Chapter 7

Solutions to Exercises

Mary Ann Blätke[1], Monika Heiner[2] and Wolfgang Marwan[1]
[1]*Otto-von-Guericke University, Magdeburg, Germany,* [2]*Brandenburg University of Technology, Cottbus, Germany*

7.1 PETRI NETS ($\mathcal{PN}$)

Exercise 7.1 (Drawing a Petri Net)**.** To draw your first Petri net, repeat the following steps.

1. Create a new net (*File* → *New*) and choose the appropriate net class. You will get a new drawing window with the name "*unnamed*."
2. Select the graph element *Place* in the menu panel on the left-hand side, and left-click on the drawing window at those positions where you want to get a place. Each click creates a new place.
3. Similarly, select the graph element *Transition* in the menu panel, and create transitions.
4. To connect two nodes, select *Edge* (which is just a synonym for *arc*) in the menu panel, click on the source node and move the pointer, while keeping the mouse button pressed, to the target node, where the mouse button is released.
5. A double click on a graph element (place, transition, edge) opens an attribute window, which allows you to edit the element-specific properties.
6. Finally, do not forget to save your work under a new name (*File* → *Save as*).

To familiarize yourself with the basic editing features, draw the $\mathcal{PN}$ in Figure 7.3.

Solution. Self-explanatory; the solution is part of the Supplementary Material of the book chapter. □

Exercise 7.2 (Drawing a Hierarchical Petri Net)**.** To hierarchically structure a Petri net, repeat the following steps.

1. Draw your flat net (or portion of it) as you wish to have it.
2. Select the subgraph that you want to be abstracted by a macro node, and select *Hierarchy* → *Coarse*. Choose the appropriate coarse element (macro place, macro transition) and hit the OK button.
3. A double click on the macro node opens its attribute window and allows you to assign a suitable name; a click on the entry with this name in the hierarchy panel on the left-hand side opens the subgraph in a separate window. The blue net parts have been automatically generated and represent the connection of the subgraph (macro node) with the neighboring nodes on the next higher hierarchy level.
4. Finally, do not forget to save your work under a new name (*File* → *Save as*).

To familiarize yourself with this feature, construct the hierarchical Petri net in Figure 7.7, left column.

Solution. Self-explanatory; the solution is part of the Supplementary Material of the book chapter. □

Exercise 7.3 (Petri Net Simulation)**.** Snoopy visualizes the token flow, so you can conveniently explore the net behavior. To animate a Petri net, follow these steps.

1. Open the net, and open the animate window (*View* → *Animation Mode*).
2. A left click on a place increases the token number by 1; a right click decreases the token number by 1.
3. *Manual mode*: A left click on a transition triggers the firing of this transition, if it is enabled.

Algebraic and Discrete Mathematical Methods for Modern Biology. http://dx.doi.org/10.1016/B978-0-12-801213-0.00026-5

4. *Automatic mode*: Clicking directly on the control panel in the animation window leaves the decision of which transition to fire to the simulation/animation algorithm.

The set of all enabled transitions is determined in each step of the automatic mode. There are three strategies to choose the transition(s), among all enabled transitions, to be fired in the next execution step.

(a) *Single step*: one single transition is randomly chosen.

(b) *Intermediate step*: an arbitrary subset of concurrent transitions is randomly chosen.

(c) *Maximal step*: a maximal set of concurrent transitions is randomly chosen.

Explore these three strategies for the running example and find for each strategy a firing sequence where each transition fires at least once. To make it a challenge, try to find short sequences (in terms of number of steps).

Hint. Steps just done can be played backward up to a depth of 10. This default value can be changed in the *Global Preferences* dialog.

Solution. The main objective of this exercise is to learn the differences between the three firing strategies and to understand why we will later use single-step firing for all analysis techniques.

Students should be able to solve this exercise without tool support, but one can check answers with the help of Snoopy's animation, or even generate (not necessarily short) solutions, because Snoopy supports the recording of simulation sequences run in manual or automatic mode; see *Export* in the animation window. These traces can then be imported again to trigger specific simulation runs for demonstration purposes of specific scenarios.

Subtasks (b) and (c) need to decide how to deal with read arcs. In the standard interleaving semantics, read arcs are behaviorally equivalent with two opposite arcs; that is, they create a conflict. However, they could also be treated as not causing conflicts, which then requires us to fire, for example, $r5$ and $r6$ in one maximal step, if there is one token on *mrnaA*.

Obviously, there are many solutions, even with the constraint of possibly short sequences. The following traces are not all minimal (in terms of their length); check the Supplementary Material to repeat these sequences with the help of Snoopy's feature to import transition sequences.

(a) *Single step*: We start with the observation that $r5$ has to fire twice to supply $r7$ and $r8$ with tokens; all other transitions may fire only once. Thus, as there are 16 transitions, a minimal sequence has a length of 17 single firing steps. Due to concurrency, there are several interleaving sequences. We give one of them:

$r3, r5, r1, r4, r2, r6, r11, r9, r12, r10, r14, r13, r8, r16, r15, r5, r7.$

(b) *Intermediate step*: Special cases would be the single-step sequence and the maximal step sequence; one other possible run is

$r3 + r11,$
$r5 + r6 + r13 + r14,$
$r1,$
$r4 + r11,$
$r2,$
$r9,$
$r3 + r12,$
$r3 + r6 + r10,$
$r6 + r8 + r14,$
$r5 + r16,$
$r7 + r15.$ (11 steps)

(c) *Maximal step*: This firing sequence assumes that read arcs do not create conflicts; that is, we always get the following transitions in pairs: $(r2, r4), (r5, r6), (r10, r12), (r13, r14)$:

$r3 + r11,$
$r3 + r5 + r6 + r11 + r13 + r14,$
$r1 + r5 + r6 + r13 + r14 + r15,$
$r2 + r4 + r9 + r15,$
$r3 + r5 + r6 + r7 + r10 + r12,$
$r3 + r5 + r6 + r7 + r14 + r13,$

$r3 + r5 + r6 + r7 + r8 + r11$,
$r3 + r5 + r6 + r9 + r13 + r14 + r16$. (8 steps)

An option to increase the difficulty of this task is to specify a target state that should be reached by the firing sequence. Such target states need to be chosen with great care, as the set of reachable states generally depends on the firing strategy.

Another possible extension of this exercise is the orthogonal feature *auto-concurrency*: in one step, a transition may fire to itself concurrently if there is a sufficient amount of tokens to enable it to multiply (the enabling degree is higher than 1). Technically speaking, auto-concurrency fires multisets of transitions in each step. The standard Petri net behavior does not involve auto-concurrency.

Auto-concurrency may be helpful to mimic mass-action kinetics in a purely qualitative manner. □

Exercise 7.4 (Walking Through the State Space)**.** Explore the state space of the running example by playing the token game. Try to answer the following questions.

(a) Is it possible to reach a state with exactly one token on every place? Prove your answer by a witness firing sequence, or explain the nonreachability.

(b) Is the state space finite or infinite? Try to figure out the maximal token numbers you can get on each place. Try to characterize the set of reachable states by a regular expression.

(c) Having played with the net for a while, is it always possible to reach again the given initial state? Explain your answer.

(d) A state in which no transition is enabled is called a *dead state*. Are there reachable dead states for the given initial state? Explain your answer.

Solution. Doing this exercise at this point, as suggested, should help you get a feeling for the different behavioral properties a $\mathcal{PN}$ may exhibit; it is meant to prepare you for the next section ($\mathcal{PN}$ analysis).

Playing with a Petri net usually helps us understand its behavior. Then it should not be tricky to find the following answers. The explanations given in *brackets and italics* use terminology that is introduced later. So, the kinds of answers to expect depend somewhat on the point at which this exercise is actually done.

(a) *Exactly one token on every place*: No, this state is unreachable. It is impossible to get *geneA* and *geneA_A* marked at the same time; likewise for *geneR* and *geneR_A*. The following firing sequence marks all other places with exactly one token: $r3, r5, r5, r11, r13, r13, r8$.

(b) *Finite/infinite state space*: The state space is obviously infinite; for example, with a token on *geneA*, the transition $r3$ can arbitrarily often fire, generating a possibly infinite number of tokens on *mrnaA*.

There is no way to get more than one token on *geneA*, *geneA_A*, *geneR*, *geneR_A*, while all other places can get an arbitrary number of tokens.

[The places geneA, geneA_A, geneR, geneR_A are 1-bounded, while all other places are unbounded; that is, there is no upper bound for the number of tokens they may get.]

Let's assume the following arbitrarily chosen order of places:
geneA, $geneA_A$, *geneR*, $geneR_A$, *mrnaA*, *mrnaR*, *A*, *R*, *A_R*.

Then, a state can be written as an integer vector of length 9; for example,
the initial state is $(1, 1, 0, 0, 0, 0, 0, 0, 0)$.

The set of all reachable states in vector notation is given by the following regular expression:
$([0, 1], [0, 1], [0, 1], [0, 1], [0, n], [0, n], [0, n], [0, n], [0, n])$, with $n \in \mathbb{N}$,
or even shorter $(\{[0, 1], \}^4\{[0, 1], \}^4[0, 1])$.

Hint. Curly and square brackets are part of the metalanguage; $[0, 1]$ means "either 0 or 1"; $[0, n]$ means "one value out of the range 0 to n" (including the boundaries).

In words: The genes *A* and *R* can be independently free or bound to the activator protein, while the token number on all other places can arbitrarily vary, or even go to infinity.

(c) *Going back to the initial state*: Yes, that's always possible, because $r1$ is counterbalanced by $r2$, likewise for $r9$ and $r10$, and all generated tokens can always be removed again, for example, $r3$ is counterbalanced by $r6$.

[*The Petri net is reversible.*]

(d) *Dead state*: No, there are no dead states, which immediately follows from the set of reachable states: in each state, there is at least one enabled transition. □

Exercise 7.5 (Improved Petri Net Simulation)**.** Algorithm 1 gives the very basic procedure for simulating a Petri net. This naive algorithm assumes that there is at least one enabled transition in each step, otherwise the algorithm would encounter a dead state—a state where none of the transitions is enabled. Improve this pseudocode with the following features.

(a) The algorithm terminates upon encountering a dead state.

(b) The algorithm allows you to choose between single/intermediate/maximal step firing; see Exercise 7.3.

Solution. We deliberately treat the algorithmic extensions separately to keep the pseudocode readable. Of course, both extensions could be combined.

(a) *Recognition of dead states*: See Algorithm 1; the new lines are 1, 7-11, 15.

