fnyzer: a Python package for the analysis of Flexible Nets

Jorge Júlvez[1][0000−0002−7093−228X] and Stephen G Oliver[2,3][0000−0003−3410−6439]

[1] Department of Computer Science and Systems Engineering, University of Zaragoza, Zaragoza, Spain, julvez@unizar.es
[2] Cambridge Systems Biology Centre, University of Cambridge, Cambridge, UK, sgo24@cam.ac.uk
[3] Department of Biochemistry, University of Cambridge, Cambridge, UK

Abstract. This paper introduces *fnyzer*, a Python package for the analysis of Flexible Nets (FNs). FNs is a modelling formalism for dynamical systems that can accommodate a number of uncertain parameters, and that is particularly well suited to model the different types of networks arising in systems biology. *fnyzer* offers different types of analysis, can handle nonlinear dynamics, and can transform models expressed in Systems Biology Markup Language (SBML) into FN format.

1 Overview

Flexible Nets (FNs) [5] is a modelling formalism for dynamical systems, inspired by Petri [10] nets, that can handle uncertain parameters and that offer different analysis possibilities. FNs have four types of vertices: *places*, *transitions*, *event handlers*, and *intensity handlers*. *Places* are represented as circles and model state variables, e.g. metabolites. The value of the state variable, e.g. metabolite concentration, modelled by a place is called marking. *Transitions* are represented as rectangles and model processes that can modify the marking, e.g. reactions.

In addition to places and transitions, FNs incorporate: a) *event handlers*, represented as dots, that model how the transitions modify the marking; and b) *intensity handlers*, also represented as dots, that model how the marking modifies the speed (or intensity) of the transitions. This way, places and transitions are not connected directly but only through handlers. This connection can be established either by means of *arcs*, which model consumption/production of marking or intensity, or *edges*, which model the use of marking or intensity. In order to account for the relationships "process–marking change" and "marking–speed change", both, event and intensity handlers, are associated with sets of equalities and inequalities.

As an example, the FN in Fig. 1(a) is composed of 3 places A, B, and C (with initial markings 6, 4, and 0 respectively) that are connected through the event handler v by means of arcs. The equalities associated with v determine the stoichiometry of the reaction modelled by the net, namely $a{=}2b$ establishes that two units of A are consumed per each unit of B that is consumed; and $b{=}c$

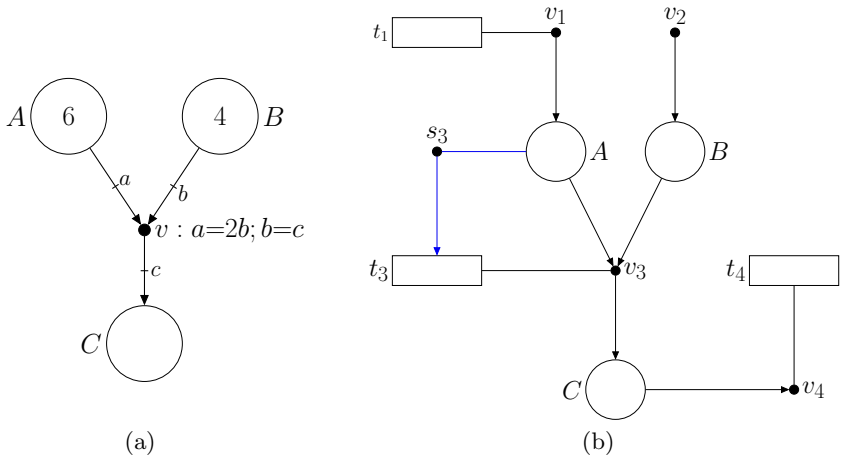


Fig. 1: FNs modelling the stoichiometry of reaction $R : 2A + B \rightarrow C$ (a) and; exchange reactions with partially known dynamics (b).

