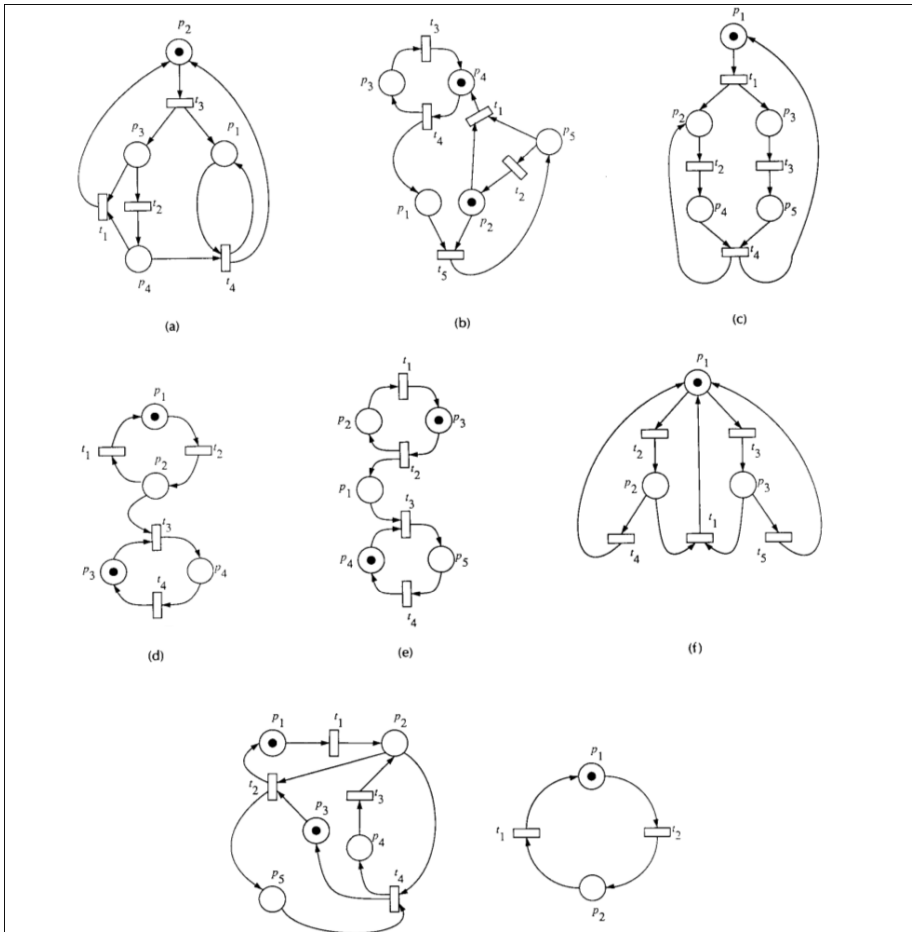
Petri-Nets

June 12, 2025

1 Liveness, Boundedness, and Reversability

For the following 8 Petri nets, determine if they are i.) bounded, ii.) live, iii.) quasi-live, and iv.) reversible.

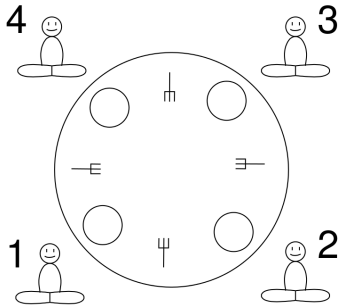


2 Modeling With Petri-Nets — `LoLA`

Please note, that all the tools used for this exercise are freely available. The command line tool `lola` is installed in the docker image provided, alternatively you can download, compile and install lola from `https://theo.informatik.uni-rostock.de/theo-forschung/tools/lola/`).

For details wrt. the syntax of all the commands given in this exercise, all details (and much more) can be found in the `LoLa` documentation.

Four philosophers sit around a round table.



There are forks on the table, one between each pair of philosophers. The philosophers want to eat spaghetti from a large bowl in the center of the table. Unfortunately the spaghetti is of a particularly slippery type, and a philosopher needs both forks in order to eat it.

a) The philosophers have agreed on the following protocol to obtain the forks: Initially philosophers think about philosophy, when they get hungry they do the following: (1) take the left and the right fork simultaneously and start eating, (2) return both forks simultaneously, and repeat from the beginning.

   a1) Make a Petri net model of the described situation. (In case you get stuck during the modelling or want to verify your modelling: an illustration can be found here.)