ALGORITHM 1 $\mathcal{PN}$ **simulation (animation) algorithm with dead state recognition.**

Require: $\mathcal{PN}$ with initial state s_0;
number of steps N;

```
 1: transitionSet ET                          ▷ ET—set of enabled transitions
 2: integer count ← 0
 3: state s ← s_0                             ▷ make initial state to current state
 4: write(count, s)                           ▷ add s_0 to trace
 5: while count < N do
 6:   count ← count + 1
 7:   ET ← enabled(s)                         ▷ compute set of transitions enabled at s
 8:   if ET = ∅ then
 9:     write(dead state)
10:     exit                                 ▷ terminate algorithm
11:   else
12:     pick one transition t ∈ ET           ▷ random choice of the single transition to fire
13:     s ← fire(s, t)                       ▷ compute new state s by firing of t
14:     write(count, s)                      ▷ add s to trace
15:   end if
16: end while
```

(b) *Selection of firing strategy*: See Algorithm 2; the new lines are 1-2, 8-16. □

Exercise 7.6 (Orthogonality of Behavioral Properties)**.** The basic Petri net properties of boundedness, liveness, and reversibility are orthogonal, that is, independent from each other.

(a) Prove this statement by providing a net as small as possible for each possible combination (obviously, there are eight of them). Argue for every net why the properties hold as you claim.

If you find this too easy, repeat the exercise with the additional constraints that the nets should be

(b) ordinary

(c) ordinary and pure

(d) ordinary, pure, and strongly connected.

Solution. The motivation for this exercise is mainly to get a feeling for the net behavior induced by a given structure, and of course to gain the experience that the orthogonal properties are indeed independent from each other. Make it a competition: Who can find the smallest nets (in terms of number of nodes + arcs)?

ALGORITHM 2 $\mathcal{PN}$ **simulation (animation) algorithm with firing mode selection.**

Require: $\mathcal{PN}$ with initial state s_0;
number of steps N;
firingMode $mode \in \{single, inter, max\}$

```
1:  transitionSet ET                                  ▷ ET—set of enabled transitions
2:  setOfTransitionsSet ETS                           ▷ ETS—set of enabled transition sets
3:  integer count ← 0
4:  state s ← s_0                                     ▷ make initial state to current state
5:  write(count, s)                                   ▷ add s_0 to trace
6:  while count < N do
7:    count ← count + 1
8:    if mode = single then
9:      ETS ← enabledSingle(s)                        ▷ get all single sets of enabled transitions
10:   else if mode = inter then
11:     ETS ← enabledInter(s)                         ▷ get all intermediate sets of enabled transitions
12:   else if mode = max then
13:     ETS ← enabledMax(s)                           ▷ get all maximal sets of enabled transitions
14:   end if
15:   pick one transition set ET ∈ ETS                ▷ random choice of the transition set to fire
16:   s ← fire(s, ET)                                 ▷ compute new state s by firing all t ∈ ET
17:   write(count, s)                                 ▷ add s to trace
18: end while
```

The properties of all bounded nets can be decided with the help of the reachability graph, and the properties of some unbounded nets with the help of static analysis techniques. Give Charlie a try; see [1] for details. However, the required knowledge has not yet been introduced. It might make sense to come back to this exercise at the end of Section 7.3.2 ($\mathcal{PN}$ analysis).

Hint. The general behavioral properties are provably decidable for any $\mathcal{PN}$. However, there is no efficient algorithm known to do it for unbounded nets in the general case (if none of the static analyses help). It is even not known if such an algorithm exists that can do it in exponential time.

(a) *No constraints*: see Figure S7.1; blue nets allowed.
(b) *Ordinary*: see Figure S7.1; black nets.
(c) *Ordinary and pure*: see Figure S7.2; more solutions can be found in [2].
(d) *Ordinary, pure, and strongly connected*: see Figure S7.3. □

Exercise 7.7 (Incidence Matrix)**.** The computation of P/T-invariants builds on a matrix, which is called incidence matrix in Petri net theory, and stoichiometric matrix in chemical network theory. The incidence matrix encodes the structure of a Petri net by having as many rows as there are places and as many columns as there are transitions. The integer matrix entry (p, t) gives the token change on place p by the firing of transition t.

(a) Show that a Petri net with read arcs cannot be uniquely described by the incidence matrix. Which other situations are not fully reflected by the incidence matrix? Give examples.
(b) A well-defined matrix operation is the transposition, which exchanges rows and columns. If we apply the matrix transposition to the incidence matrix of a Petri net, What happens with the Petri net and its invariants?

Solution. We assume familiarity with the notion *incidence matrix* (*stoichiometric matrix*); otherwise consult, for example, [3, 4].

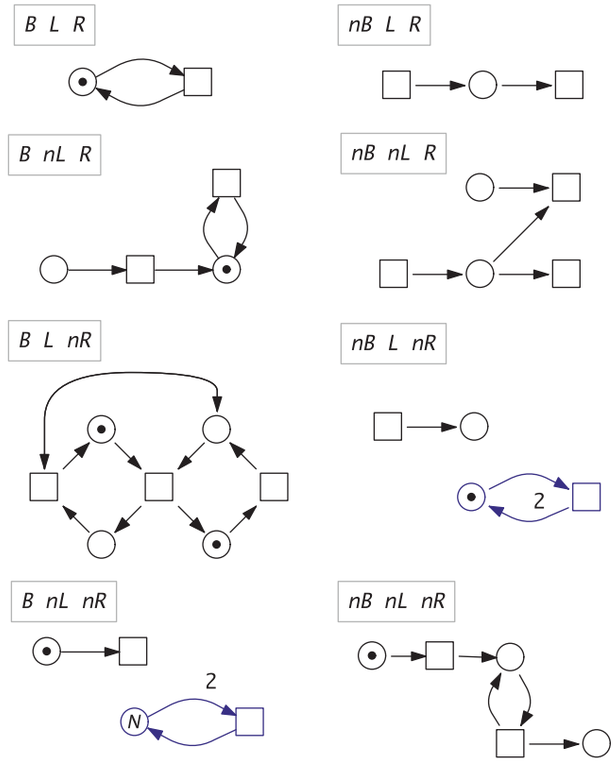


FIGURE S7.1 Eight Petri nets (no constraints—blue, ordinary—black) proving the orthogonality of the three behavioral properties *boundedness-liveness-reversibility*.

(a) *Read arcs*: or two opposite arcs with equal arc weight yield zero entries in the incidence matrix; see the example shown in Figure S7.4. Thus, drawing a $\mathcal{PN}$ given by its incidence matrix does not reveal the dependencies on any side conditions.

Likewise, if a place p and a transition t are connected in both directions by arcs with different arc weights, then the matrix entry just gives the difference of these weights. Let's assume the two weighted arcs $p \xrightarrow{m} t$ and $t \xrightarrow{n} p$, then the corresponding entry in the incidence matrix is $c(p,t) = n - m$.

(b) *Matrix transposition*: This exercise is best discussed with the help of a small example. We take the one used in [3] to introduce invariants; see Figure S7.4, and Figure S7.5 for the subnets induced by the invariants.

Transposing the incidence matrix and going back to the corresponding net representation yields the $\mathcal{PN}$ shown in Figure S7.6, with its invariants (subnets induced by them) shown in Figure S7.7. Obviously, the matrix transposition exchanges places and transitions, and reverts all arcs. The P-invariants (T-invariants) of the original net become the T-invariants (P-invariants) of the transposed net.

Lesson learned: The correspondence of the invariants means that only one algorithm needs to be implemented: either the computation of the P-invariants or the computation of the T-invariants, complemented by the simple procedure of matrix transposition. See [4] for pseudocode to compute P-invariants. □

Exercise 7.8 (Visualizing Invariants)**.** Invariants are computed with Charlie, but can be visualized with Snoopy using the following steps.

1. Open the net to which the invariants belong.
2. Go to *Extras* → *Load Node Set file*, and select the text file written by Charlie (its default extension is *inv*).

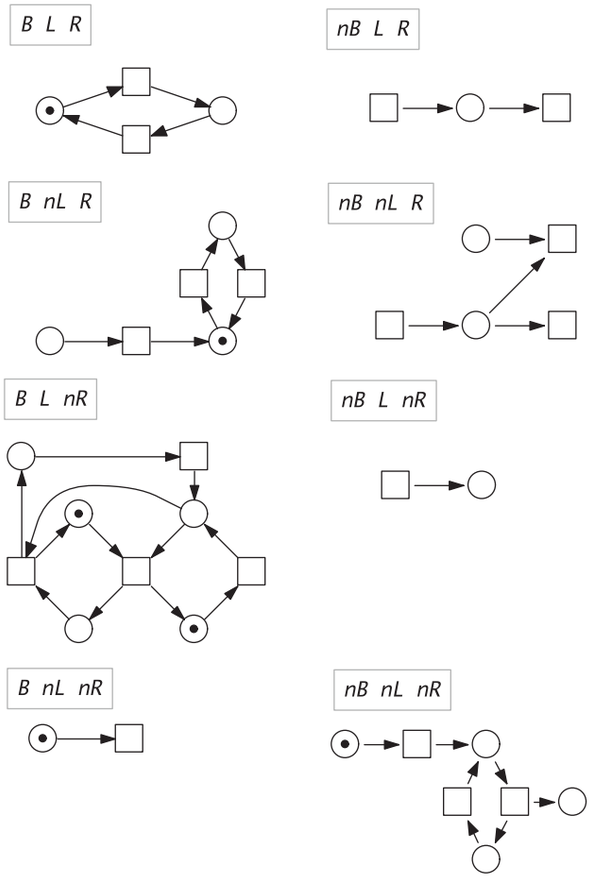


FIGURE S7.2 Eight ordinary and pure Petri nets proving the orthogonality of the three behavioral properties *boundedness-liveness-reversibility*.

3. A new window pops up, which allows you to run through all invariants and highlight the induced subnets in your favorite color. Single invariants or sets of invariants can be chosen. Invariants generally overlap; thus, when selecting a set of invariants, one can choose between *intersection* and *union*.

4. If you want to save the highlighting, choose *keep coloring* before closing the window.

To familiarize yourself with these features, generate for the running example a $\mathcal{PN}$, showing the subnets induced by the two P-invariants in color.

Solution. The $\mathcal{PN}$ as shown in Figure S7.8 is obtained by following the given procedure twice, changing the color in between.

Coloring the T-invariants for the running example requires a bit more effort, and the T-invariants also overlap. So one would need two copies of the $\mathcal{PN}$ to visualize all T-invariants. □

Exercise 7.9 (CTL Model Checking)**.** While computational tree logic (CTL) model checking fits particularly for the decision of special properties, it can also be used to decide general properties. Express for the bounded version of the running example the following properties in CTL and check them with Marcie.

