pypepa Documentation

Release latest

Dariusz Dwornikowski

August 20, 2016

Contents

pypepa is a library and a toolset for Performance Evaluation Process Algebra (PEPA) by Jane Hillston. pypepa is not a fully PEPA compatible tool, it supports a limited (for now) PEPA syntax (we only allow <> operator in system equation), i.e. it does not suport hiding operator (e.g. `P\{a,b,}`), does not calculate passage time and provide model checking. pypepa also does not use Kronecker state space representation and Hillston's aggregation algorithms, so it can have worse performance than the PEPA Eclipse Plugin. All these features, plus more, are planned to be added in future versions.

---

pypepa consist of three parts:

1. `libpepa` - a library written in Python,
2. `pypepa` - a command line tool for solving and graphing,
3. `distr` - map reduce tools for solving large PEPA experiments (not done yet)

CHAPTER 1

News

(18.07.2013) pypepa can now calculate utilisations of components' states, output argument works again

(07.06.2013) Added support for defining rates as mathematical expressions, e.g. r=2*3+7*n;

CHAPTER 2

Documentation

2.1 Installation

pypepa is being developed under Python version 3.3 but it also should work with Python 2.7.

2.1.1 From Package

Using pip:

```
$ pip install pypepa
```

Manually:

1. Clone the project

```
$ git clone git@github.com:tdi/pyPEPA.git pypepa
$ cd pypepa
```

2. Run install

```
$ python setup.py install
```

2.1.2 From the source

For the current version I recommend installing in a virtualenv.

1. Clone the project

```
$ git clone git@github.com:tdi/pyPEPA.git pypepa
$ cd pypepa
```

2. Make a virtualenv

```
$ mkvirtualenv -p /usr/bin/python3 pypepa
$ workon pypepa
```

3. Install all requirements

```
$ pip install pyparsing numpy scipy matplotlib
```

2.2 CLI documentation

2.2.1 Basic arguments

Show help command:

```
$ pypepa -h
```

Set logging level (the default is NONE):

```
$ pypepa --log {DEBUG, INFO, ERROR, NONE}
```

2.2.2 Calculations

Calculate steady state for bank scenario. The putput is by default directed to your terminal.

```
$ pypepa -st models/bankscenario.pepa

Statespace of models/bankscenario.pepa.1 has 7 states

Steady state vector
Using ; delimiter
1;Idle,WaitingForCustomer,WaitingForEmployee;0.08333333333333337
2;Informed,WaitingForCustomer,WaitingForEmployee;0.25
3;WaitingBankResponse,RequestReceived,WaitingForEmployee;0.16666666666666666
4;WaitingBankResponse,CustomerNotReliable,WaitingForEmployee;0.16666666666666666
5;WaitingBankResponse,CustomerReliable,WaitingForEmployee;0.16666666666666666
6;WaitingBankResponse,WaitingManagerResponse,EvaluatingOffer;0.08333333333333333
7;OfferReceived,WaitingForCustomer,WaitingForEmployee;0.08333333333333333
```

Calculate actions' throughput:

```
$ pypepa -th models/bankscenario.pepa

Statespace of models/bankscenario.pepa.1 has 7 states

Throuhoutput (successful action completion in one time unit)

readInformation                         0.08333333333333337
createLoanRequest                             0.25
getNotReliableMessage                   0.16666666666666666
badOffer                                0.08333333333333333
askManager                              0.16666666666666666
reset                                   0.08333333333333333
goodOffer                               0.08333333333333333
checkReliability                        0.3333333333333333
```

You can calculate transient time proability for some number of time steps:

```
$ pypepa --transient 5 models/bankscenario.pepa

Transient analysis from time 0 to 10

Using ; delimiter
1;Idle,WaitingForCustomer,WaitingForEmployee;0.08351202761947342
2;Informed,WaitingForCustomer,WaitingForEmployee;0.2500169897974121
3;WaitingBankResponse,RequestReceived,WaitingForEmployee;0.16662129023697114
```

```
4;WaitingBankResponse,CustomerNotReliable,WaitingForEmployee;0.16657721277634494
5;WaitingBankResponse,CustomerReliable,WaitingForEmployee;0.16657721277634485
6;WaitingBankResponse,WaitingManagerResponse,EvaluatingOffer;0.08328947039778702
7;OfferReceived,WaitingForCustomer,WaitingForEmployee;0.08340579639566591
```

You can choose a solver by specifying `--solver|-s {direct, sparse}`. By defalt we use sparse solver with LIL matrix becuase it is faster and in overall matrices generated from PEPA models are sparse. There is also an insignificant difference in results.

pypepa allows you to visualise all PEPA components and the whole state space of a model by specifying `-gd` switch. The generated graphiz dot files are by deault saved in `dots` folder in the current directory. You can browse dot files with `xdot`, which you need to install first.

```
$ pypepa -gd bankdots models/bankscenario.pepa
```

Finally pypepa can provide us with a tool for experimentation with rates and actions. Let's check how throughtput of `askManager` action changes when `rateReset` changes from 1 to 50 with step 1. The default result of this command will be a matplotlib graph. The format of `-var` is "vartype:varname:value range specifier:value range value". The one valid vartype for now is `rate`, for value range specifiers you can choose: `range` or `list`. For `range` you need to provide START, STOP, STEP, whereas for `list` a comma separated list of values. You can specify other output options with `-f` argument: graph, console, csv.

```
$ pypepa -var "rate:rateReset:range:1,50,1" -val askManager  models/bankscenario.pepa
```