   a2) Make an input file for `LoLA` based on your Petri net model. Here is a trivial template for the syntax:

```
PLACE

p1, p2, p3;

MARKING
     p1 : 1,
     p2 : 2;

TRANSITION tr.1
     CONSUME
```

```
            p1 : 1,
            p2 : 1;
      PRODUCE
            p3 : 1;

TRANSITION tr.2
      CONSUME
            p3 : 1;
      PRODUCE
            p1 : 1;
```

a3) Check if 2 philosophers sitting next to each other can eat simultaneously. You should use a command line similar to

`lola --formula="EF (ea1 > 0 AND ea2 > 0)" phils.lola`

where `ea1` and `ea2` refer to places in your Petri net (you might have named them differently).

a3) Check if 2 philosophers sitting *not* next to each other can eat simultaneously.

a4) A transition is live if it not dead in any reachable marking. Are there any transitions in your modelling that are not live? Use the option `--formula="AGEF FIREABLE(t)"` (see Section 4.2.10 in the LoLa documentation) page in order to check this for a transition `t`. In case you find a transition that is not fireable, use `--path=witness` in order to create a file called `witness` that includes the sequence of transitions that proofs that a transition is not fireable.

a5) Is your Petri net deadlock free? (Use `--formula="EF DEADLOCK"`) in order to check for this property. If you find a deadlock, use the option `--path=witness` in order to store a proof, i.e., a sequence of firing transitions in order to find a sequence that proofs that a deadlock situation can occur.

b) Assume now that the philosophers have agreed on the following protocol to obtain the forks: Initially philosophers think about philosophy, when they get hungry they do the following: (1) take the left fork and wait for the right fork becoming available, (2) take the right fork start eating, (3) return both forks simultaneously, think, and repeat from the beginning. Analyse your Petri net stepwise similar to a). Will an individual philosopher eventually eat, assuming she wants to?

c*) (This is very similar to b) Assume now that the philosophers have agreed on even another protocol to obtain the forks: Initially philosophers think

about philosophy, when they get hungry they do the following: (1) take the left fork and wait for the right fork becoming available, (2) take the right fork start eating, (3) return the left fork, (4) return the right fork and start thinking again, and repeat from the beginning. Analyse your Petri net stepwise similar to a). Will an individual philosopher eventually eat, assuming she wants to?

d) Assume you would use a Petri net to model a chemical system, where the places correspond to chemical entities and the transitions correspond to chemical reactions. What would a dead transition indicate?

3 Modeling With Petri-Nets — `MonaLisa` - Still No Chemistry

MonLisa can be found here:
`http://www.bioinformatik.uni-frankfurt.de/tools/monalisa/`.
Start MonaLisa with
`java -jar ./MonaLisa.jar`.

a) Determine (not using any other tool than your brain) the P-inariants and the T-invariants for the Petri nets modelled in the exercise before.

b) Create in Petri net in `MonaLisa` for the 4 philosophers problem which follows the modelling of exercise 1b), i.e., each philosopher can be in one of three states (thinking, has-left-fork, eating). Determine the P-invariants and the T-invariants; how many of each kind did you find? Give an interpretation for all the invariants. Verify all the invariants with `MonaLisa`.

c) In the simulation mode (asynchronous simulation), simulate a firing sequence such that you end up in a deadlock situation.

d) Simulate a firing sequence such that you end up in a deadlock situation.

e) Do a knock-out analysis for one of your networks. Try to do a *single knock-out analysis for all species* and a *single knock-out analysis for all reaction*, export your results and give an interpretation of your findings.

4 Modeling With Petri-Nets — `MonaLisa` - A Biological / Chemical System

Background information for this exercise is given in:

Application of Petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber, Ina Koch, Björn H. Junker, and Monika

Heiner, *Bionformatics*, Vol. 21 no. 7, 2005, pages 1219–1226, doi: 10.1093/bioinformatics/bti145

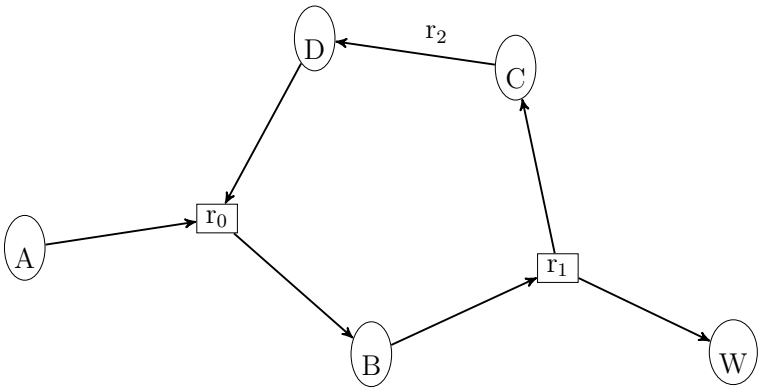
Start MonaLisa and load the project `potato.mlproject`. In the NetViewer view you will see the Petri net model of the sucrose-to-starch metabolism in the potato tuber.

a) Use `MonaLisa` to identify all P-inariants. Without having a biology/chemistry background knowledge: can you give a biological interpretation? Do the same for the T-invariants.

b*) Use `MonaLisa` to do a *single knock-out analysis for all species* and a *single knock-out analysis for all reaction* and export your results. In case you by chance know about biology / chemistry: give an interpretation of your result, if not, you can safely ignore this question.