(a) Check the liveness of reactions $r1$ and $r8$. What had to be done to decide the liveness of the net?

(b) Check the reversibility of the net.

(c) Check the token conservation of the two P-invariants.

(d) Checking the upper bounds for A, R, and A_R by the queries

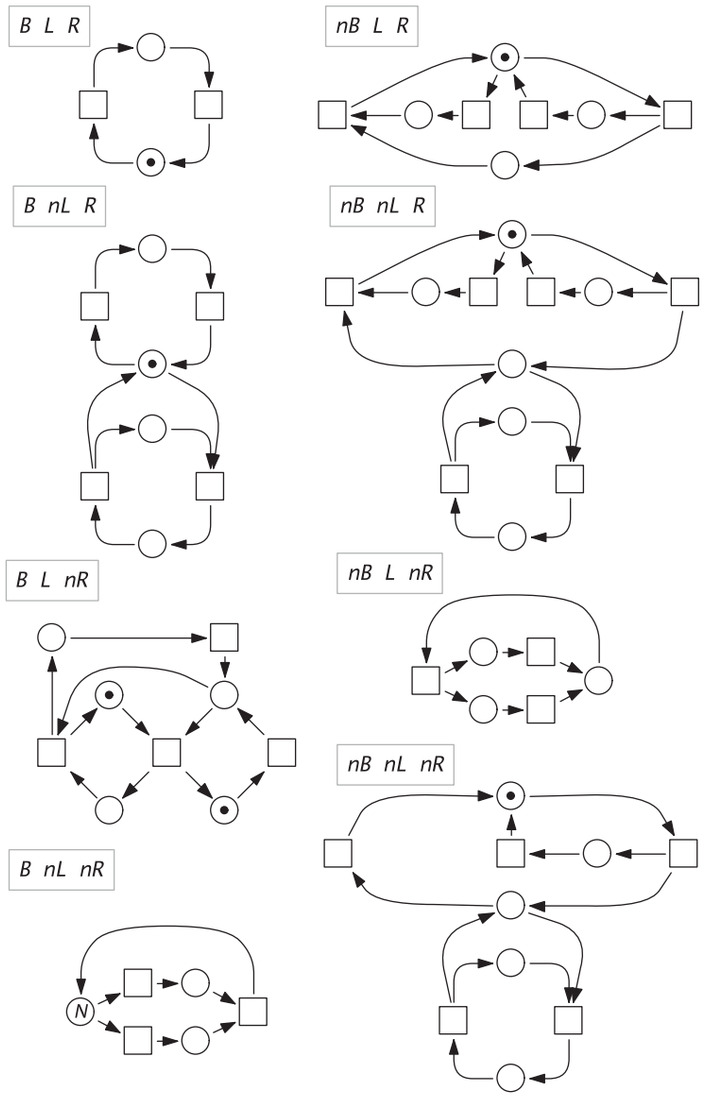


FIGURE S7.3 Eight ordinary, pure, and strongly connected Petri nets proving the orthogonality of the three behavioral properties *boundedness-liveness-reversibility*.

EF[$A > C2$], **EF**[$R > C2$], **EF**[$A\_R > C2$]

yields in the last case FALSE (as expected), but for the other two cases TRUE. Why? What are the correct upper bounds?

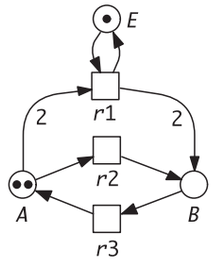
Solution. We assume basic understanding of the CTL operators; otherwise consult, for example, [3]. We also assume that Marcie has been installed so it can be called from everywhere on your computer; see Marcie's manual [5] for how to do it.

Hint. All CTL operators are reserved keywords for CTL formulae; among them are *A*, and *R*; compare also Marcie's FAQ page.

1. We first create an $(\mathcal{X})\mathcal{PN}$ file, called *vilar-bnd-ctl*, where we replace *A* by *_A*, and *R* by *_R*. Exporting this $(\mathcal{X})\mathcal{PN}$ to the Abstract Net Description Language (ANDL) generates the file *vilar-bnd-ctl.andl*.
2. Next, we define the file *vilar-bnd.ctl*:

$$\begin{array}{rrcl} \text{r1:} & 2\ \text{A} & \xrightarrow{E} & 2\ \text{B} \\ \text{r2/3:} & \text{A} & \rightleftharpoons & \text{B} \end{array}$$

	r1	r2	r3
A	−2	−1	1
B	2	1	−1
E	0	0	0



$$pi_1 = (1,1,0) = (\text{A, B}),$$
$$pi_2 = (0,0,1) = (\text{E})$$

$$ti_1 = (1,0,2) = (\text{r1, } 2 \cdot \text{r3}),$$
$$ti_2 = (0,1,1) = (\text{r2, r3})$$

$$ti_3 = (1,1,3) = ti_1 + ti_2$$

FIGURE S7.4 Two reaction equations with the corresponding $\mathcal{PN}$; its incidence matrix; the minimal P-invariants pi_1, pi_2; the minimal T-invariants ti_1, ti_2; and a nonminimal T-invariant ti_3. The invariants are given in the standard vector notation as well as in a shorthand notation, listing the nonzero entries only. The net is not pure; the incidence matrix does not reflect the dependency of $r1$ on E.

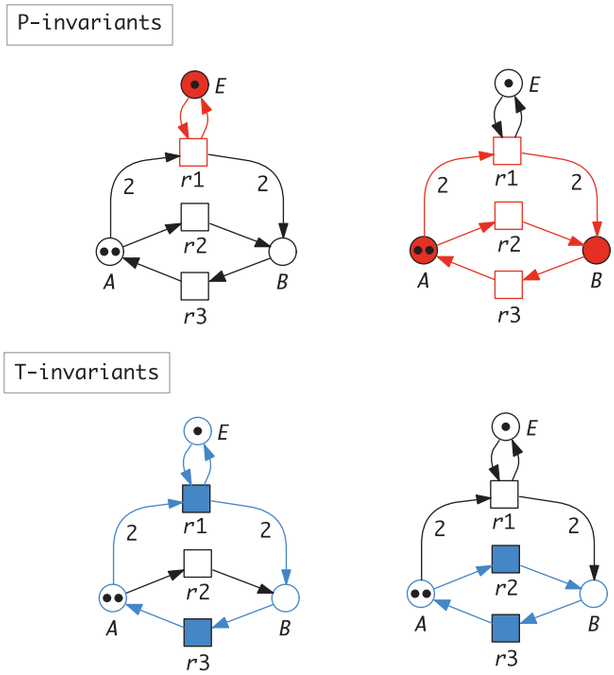


FIGURE S7.5 Subnets induced by the P/T-invariants for the $\mathcal{PN}$ given in Figure S7.4.

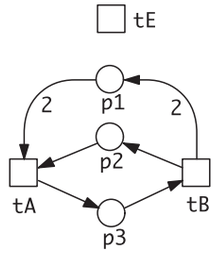
```
// Exercise 7.9 (a)
// liveness of r1;
AG [ EF [ geneA=1 & _A>=1 ] ];

// liveness of r8;
AG [ EF [ _A>=1 & _R>=1 ] ];

// Exercise 7.9 (b) - reversibility
AG [ EF [ geneA=1 & geneR=1 ] ];

// Exercise 7.9 (c) - token conservation of the two P-invariants
AG [ [geneA=1 & geneA_A=0] | [geneA=0 & geneA_A=1] ];
AG [ [geneR=1 & geneR_A=0] | [geneR=0 & geneR_A=1] ];
```

	tA	tB	tE
p1	−2	2	0
p2	−1	1	0
p3	1	−1	0



$ti_1 = (1, 1, 0) = (\text{tA}, \text{tB}),$
$ti_2 = (0, 0, 1) = (\text{tE})$

$pi_1 = (1, 0, 2) = (\text{p1}, 2 \cdot \text{p3}),$
$pi_2 = (0, 1, 1) = (\text{p2}, \text{p3})$

$pi_3 = (1, 1, 3) = pi_1 + pi_2$

FIGURE S7.6 The transposed incidence matrix; its $\mathcal{PN}$; the minimal T-invariants ti_1, ti_2; the minimal P-invariants pi_1, pi_2; and a nonminimal P-invariant pi_3. The invariants are given in the standard vector notation as well as in a shorthand notation, listing the nonzero entries only. The net is not connected.

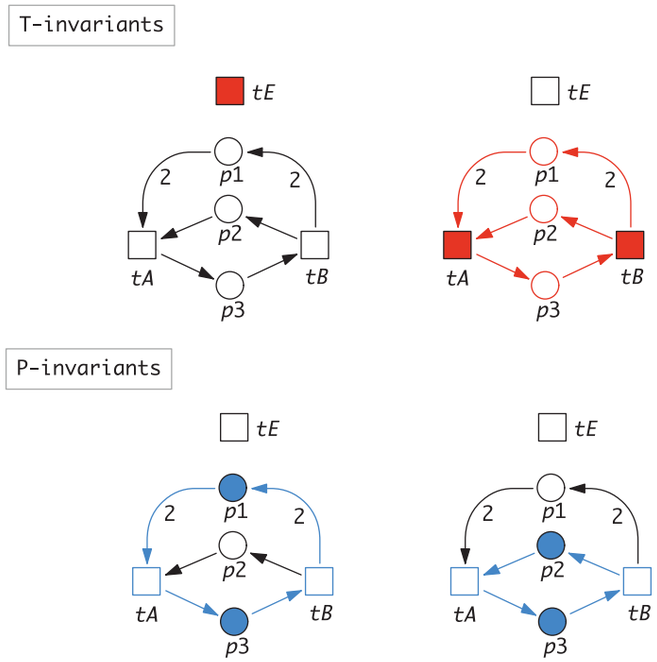


FIGURE S7.7 Subnets induced by the P/T-invariants for the $\mathcal{PN}$ given in Figure S7.6.

```
// Exercise 7.9 (d) - upper bounds, first shot
EF [_A > C2];
EF [_R > C2];
EF [A_R > C2];

// Exercise 7.9 (d) - correct upper bounds
EF [_A = C2+2];
EF [_A > C2+2];
EF [_R = 2*C2];
EF [_R > 2*C2];
```

Hint. To check the liveness of the net, we had to write for each transition t a CTL formula according to the pattern `AG[ EF[ enabled(t) ]]`, with `enabled(t)` specifying the predicate for the enabledness of t.

The number of tokens on A can exceed $C2$ by 2, because two A tokens can be bound to *geneA_A* and *geneR_A*, which may be released by $r2$ and $r10$ after the capacity was exhausted by the firing of $r5$.