2.2.3 Formatting

You can specify formats of `-st`, `-th` and `--varrate` with a `--format` option. Currently we support CSV (although *;* not comma delimited), console (the default) and graph (only for varrate experiments). Additionally you can specify `-o|--output` option with a file argument to specify where to save the CSV.

```
$ pypepa -st models/bankscenario.pepa -f csv -o bank_steady.csv
```

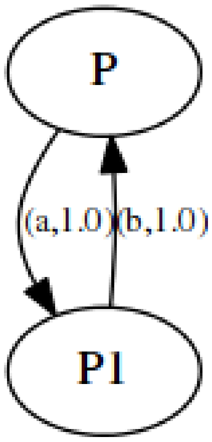
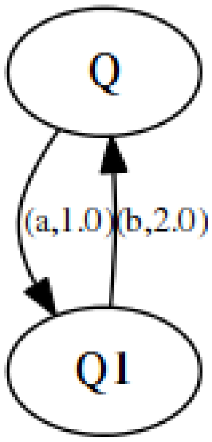
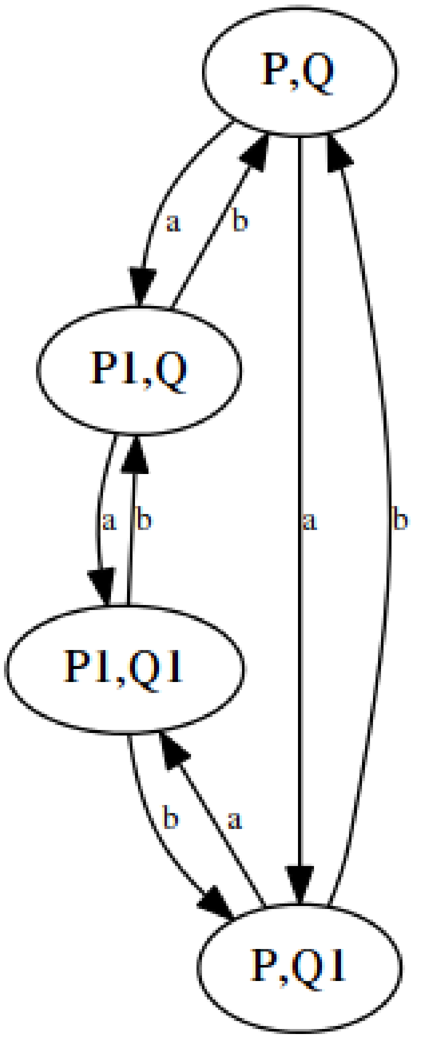
The command will output a `bank_steady-steady.csv`, analogically for utilisation it will be `-utilisation` postfix and for transient analysis `-transient`

2.2.4 Generating state space graphs

By specifying `-gd|--gendots DIR` you tell pypepa to generate dot files for graphiz in a directory DIR. Dot files can be processed by graphiz package or displayed more interactively using `xdot` package that can be installed from PyPI (`pip install xdot`).

```
$ pypepa -gd dots tests/simple.pepa
```

This command will generate a dot file representing state space of each component, as well as for of the whole state space. Below you can see an exemplary output:

Component P	Component Q	Whole state space
		 P,Q a b P1,Q a b a b P1,Q1 b a P,Q1

2.3 Using libpepa library

Large part of pypepa is libpepa library which can be used to embed PEPA calculations in your Python project. The main entry point to libpepa is PEPAModel class.

2.3.1 PEPAModel object

PEPAModel object needs be provided with keyword arguments:

- `name` - it is an optional argument. If it is not given, it will be become the basename of the `file`, otherwise it will default to `model` (if modelstring is given)
- `file` - a path to a file with a PEPA model definition
- `modelstring` - a string with a PEPA model defintion
- `solver` - either `sparse` or `direct`, default is `sparse`

Basic usage:

```
from pypepa import PEPAModel

pargs = {"file": "tests/simple.pepa"}
pm = PEPAModel(**pargs)
pm.derive()
```

After that the PEPA model is derived. Now, in order to make some calculations you need to use corresponding methods from `PEPAModel` class.

Steady state calculation

```
pm.steady_state()
vector = pm.get_steady_state_vector()
```

In this case vector is a list of steady state probabilities for states [0..n].

Throughput

```
pm.steady_state()
thr = pm.get_throughput()
```

Here, `thr` will be a list of tuples `(action, throughput)`

Transient time analsis

```
vector = pm.transient(0, timestop)
```

Vector is list of probabilities.

Utlisations

```
pm.steady_state()
usabilities = pm.get_utilisations()
```

Here `usabilities` is a list of Counter objects, each being a dict, corresponding to a component in a model, where keys are state names and values are usabilities. Example:

```
[Counter({'P1': 0.5, 'P': 0.49999999999999989}), Counter({'Q': 0.66666666666666652, 'Q1': 0.33333333
```

Generating dots

```
pm.generate_dots(out_dir=path)
```

This method will generate dots in a directory specified by `path`. If this argument is not supplied, by default dots will be generated in `dots` directory.

Additional methods

get_state_names()

Returns a list of state names used for matching state number with a correponding name. Index of the list is the number of a state.

get_rates()

Returns a dict with rate names mapped to rate values.

2.4 Credits

pypepa is being developed in the Institute of Computing Science, at Poznań University of Technology. The project is led by Dariusz Dwornikowski.

Credits and thanks:

```
    * Allan Clark
    * Jan Lamecki
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index