5 Modeling With Petri-Nets — Using `MØD` - A Simple Abstract Chemical System

Given is the following abstract chemical network. Note that square nodes correspond to hyperedges (i.e. chemical reactions).



a) Assume you have in addtion to the 3 given reactions a pseudo reaction for `A` in order to model inflow to the network, and a pseudo reaction for `W` in order to model outflow to the network. What is the stoichiometic of the simple network given above. What are the P-inariants and the T-invariants of this network?

b Assuming an inflow to `A` of one unit, give a hyperflow that maximizes the outflow of `W`.

c) With the following piece of (Python) code you can create an abstract derivation graph (i.e. a chemical network) of the network:

```
str="""
A + D -> B
B -> C + W
C -> D
"""

dg = DG()
dg.build().addAbstract(str)
```

Use MØD to find a hyperflow through the network. Having *only* one unit of `A` (and nothing else) available, can an instance of `W` be created? Assuming you would have in addition eiter one instance of `B`, `C`, or `D` available, can then a `W` be created?

d) Use the provided code in order to find catalysts, that make the the creation of `W` *realizable*.

6 Modelling with MØD - Using the Petri Net features - The Non-oxidative phase of the Pentose phosphate pathway (PPP)

a) Reconsider the PPP Pathway from practical assignment 1. What you are provided in the scripts is a graph grammar, different methods to expand the derivation graph including a rather strict "strategy" to expand the chemical space based on the grammar. In order to get a sense for the combinatorical explosion of the chemical space, replace your strategy to expand the chemical space with a strategy that applies *all* rules repeatedly three times.

```
dg = DG(graphDatabase=inputGraphs)
dg.build().execute(
        addSubset(ribuloseP, water)
        >> repeat[steps](inputRules)
)
dg.print()
```

In the following, better use `dgStrict.py` to expand the space.

b) Open `dgStrict.py`, `pathway.py`, and `printStuff.py`. It will allow to expand the chemical space, find a (hyper-)flow and print it. Add the line

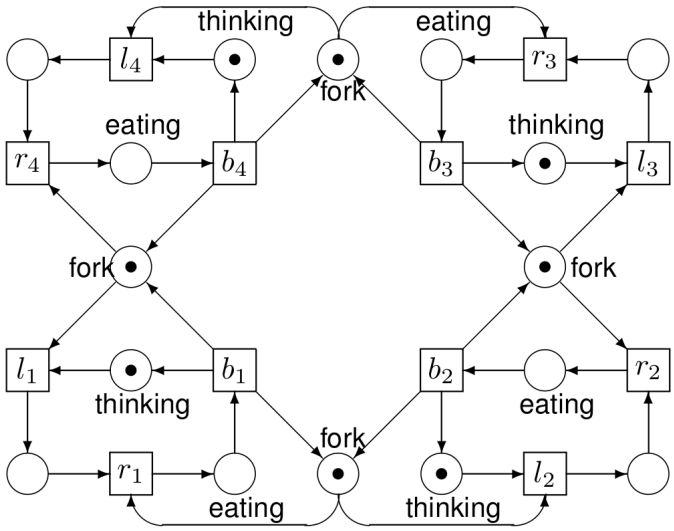
```
flow.setSolverEnumerateBy(maxNumSolutions=10)
```

just before `flow.calc()` in order to create 10 flow solutions now. Run the program and wait a while, a PDF that includes all 10 solutions should be created. In the following we will analyse these 10 solutions identical to the procedure in Exercise 4.

c) For this part, please use the command line version of `mod`. Use the following command line

```
mod -f dgStrict.py -f pathway.py -f petriNet.py
```

in order to do this analysis. We will discuss the details of `petriNet.py`, but superficially all flow solutions are analysed wrt. their realizabilty. In case that a solution is not realizable, and of the chemical compounds in the chemical space is tried as a catalyst for the realization of the overall reaction.


thinking
eating
l_4
r_3
fork
eating
thinking
r_4
b_4
b_3
l_3
fork
fork
l_1
b_1
b_2
r_2
thinking
eating
fork
r_1
l_2
eating
thinking