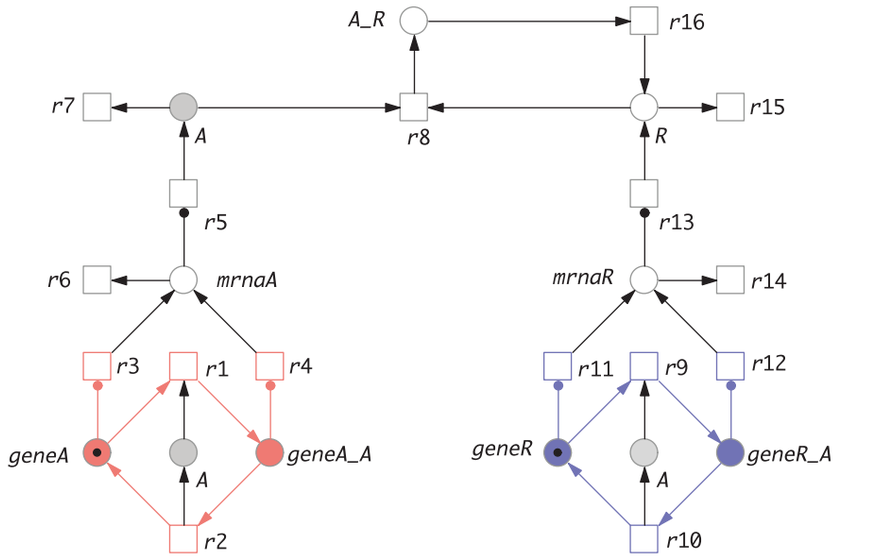


FIGURE S7.8 Running example with highlighted subnets that are induced by the two P-invariants. Charlie's result vector is shown as well. The undecided properties (-) cannot be determined for this $\mathcal{PN}$ with the help of static analysis techniques.

Likewise, only the firing of $r13$ is limited by the upper bound $C2$, but also $r16$ puts tokens on R. Thus, while $C2$ tokens are waiting on A_R (more is impossible), R could be filled again to the limit, and afterward $r16$ fires $C2$ times to increase the token number to $2 \cdot C2$.

3. Finally, we assume specific values for all constants, for example, 10, and call Marcie (from the folder containing the files *.andl and *.ctl) with the command

```
marcie \
-net-file=vilar-bnd-ctl.andl \
-ctl-file=vilar-bnd.ctl \
-const C1=10,C2=10
```

Hint. Just copy/paste the command as given into a terminal.

We obtain, as expected, TRUE for all queries, except for `EF[_A>C2+2]` and `EF[_R>2*C2]`, which have been added as a maximality test, likewise for all reasonable values for the constants $C1$ and $C2$.

4. *Determining upper bounds*: Using CTL, we need to make a guess, and then Marcie tells us if we got it right or wrong. Alternatively, we could ask Marcie straightaway to tell us the upper bounds with the command

```
marcie -net-file=vilar_bnd_ctl.andl \
-Max-vals=vilar-bounds.txt \
-const C1=10,C2=10
```

which writes a list of the upper bounds for all places into the file *vilar-bounds.txt*. □

Exercise 7.10 (Behavior of Open vs. Closed Systems)**.** Consider the following toy example for a system of chemical equations.

$$\begin{aligned} 2C + O_2 &\to 2CO \\ C + O_2 &\to CO_2 \\ C + CO_2 &\rightleftarrows 2CO \end{aligned}$$

Derive a corresponding Petri net model and explore its behavior by qualitative analysis techniques.

(a) Which structural properties hold?

(b) Determine and interpret the P- and T-invariants.

(c) The $\mathcal{PN}$ is bounded, but neither live nor reversible (for any initial state). Construct a live and reversible $\mathcal{PN}$ by assuming appropriate interconnections with an environment. This could be done as

(c.1) Open system—using input/output transitions, and

(c.2) Closed system—avoiding input/output transitions.

Do not forget to define appropriate but minimal initial states.

(d) Determine and interpret the P- and T-invariants for the open and closed system. Which structural properties are different?

(e) Determine the behavioral properties for the open and closed system.

(f) Try to construct the reachability graph for the open system. What is the problem? Construct the reachability graph for the closed system. Decide liveness, reversibility, and existence of dynamic conflicts in terms of the reachability graph. Are there concurrent reactions? Check the feasibility of the minimal T-invariants.

(g) The infinitely repeated occurrence of the first chemical reaction $2C + O_2 \rightarrow 2CO$ could be translated into

(g.1) $\mathbf{AG}[(C \geq 2 \;\&\; O_2 \geq 1) \rightarrow \mathbf{AX}(CO \geq 2)]$

(g.2) $\mathbf{AG}[(C \geq 2 \;\&\; O_2 \geq 1) \rightarrow \mathbf{EX}(CO \geq 2)]$

(g.3) $\mathbf{AG}[(C \geq 2 \;\&\; O_2 \geq 1) \rightarrow \mathbf{AF}(CO \geq 2)]$

(g.4) $\mathbf{AG}[(C \geq 2 \;\&\; O_2 \geq 1) \rightarrow \mathbf{EF}(CO \geq 2)]$

What is the difference? Which of them hold?

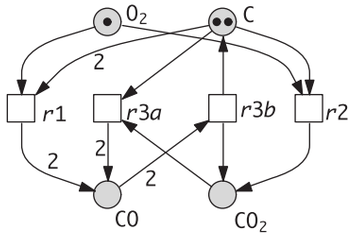
You should be able to solve all subtasks of this exercise without tool support, but you can check most of your answers with the help of Snoopy, Charlie, and Marcie.

Solution. This toy example allows you to practice most of the Petri net analysis techniques. We summarize here the core of the expected answers; for more details, see the Supplementary Material *exercise_7_10_appendix.sld2.pdf* (*slides*).

Transforming the given system of chemical equations yields the $\mathcal{PN}$ shown in Figure S7.9. It could also be defined in a modular way with the use of logical places; see the slides in the Supplementary Material.

(a) *Structural properties*: See the first line in Charlie's result vector, given in Figure S7.9.

(b) *P/T-invariants*: There are two minimal P-invariants (in Charlie's notation as read by Snoopy):




```
Carbon combustion, open system

(1)  2 C + O2      -->  2 CO
(2)    C + O2      -->         CO2
(3)    C +    CO2 <-->  2 CO
```

PUR	ORD	HOM	NBM	CSV	SCF	FT0	TF0	FP0	PF0	CON	SC	NC
Y	N	N	N	N	N	Y	Y	N	Y	Y	N	ES
RKTH	STP	CPI	CTI	SCTI	SB	k-B	1-B	DCF	DSt	DTr	LIV	REV
N	N	Y	N	N	Y	2	N	N	0	Y	N	N

FIGURE S7.9 The $\mathcal{PN}$ *carbon1* for the given chemical equations and its Charlie result vector; see the Supplementary Material for Charlie's complete analysis protocol, and Figure S7.10 for a related screenshot. The initial state has been chosen to avoid dead states.

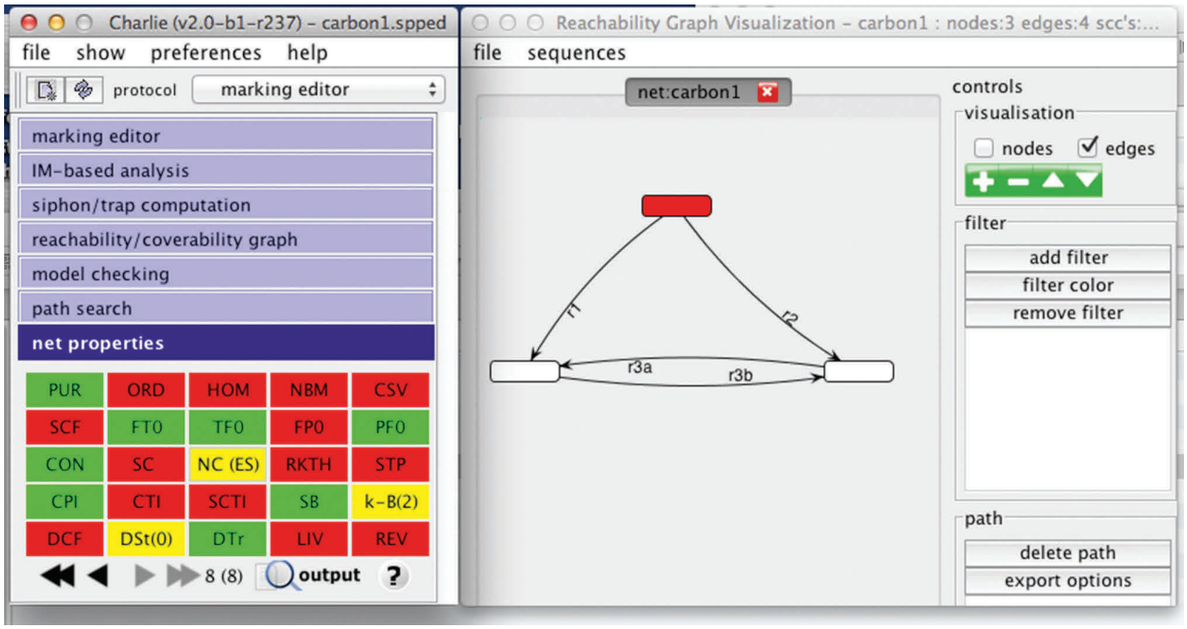


FIGURE S7.10 Screenshot for analyzing *carbon1* given in Figure S7.9 with Charlie. On the right-hand side, the reachability graph for the chosen initial state (in red) can be seen.

```
1 | 0.CO2       :2,
  | 1.CO        :1,
  | 2.O2        :2
2 | 0.CO2       :1,
  | 1.CO        :1,
  | 3.C         :1
```

The first P-invariant reflects the preservation of oxygen (O) molecules, the second one the preservation of carbon (C) molecules. There is one minimal T-invariant, which is a trivial T-invariant for the two directions of the reversible reaction.

```
1 | 0.r3b       :1,
  | 2.r3a       :1
```

Hint. In this notation, each entry "$|x.y : z$" specifies one node involved, with x—automatically generated node identifier (just ignore it), y—user-defined node name, z—factor (corresponding entry in the vector notation). For example, using the node identifiers to order the entries, the first P-invariant could also be written in vector notation as $(2, 1, 2, 0)$, and the T-invariant as $(1, 0, 1, 0)$.

(c) *Adding environment*: For more details, see the slides in the Supplementary Material.

(c.1) Open system—using input/output transitions; see Figure S7.11.

