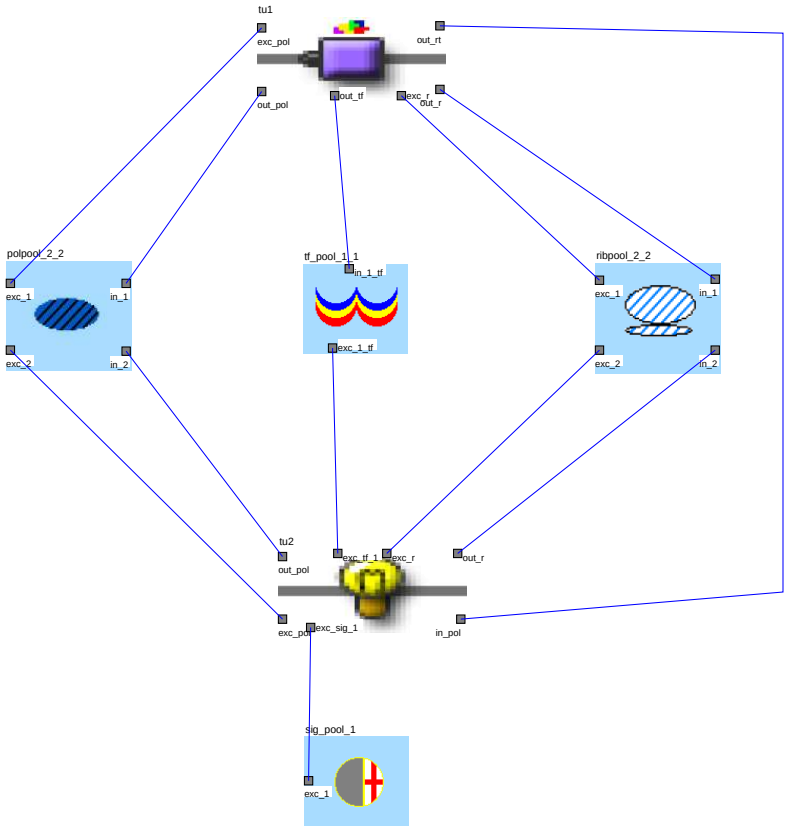

Figure 2.10: Compact version of the one-step cascade.

Acknowledgments

I want to thank Michael Rempel for integrating my code into ProMoT.

Bibliography

[Ginkel et al., 2003] Ginkel, M., Kremling, A., Nutsch, T., Rehner, R., and Gilles, E. D. (2003). Modular modeling of cellular systems with ProMoT/Diva. *Bioinformatics*, 19(9):1169–1176.

[Hoops et al., 2006] Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074.

[Marchisio, 2008] Marchisio, M. A. (2008). *A guide to composable parts and pools.* D-BSSE, ETHZ, Mattenstrasse 26, CH-4058 Basel.

[Marchisio and Stelling, 2008] Marchisio, M. A. and Stelling, J. (2008). Computational design of synthetic gene circuits with composable parts. *Bioinformatics*, 24(17):1903–1910.

[Ramsey et al., 2005] Ramsey, S., Orrell, D., and Bolouri, H. (2005). Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J Bioinform Comput Biol*, 3(2):415–436.