establishes that one unit of B is consumed per each unit of C that is produced. This way, the FN models the reaction $R : 2A + B \rightarrow C$, and it does not specify a speed as there is no transition in the net. FNs are defined in *fnyzer* by `Python` dictionaries, e.g. the `Python` dictionary that defines the FN in Fig. 1(a) is:

```
stonet = { # FN modelling reaction R: 2A + B -> C
  'name': 'stonet',
  'solver': 'glpk',
  'places': {'A': {'m0': 6}, 'B': {'m0': 4}, 'C': {'m0': 0}},
  'vhandlers': {                          # Event handler
    'v': [{'a': ('A','v'), 'b': ('B','v'), 'c': ('v','C')},
          'a == 2*b', 'b == c']},         # Stoichiometry
  'obj': {'f': "m['C']", 'sense': 'max'},# Objective function
  'options': {'antype': 'un'}             # Untimed analysis
}
```

Thus, in addition to the net structure determined by the keys `places` and `vhandlers`, the dictionary contains information about the objective function and the type of analysis to be carried out by *fnyzer*.

Figure 2 sketches (in FN fashion) the main tasks performed by *fnyzer* in order to analyse an FN. First, a set of mathematical constraints is derived from the FN definition and the desired type of analysis. This set of constraints, which represent necessary reachability conditions, together with an objective function are used to set up a programming problem by using the package Pyomo [4]. This programming problem is solved by a state-of-the-art solver (current supported solvers are CPLEX [1], Gurobi [3] and GLPK [9]) and the obtained solution is saved in a spreadsheet and plotted.

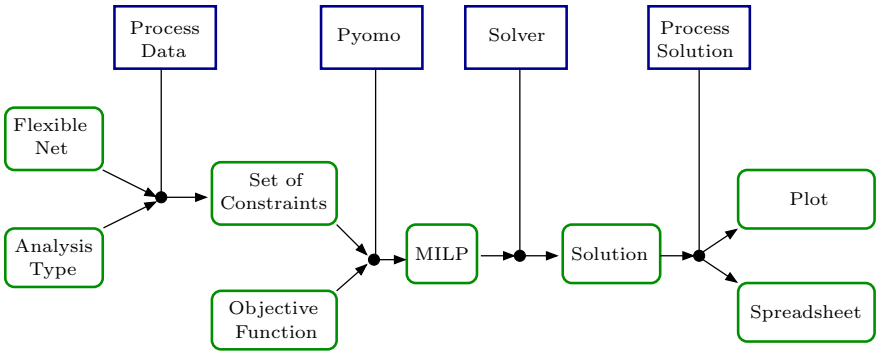


Fig. 2: Main pipeline of *fnyzer* from the input data to the results.

2 Installation and use

fnyzer is an open `Python` package that can be installed with `pip`, the standard tool for installing `Python` packages:

```
$ pip install fnyzer
```

The online documentation of `fnyzer` detailing all the available options can be found at `https://fnyzer.readthedocs.io`, the source code is available at `https://bitbucket.org/Julvez/fnyzer`, and the file `nets/fnexamples.py` in that repository contains a number of FN examples, e.g. the above dictionary `stonet` is in that file.

Assuming that the mentioned file `fnexamples.py` is in your working directory, the execution of:

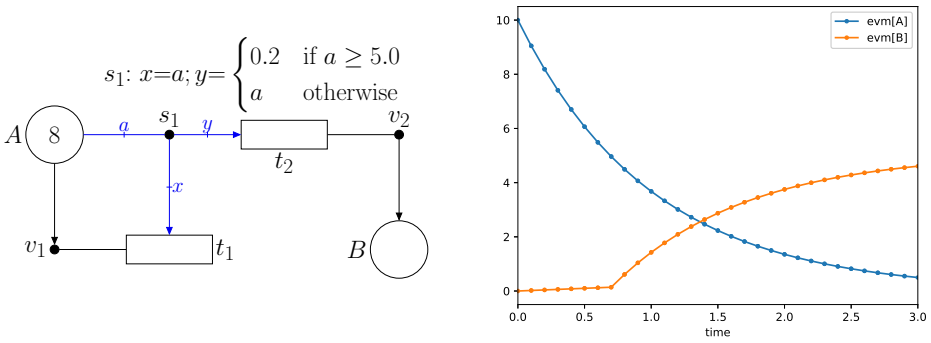
```
$ fnyzer fnexamples.py stonet
```

produces: a) the spreadsheet `stonet.xls` with the optimization results and CPU times; and b) the file `stonet.pkl` with the pickled FN object (see "Accessing saved objects" in the next section). For the proposed objective function, $max\ m[C]$ (i.e. maximize the final concentration of C), the value obtained is 3 (the final concentrations obtained for A and B are 0 and 1 respectively).