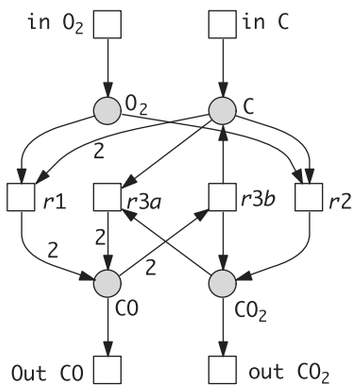
(c.2) Closed system—avoiding input/output transitions; see Figures S7.12 and S7.13.

(d) *P/T-invariants*:

(d.1) Open system—there are no P-invariants, but five T-invariants covering the $\mathcal{PN}$; see Supplementary Material.

(d.2) Closed system—there are two P-invariants and four T-invariants, each covering the $\mathcal{PN}$; see Supplementary Material.

The two $\mathcal{PN}$ differ in the following structural properties. Compare the Charlie result vectors in Figures S7.11 and S7.12: *carbon3* is strongly connected, thus without boundary nodes, while *carbon2* has input and output transitions, thus it is not strongly connected. All other structural properties are identical.

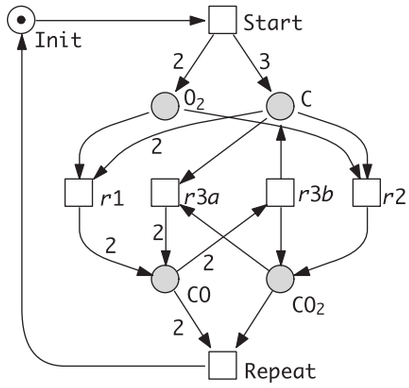


Carbon combustion, open system

(1) 2 C + O_2 --> 2 CO
(2) C + O_2 --> CO_2
(3) C + CO_2 <--> 2 CO

PUR	ORD	HOM	NBM	CSV	SCF	FT0	TF0	FP0	PF0	CON	SC	NC
Y	N	N	N	N	N	N	N	Y	Y	Y	N	nES
RKTH	STP	CPI	CTI	SCTI	SB	k-B	1-B	DCF	DSt	DTr	LIV	REV
N	Y	N	Y	-	N	N	N	-	0	Y	Y	-

FIGURE S7.11 The $\mathcal{PN}$ *carbon2* for the given chemical equations and its Charlie result vector; see the Supplementary Material for Charlie's complete analysis protocol. Here, an environment is assumed that provides a permanent in/out flow (modeled by input/output transitions); we obtain an open system.



Carbon combustion, closed system

(1) 2 C + O_2 --> 2 CO
(2) C + O_2 --> CO_2
(3) C + CO_2 <--> 2 CO

PUR	ORD	HOM	NBM	CSV	SCF	FT0	TF0	FP0	PF0	CON	SC	NC
Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	nES
RKTH	STP	CPI	CTI	SCTI	SB	k-B	1-B	DCF	DSt	DTr	LIV	REV
N	N	Y	Y	Y	Y	3	N	N	0	Y	Y	Y

FIGURE S7.12 The $\mathcal{PN}$ *carbon3* for the given chemical equations and its Charlie result vector; see the Supplementary Material for Charlie's complete analysis protocol. Here, the environment is assumed to recycle the outcome (modeled by the transitions *start* and *repeat*); we obtain a closed system. See the slides in the Supplementary Material on how to determine the environment subnet.

(e) *Behavioral properties*: Both model versions are (apparently) live and reversible, but the open system is unbounded, while the closed system is bounded.

The open system does not have a siphon, thus we can structurally deduce that it is live.

These behavioral properties can be easily checked for the closed system with the help of the reachability graph; see next subtask.

(f) *Reachability graph*:

(f.1) Open system—The state space is infinite and so is the reachability graph, which precludes its construction.

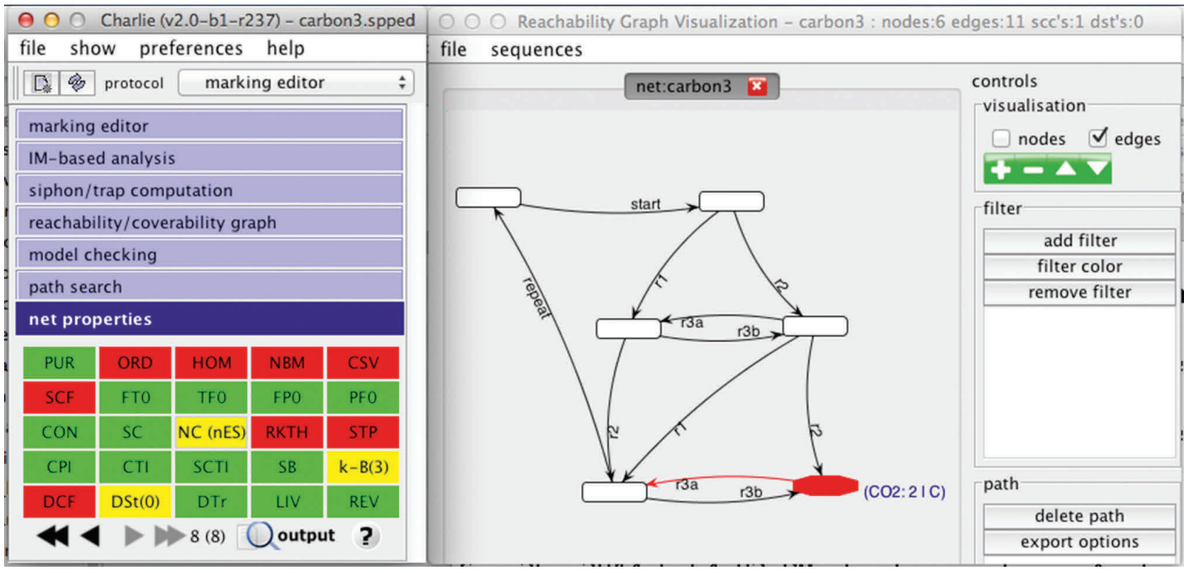


FIGURE S7.13 Screenshot for analyzing *carbon3* given in Figure S7.12 with Charlie. On the right-hand side, the reachability graph for the chosen initial state (in red) can be seen.

(f.2) Closed system—The reachability graph is given in Figure S7.13, right-hand side.

- The reachability graph is strongly connected, thus *carbon3* is reversible.
- All transitions serve as a label for at least one arc in the strongly connected reachability graph; thus *carbon3* is live.
- There is a dynamic conflict in the state $(O_2, 2C, CO_2)$; $r1$ and $r2$ compete for the tokens on C and O_2, which are consumed either by $r1$ or by $r2$, disabling the other reaction.
- A T-invariant is feasible for a given system ($\mathcal{PN}$ with initial state) if there is a corresponding cycle in the reachability graph. This holds for all minimal T-invariants; compare the slides in the Supplementary Material.

(g) *CTL properties*:

(g.1) $\mathbf{AG}[(C \geq 2 \,\&\, O_2 \geq 1) \rightarrow \mathbf{AX}(CO \geq 2)]$

In words: Forever it holds, if $r1$ is enabled, then there are at least two tokens on CO in all states reachable by one step.

(g.2) $\mathbf{AG}[(C \geq 2 \,\&\, O_2 \geq 1) \rightarrow \mathbf{EX}(CO \geq 2)]$

In words: Forever it holds, if $r1$ is enabled, then there is a state with at least two tokens on CO reachable by one step.

(g.3) $\mathbf{AG}[(C \geq 2 \,\&\, O_2 \geq 1) \rightarrow \mathbf{AF}(CO \geq 2)]$

In words: Forever it holds, if $r1$ is enabled, then all paths lead (sooner or later) to a state with at least two tokens on CO.

(g.4) $\mathbf{AG}[(C \geq 2 \,\&\, O_2 \geq 1) \rightarrow \mathbf{EF}(CO \geq 2)]$

In words: Forever it holds, if $r1$ is enabled, then there is a path leading (sooner or later) to a state with at least two tokens on CO.

The first property does not hold, due to the concurrent enabledness of $r1$ and $r2$ in the state $(2O_2, 3C)$, which is replaced by the two interleaving sequences $r1, r2$ and $r2, r1$; compare slides in the Supplementary Material. Thus, if $r2$ is considered first, then there are no two tokens on CO in the next state. The other three properties hold.

We provide the CTL file *carbon3.ctl* to check these properties for *carbon3* with Marcie:

```
marcie \
-net-file=carbon3.andl \
-ctl-file=carbon3.ctl
```

As the state space is tiny, it could also be done with Charlie, or just manually. □

7.2 STOCHASTIC PETRI NETS ($\mathcal{SPN}$)

Exercise 7.11 (Changing the Net Class)**.** Snoopy supports the conversion of a (qualitative) Petri net into a quantitative Petri net (or vice versa), which allows us to reuse the structure. Only the additional attributes have to be set.

1. Open the $\mathcal{PN}$; go to the export Window (*File* → *Export*); choose the appropriate target(s), among them $\mathcal{SPN}$; and hit the OK button.
2. Open the $\mathcal{SPN}$ just generated. It looks exactly the same as the $\mathcal{PN}$ (when *show rate functions* is switched off). It also has the same constants and functions; see *Declarations* panel.
3. The default rate function automatically set for stochastic transitions is *MassAction*(1).
4. Now we are ready to specify $\mathcal{SPN}$-specific attributes.
   - Add the kinetic parameters as constants in the *Declarations* panel.
   - Double-clicking on a transition opens the *Edit Properties* window. Go to *Function* in order to change the rate functions.

   Generate for the running example the $\mathcal{SPN}$ out of its $\mathcal{PN}$.

Solution. Self-explanatory; the solution is part of the Supplementary Material of the book chapter. □

Exercise 7.12 (Repeated Stochastic Simulation)**.** Take the $\mathcal{SPN}$ of the running example and familiarize yourself with Snoopy's simulator. The simulation can be parameterized in four ways.

- *Model configuration*: There are different sets of initial states (markings), rate functions and parameters can be defined, and one of each has to be chosen for a given experiment.
- *Simulation time*: The simulation starts always at time 0 and goes until the specified *Interval end* if no dead state is encountered. But, the simulation data are only recorded for the simulation interval given by *interval start* and *interval end*.
- *Output step count*: The number of output steps in the specified simulation interval defines the output grid of the recorded simulation data.
- *Number of runs and threads*: The number of simulation runs to be averaged and the number of parallel threads to be deployed have to be specified.

Having started the simulation, the progress bar indicates how far the simulation has been going until now, and the time consumed by the simulation is displayed too.

There are three export options:

- *Direct export*: The averaged result of the selected places or transitions is exported.
- *Single trace export*: The result of every simulation run is exported separately.
- *Exact trace export*: Every change of the marking of the selected places or the rates of the selected transitions at any time point is exported.

The simulation results can be shown as

- *Table*: Each column represents a selected place (transition) and each row shows the averaged marking (firing times) at one output step point.
- *Plot*: Each curve stands for one selected place (transition); the x-axis holds the time and the y-axis holds the number of tokens (firing times).

Different tables and plots can be created to switch conveniently between different views of the simulation results. Each view is characterized by a set of selected places (transitions). Simulation plots can also be saved. For more details, consult Snoopy's FAQ.

Solution. Self-explanatory; this exercise should permit you to reproduce Figure 7.16. □

Exercise 7.13 (Exploring Stochasticity)**.** Take the $\mathcal{SPN}_1$ in Figure 7.10 and the $\mathcal{SPN}_2$ in Figure 7.11 to explore the effect of stochastic simulations by varying the initial token number N on A, the kinetic

parameters k (k_1, k_2), and the number of runs. Find suitable simulation options for *interval end* and *output step count*.

Solution. We suggest trying the following options for both models.

- *initial token number N*: 1, 10, 100, 1000, 10 000, and 100 000.
- *kinetic parameters* k, k_1, k_2: 0.01, 0.1, 1, and 10; also try different values for k_1, k_2.
- *number of runs*: 1, 10, 100, 1000, and 10 000.
- *interval end*: 10 ($\mathcal{SPN}_1$), 50 ($\mathcal{SPN}_2$).
- *output step count*: 1, 10, 100, and 1000. *Hint*: Use the *tabular view* to better observe the difference. □

Exercise 7.14 (Estimating Memory Consumption)**.** Stochastic analyses are often quite memory-expensive. To get the idea, calculate the following estimates. We assume that the memory consumption for the Petri net itself and the algorithms can be neglected. All considered data structures store double values (8 byte).

(a) *Stochastic simulation*: Two types of data structures are required.

- One matrix serves as the result table and stores a value for each place and each time point in the output grid.
- Two vectors in the size of the number of places keep the initial and current state.

Estimate the required memory for Petri nets with 10, 100, 1000, 10,000, and 100,000 places, and for output grids of 10, 100, 1000, 10,000, and 100,000 time points.

Assuming you have a computer with 1/4/8/16/32 GB (free) memory, Which kind of simulation experiments can you handle?

(b) *Stochastic numerical methods*: Two types of data structures are required.

- One square matrix in the size of the number of states encodes the Continuous-time Markov chain (CTMC).

  Marcie does not explicitly hold the CTMC; instead it recomputes, when required, the matrix on the fly.
- Two to four vectors in the size of the state space keep the results and auxiliary data, depending on the engine and the used algorithms.

Let's consider the algorithms requiring two vectors. Estimate the required memory for the state spaces given in Table 7.7 for (b1) Marcie (CTMC memory consumption can be neglected), and (b2) a hypothetical tool explicitly encoding the CTMC matrix.

Assuming you have a computer with 1/4/8/16/32 GB (free) memory, Which problem size can you handle?

Solution. *Hint.* The prefix *giga* means 10^9 in the International System of Units (SI), therefore one gigabyte (1 GB) is 1,000,000,000 bytes.

(a) *Stochastic simulation*: total memory consumption = (number of places × grid size + 2 × number of places) × 8 byte.

To prepare our answer, we calculate the spreadsheet given in Table S7.1.

- 1 GB = 10^9 bytes allow us to record simulation results on 10^4 grid points for models with 10^4 places, or simulation results on 10^3 grid points for models with 10^5 places.
- 4 GB = 4×10^9 bytes allow us to record simulation results on 10^5 grid points for models with 10^4 places, or simulation results on 10^3 grid points for models with 10^5 places.
- 8 GB = 8×10^9 bytes allow the same size simulation experiments as 4 GB.
- 16 GB = 1.6×10^{10} bytes permit the largest simulation experiments considered in Table S7.1, that is, recording of 10^5 grid points for models with 10^5 places.
- 32 GB = 3.2×10^{10} bytes—see 16 GB.

(b) *Stochastic numerical methods*: To prepare our answer, we calculate the spreadsheet given in Table S7.2 and adopt the notation ($C1, C2$) for the two parameters determining the size of the state space.

- *Marcie*: 1 GB = 10^9 bytes are sufficient to deal with up to (1000, 1), (10, 10), or (1, 100). 4 GB extend it to (100, 10), and 16 GB to (10, 100).
- *Tool X*: There is not much difference between 1 to 32 GB memory; *X* is able to deal with (1, 1), (10, 1), and (1, 10).

TABLE S7.1 Memory Consumption for Increasing the Number of Places ($|P|$) and the Output Grid Size ($|G|$)

$\|P\|$	$\|G\| = 10$	$\|G\| = 10^2$	$\|G\| = 10^3$	$\|G\| = 10^4$	$\|G\| = 10^5$
Number of double values to be stored					
10	1.2×10^2	1.02×10^3	1.002×10^4	1.0002×10^5	1.00002×10^6
10^2	1.2×10^3	1.02×10^4	1.002×10^5	1.0002×10^6	1.00002×10^7
10^3	1.2×10^4	1.02×10^5	1.002×10^6	1.0002×10^7	1.00002×10^8
10^4	1.2×10^5	1.02×10^6	1.002×10^7	1.0002×10^8	1.00002×10^9
10^5	1.2×10^6	1.02×10^7	1.002×10^8	1.0002×10^9	1.00002×10^{10}
Times 8 byte makes					
10	9.6×10^2	8.16×10^3	8.016×10^4	8.0016×10^5	1.00002×10^6
10^2	9.6×10^3	8.16×10^4	8.016×10^5	8.0016×10^6	1.00002×10^7
10^3	9.6×10^4	8.16×10^5	8.016×10^6	8.0016×10^7	1.00002×10^8
10^4	9.6×10^5	8.16×10^6	8.016×10^7	8.0016×10^8	1.00002×10^9
10^5	9.6×10^6	8.16×10^7	8.016×10^8	8.0016×10^9	1.00002×10^{10}
Bytes of free memory are required.					

TABLE S7.2 Memory Consumption for Increasing the Constants $C1$, $C2$; Compare Figure 7.9

$C1$	$C2 = 1$	$C2 = 10$	$C2 = 100$	$C2 = 1000$
(b0) CTMC memory consumption $= 8 \cdot \|S\| \cdot \|S\|$, *with S—set of states*				
1	4.6×10^5	9.1×10^9	≥32 GB	≥32 GB
10	4.2×10^8	≥32 GB	≥32 GB	≥32 GB
100	≥32 GB	≥32 GB	≥32 GB	≥32 GB
1000	≥32 GB	≥32 GB	≥32 GB	≥32 GB
(b1) Marcie's memory requirement = Memory consumption for 2 vectors $= 16 \cdot \|S\|$				
1	3.8×10^3	5.4×10^5	3.9×10^8	≥ 32 GB
10	1.1×10^5	1.6×10^7	1.2×10^{10}	≥32 GB
100	9.7×10^6	1.3×10^9	≥32 GB	≥32 GB
1000	9.6×10^8	≥32 GB	≥32 GB	≥32 GB
(b2) Total memory requirement for a hypothetical tool X = (b0) + (b1)				
1	$4.6 \times 10^5 + 3.8 \times 10^3$	$9.1 \times 10^9 + 5.4 \times 10^5$	≥32 GB	≥32 GB
10	$4.2 \times 10^8 + 1.1 \times 10^5$	≥32 GB	≥32 GB	≥32 GB
100	≥32 GB	≥32 GB	≥32 GB	≥32 GB
1000	≥32 GB	≥32 GB	≥32 GB	≥32 GB

Note: This calculation builds on Table 7.7.

Lesson learned: Simulative and numerical methods differ dramatically in their memory requirements. Numerical methods with explicit CTMC representation can only deal with very small state spaces. A memory increase from 1 GB to 32 GB (or even higher) is of minor importance for numerical methods, and not important at all for numerical methods with explicit CTMC representation. □

Exercise 7.15 (Exploring the Running Example)**.** Take the running example (unbounded version) and perform the following computational experiments.

(a) Change δ_R to 0.08. How does the dynamic behavior change? Can oscillations still be obtained after this perturbation?

(b) Consider averaged stochastic simulations with increasing number of runs. How does the observable behavior change? Why?

Solution.

(a) *Changing* δ_R to 0.08: Compare Figure S7.14, showing oscillating *mrnaA* and *mrnaR*. The plots reveal changes in the oscillation in terms of amplitudes and frequency triggered by changing the parameter δ_R. The same effect can be observed for *A*, *R*, and *A_R*.

Explanation: A smaller degradation rate of *R* allows the individual repressor proteins to unfold their effect much longer; that is, more activator proteins per repressor are degraded than for a higher degradation rate. This in turn affects the repressor species itself; that is, the less activators are present, the slower the synthesis of repressors (and activators) takes place. Thus, the oscillation frequency decreases as it takes longer in each cycle until the maximum of the corresponding species has been reached.

(b) *Averaged stochastic simulations*: Due to the phase shift in the individual, independent oscillations, averaging over single runs levels out the curves; see Figure S7.15.