3 Main features

Handling uncertain parameters. Assume that the initial concentration of A in Fig. 1(a) is uncertain, but known to be in the interval $[4, 7]$. This can be captured in the `stonet` dictionary above by setting `'A': {'m0': None}` and including a keyword modelling such constraint:

```
'm0cons': ["4 <= m0['A']", "m0['A'] <= 7"]
```



(a) (b)

Fig. 3: (a) Guarded FN modelling the activation of reaction $R_2 : \emptyset \rightarrow B$ when A goes below 5.0; (b) Time trajectories of the concentrations of A and B as plotted by *fnyzer*.

The optimization of the resulting net (execute "`fnyzer fnexamples.py unstonet`") yields 3.5 as the maximum final concentration for C.

Uncertain stoichiometric weights can be incorporated in a similar way. Assume that the reaction is $R : nA + B \rightarrow C$ where n is only partially known, e.g. $n \in [19, 21]$. Such a reaction can be modelled by substituting in the dictionary above `'a == 2*b'` by two inequalities `'19*b <= a', 'a <= 21*b'`.

The FN in Fig. 1(b) models a small reaction network composed of the reactions reported in Table 1. Each reaction R_i is modelled by the event handler v_i (also the intensity handler s_3 in the case of R_3) and the net elements connected to it. The uncertain rates of R_1 and R_4 are accounted for by the constraints:

```
'l0cons': ["1 <= l0['t1']", "l0['t1'] <= 4", "l0['t4'] <= 3"]
```

that are included in the dictionary (see `excnet` in the file `fnexamples.py`) that defines the net. The completely unknown rate of R_2 is modelled by absence of transitions connected to v_2. The known rate of R_3 is modelled by s_3 and t_3.

Handling nonlinear dynamics. The rates of the reactions of the FN in Fig. 1(b) are either constant or depend linearly on the concentration of metabo-

Reaction	Modelled by	Rate
$R_1 : \emptyset \rightarrow A$	v_1	in the interval $[1, 4]$
$R_2 : \emptyset \rightarrow B$	v_2	unknown
$R_3 : A + B \rightarrow C$	v_3 and s_3	equal to the concentration of A
$R_4 : C \rightarrow \emptyset$	v_4	in the interval $[0, 3]$

Table 1: Reactions modelled by the FN in Fig. 1(b).

lites. In order to account for the complex and nonlinear dynamics exhibited by biological systems, FNs can associate piecewise linear functions with their intensity handlers. Consider the FN in Fig. 3(a) which models the reactions $R_1 : A \to \emptyset$ and $R_2 : \emptyset \to B$. The intensity handler s_1 has associated: a) a linear function $x = a$ which determines that the rate of t_1 is equal to the concentration of A; and b) a piecewise linear function $y = \begin{cases} 0.2 & \text{if } a \geq 5.0 \\ a & \text{otherwise} \end{cases}$ which establishes that the rate of R_2 is constant and equal to 0.2 if the concentration of A is greater than or equal to 5.0, and equal to the concentration of A otherwise. In the `Python` dictionary that describes the FN, this is defined by means of two regions (provided by the key `'regs'`), and linear functions associated with them:

```
'regs': {'off': ["m['A'] >= 5"], 'on': ["m['A'] <= 5"]},
'shandlers': {
   's1': [{'a':('A','s1'), 'x':('s1','t1'), 'y':('s1','t2')},
          'x == a',
          {'off': ['y == 0.2'], 'on': ['y == a']}]
},
```

Types of analysis. FNs can be analysed by `fnyzer` under 4 interpretations: untimed [6], transient state [5], Model Predictive Control (MPC) [7], and steady state [8]. Assume that it is desired to compute the maximum concentration of A of the FN in Fig. 1(b) in the steady state. This can be achieved by setting the analysis type to `'antype': 'st'`, and the objective function to:

```
'obj': {'f': "avm['A']", 'sense': 'max'}
```

where `avm` denotes average marking. The value obtained by *fnyzer* is 3.0. At this concentration of A, the flux of all the reactions is 3.0. If the minimum concentration is desired instead, then the `'sense'` of the objective function must be set to `'min'`. The resulting concentration is 1.0. These values were obtained by executing "`fnyzer fnexamples.py excnet`".