Hint. The given plots have been directly produced with Snoopy (instead of exporting the traces as CSV files, and then drawing the plots with Gnuplot, as done for the book chapter itself). In order to save (graphical) simulation plots, go in the simulation window to *options* → *image export*. The color of the curves in the

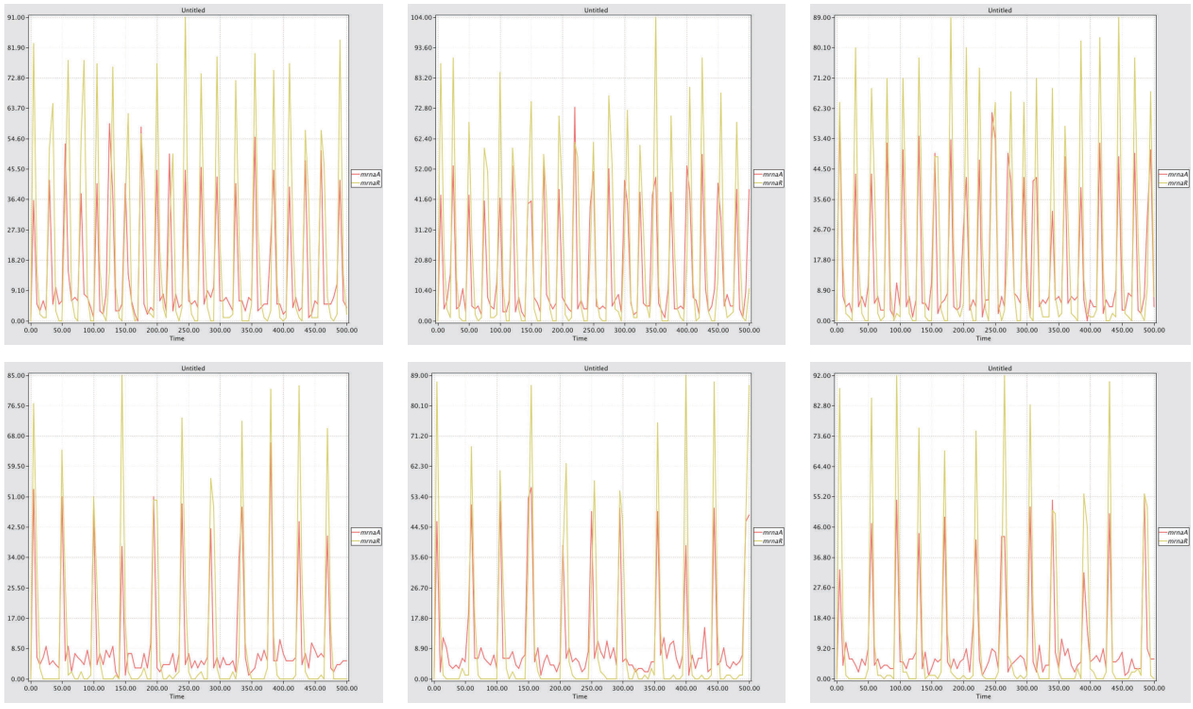


FIGURE S7.14 (First row) Three single runs with $\delta_R = 0.2$ (standard value). (Second row) Three single runs with $\delta_R = 0.08$.

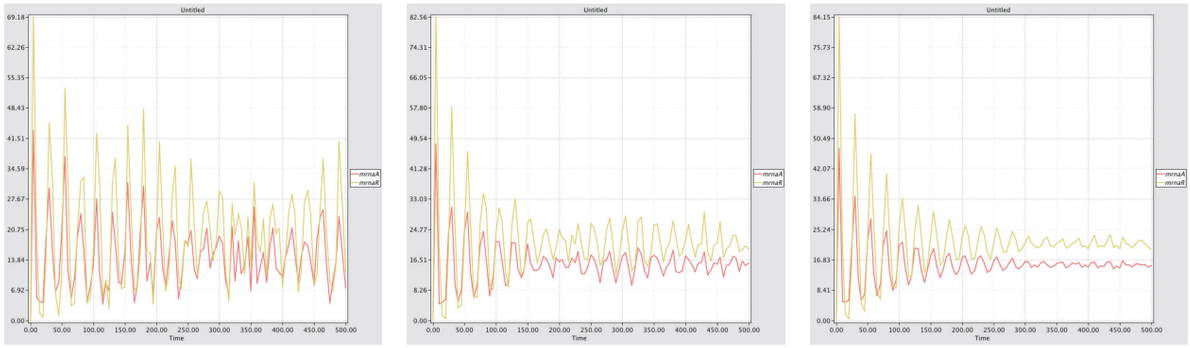


FIGURE S7.15 Averaging over 10, 1000, 10,000 runs, with $\delta_R = 0.2$.

simulation plots can be set by right-clicking on the corresponding species in the panel *select an item to view* on the right-hand side in the simulation window. □

Exercise 7.16 (Exploring the Bounded Version of the Running Example)**.** Take the $\mathcal{XPN}$ in Figure 7.9, which is a bounded version of the running case study; read it as an $\mathcal{SPN}$ with the kinetic parameters of Table 7.4; and explore its behavior by stochastic simulation. You will soon realize that it is behaviorally equivalent to the structurally unbounded version for $C1 \geq 120$, $C2 \geq 2100$.

(a) To simplify life, two constants ($C1, C2$) have been used in this bounded-model version. However, a more detailed analysis reveals more precise different bounds for *mrnaA*, *mrnaR*, *A*, *R*, and *A_R*. Adjust the $\mathcal{SPN}$ appropriately to distinguish between five constants imposing individual upper bounds.

(b) Vary the constants and observe the differences in the results (single runs), specifically for decreasing values. Explain your observations.

Hint. The insights gained in Exercise 7.9d might be worth taking into consideration.

Solution.

(a) *Refined bounded* $\mathcal{SPN}$: See Figure S7.16. Three new constants have been introduced, which are used as arc weights for the three inhibitory arcs limiting the production of *A*, *R*, and *A_R*, while the constant *C*1 controls the production of *mrnaA*, and *C*2 of *mrnaR*.

(b) *Varying constants*: See Figure S7.17 for a screenshot while selecting constants for a specific experiment, and Figure S7.18 for some plots.

Hint. A first rough estimate of appropriate upper bounds can be obtained by a couple of single simulation runs (of the unbounded model). To be more precise, ask Marcie for help.

1. Let's assume our $\mathcal{SPN}$ is called *vilar.spn*. Exporting the $\mathcal{SPN}$ to ANDL generates the file *vilar.andl*.

2. Define a PLTLc file *freevariable-A.ltl* with the corresponding query for, for example, place *A*.

```
const double t1;
const double t2;
P=? [F [t1,t2] A>$v ]
```

3. Call Marcie (from the folder containing the files *.andl and *.ltl) with the command

```
marcie \
-net-file=vilar.andl \
-ltl-file=freevariable-A.ltl \
-const t1=0,t2=100 -simulative \
-threads=4 \
-sim-run=10000
```

This computation may take a while, but will finally yield the probability distribution over the whole value range, which can be visualized with Gnuplot, as shown in Figure 7.17.

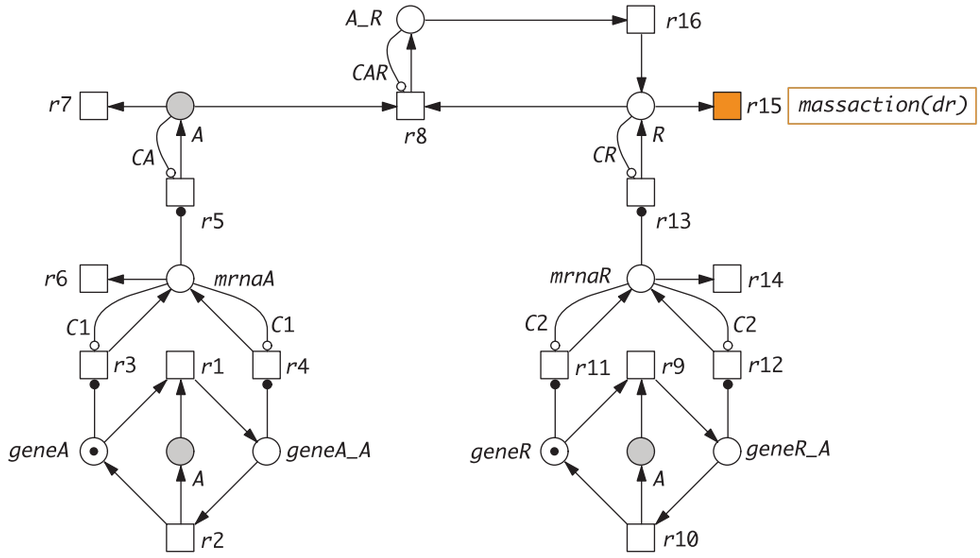


FIGURE S7.16 $\mathcal{SPN}$ of the running example with the individual upper bounds $C1, C2, CA, CR, CAR$. Please note that $C2$ now controls *mrnaR*.

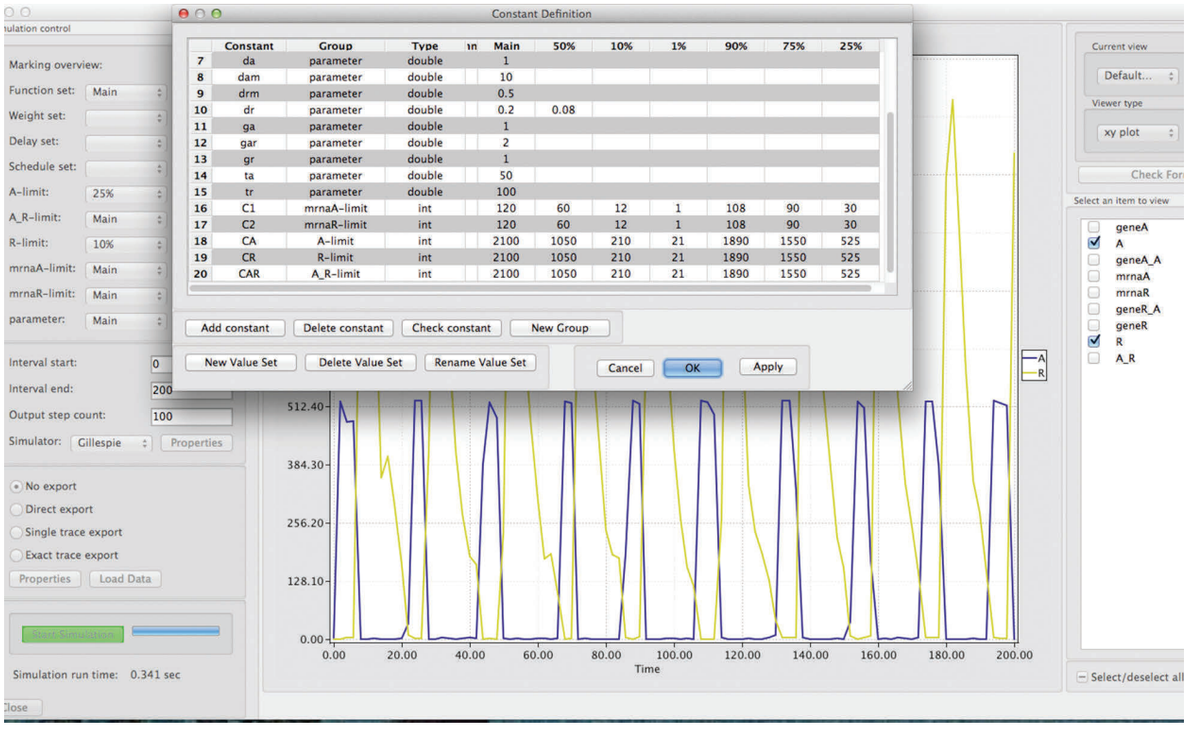


FIGURE S7.17 Screenshot demonstrating some options on how to specify and individually select specific values for every constant in simulation experiments.