As an example of MPC, time trajectories of the concentrations of A and B can be obtained by setting the analysis type to `'antype': 'mpc'`. In the trajectories shown in Fig. 3(b), which were generated by "`fnyzer fnexamples.py guardnet`", 30 time intervals of length 0.1 were considered.

Importing SBML models. In order to facilitate the manipulation of existing models, `fnyzer` offers the possibility of translating `COBRA` [2] models to FNs. Assume that a Systems Biology Markup Language (SBML) model, `MODEL000.xml`, is available in the working directory, then the lines:

```
>>> from fnyzer import optimize, cobra2fn
>>> import cobra
>>> cobra_model = cobra.io.read_sbml_model('MODEL000.xml')
>>> fndic = cobra2fn(cobra_model)
```

convert the model into the dictionary `fndic` that defines the corresponding FN and that can be extended, modified and analysed.

Accessing saved objects. `fnyzer` saves the analysis results in a file that can be easily accessed. For instance, the following lines, read the file `guardnet.pkl` generated by "`fnyzer fnexamples.py guardnet`", save the results in a different spreadsheet, plot the trajectories, and write the concentration of A over time:

```
>>> import pickle
>>> datafile = open("guardnet.pkl", 'rb')
>>> fn = pickle.load(datafile)
>>> datafile.close()
>>> fn.writexls("new_guardnet.xls")
>>> fn.plotres()
>>> [net.places['A'].m for net in fn.lnets]
```

In the above lines, the object `fn` provides access to all the values of the variables in the FN (see the online documentation for details).

4 Acknowledgments

This work was supported by the Spanish Ministry of Science, Innovation and Universities [ref. Medrese-RTI2018-098543-B-I00], by the Biotechnology & Biological Sciences Research Council (UK) grant no. BB/N02348X/1 as part of the IBiotech Program, and by the Industrial Biotechnology Catalyst (Innovate UK, BBSRC, EPSRC) to support the translation, development and commercialisation of innovative Industrial Biotechnology processes.

References

1. IBM ILOG CPLEX Optimizer (2010)
2. Ebrahim, A., Lerman, J.A., Palsson, B.O., Hyduke, D.R.: Cobrapy: Constraints-based reconstruction and analysis for python. BMC Systems Biology **7**(1), 74 (2013)
3. Gurobi Optimization, Inc.: Gurobi optimizer reference manual (2015), http://www.gurobi.com
4. Hart, W.E., Laird, C.D., Watson, J.P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D.: Pyomo–optimization modeling in Python, vol. 67. Springer Science & Business Media, second edn. (2017)
5. Júlvez, J., Dikicioglu, D., Oliver, S.G.: Handling variability and incompleteness of biological data by flexible nets: a case study for Wilson disease. npj Systems Biology and Applications **4**(1), 7 (1 2018)
6. Júlvez, J., Oliver, S.G.: Flexible nets: a modeling formalism for dynamic systems with uncertain parameters. Discrete Event Dynamic Systems **29**(3), 367–392 (2019)
7. Júlvez, J., Oliver, S.G.: Modeling, analyzing and controlling hybrid systems by guarded flexible nets. Nonlinear Analysis: Hybrid Systems **32**, 131 – 146 (2019)
8. Júlvez, J., Oliver, S.G.: Steady State Analysis of Flexible Nets. IEEE Transactions on Automatic Control pp. 1–1 (2019)
9. Makhorin, A.: GLPK (gnu linear programming kit) (2012), http://www.gnu.org/software/glpk/glpk.html
10. Murata, T.: Petri Nets: Properties, Analysis and Applications. Procs. of the IEEE **77**(4), 541–580 (1989)