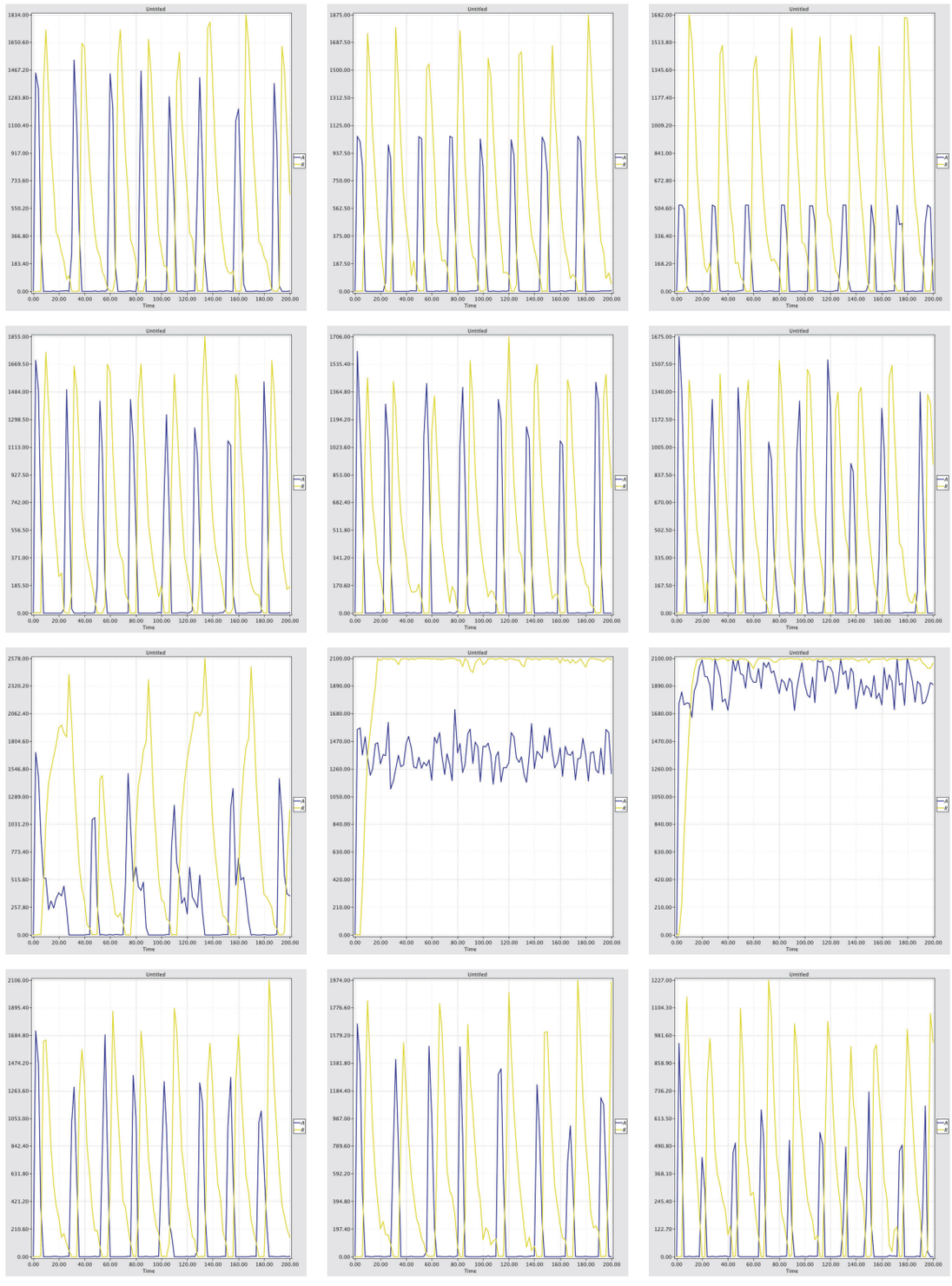


FIGURE S7.18 Decreasing individual constants; left-middle-right column: 90-50-25% of the original value, for (from top to bottom) $A, R, A\_R, mrnaA$.

Hint. We assume that Marcie has been installed so it can be called from everywhere on your computer; see Marcie's manual [5] to learn how to do it. □

7.3 CONTINUOUS PETRI NETS ($\mathcal{CPN}$)

Exercise 7.17 (Changing the Net Class)**.** Snoopy supports the conversion between quantitative Petri nets, which allows us, for example, to read an $\mathcal{SPN}$ as $\mathcal{CPN}$ (or vice versa) without having to do any further changes.

1. Open the $\mathcal{SPN}$; go to the export Window (*File* → *Export*); choose the appropriate target(s), among them $\mathcal{CPN}$; and hit the OK button.
2. Open the $\mathcal{CPN}$ just generated. It looks exactly the same as the $\mathcal{SPN}$. It also has
   - The same constants and functions; see *Declarations* panel.
   - The same rate functions. Double-clicking on a transition opens the *Edit Properties* window. Go to *Function* in order to check the rate functions.
3. Use constants if you want to scale the kinetic parameters; see [6].
4. Moving to the continuous paradigm may require you to adjust the type of some arcs; see also Exercise 7.18b. To change the type of an arc, select the arc in question and go to *Edit* → *Convert To*.

Take the $\mathcal{SPN}$ of the running example and produce its $\mathcal{CPN}$ in two versions—(a) keeping all read arcs, and (b) replacing all read arcs with two opposite arcs. The latter in turn should generate the system of ordinary differential equations (ODEs) given in Table 7.9.

Solution. *Hint.* To edit several net elements at once, select all of them and go to *Edit* → *Edit selected elements*.

(a) See Figure S7.19 for the $\mathcal{CPN}$ of the running example with all read arcs kept as in the $\mathcal{SPN}$, but parameterized by the real-valued constant x.

(b) See Figure S7.20 for the $\mathcal{CPN}$ of the running example with all read arcs replaced by two opposite arcs. □

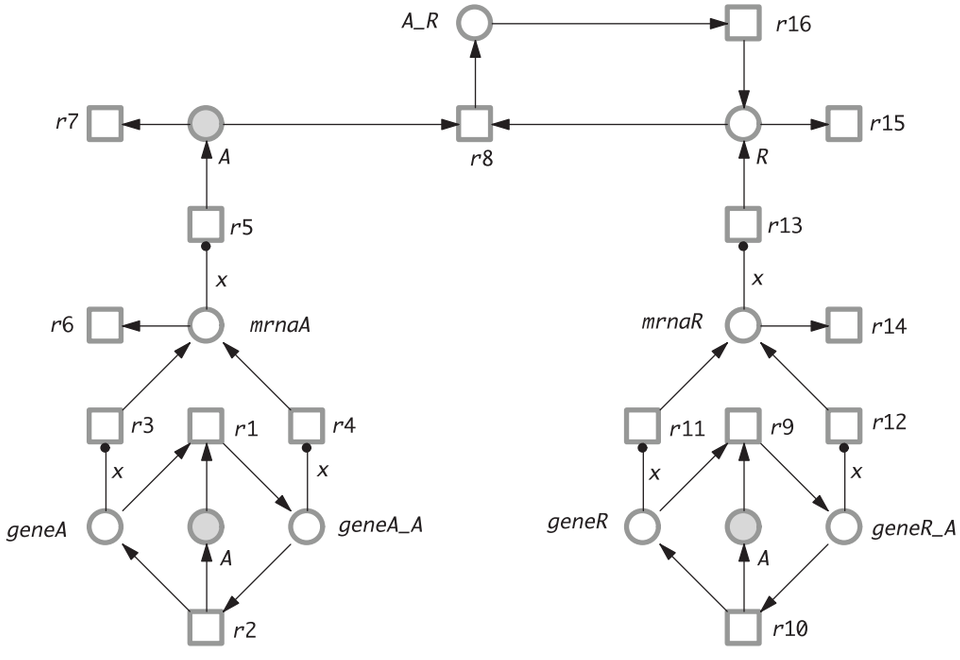


FIGURE S7.19 $\mathcal{CPN}$ of the running example with all read arcs kept as in the $\mathcal{SPN}$.

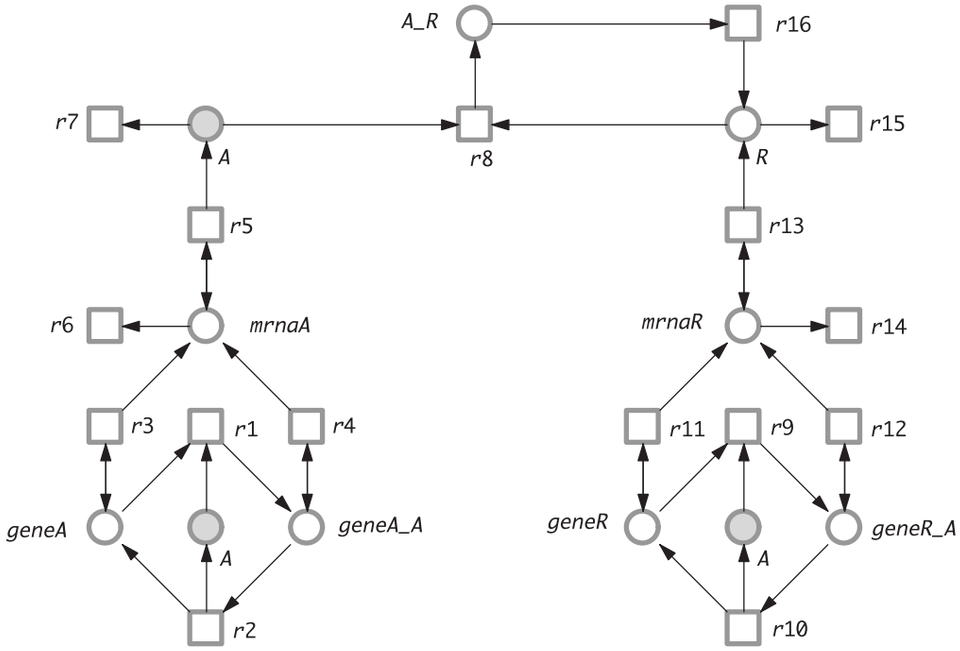


FIGURE S7.20 $\mathcal{CPN}$ of the running example with all read arcs replaced by two opposite arcs.

Exercise 7.18 (Deterministic Simulation)**.** Familiarize yourself with Snoopy's deterministic simulator. Like the stochastic simulation of $\mathcal{SPN}$, the deterministic simulation of $\mathcal{CPN}$ can be parameterized in several ways; see Exercise 7.12. For more details, see also Snoopy's FAQ webpage.

(a) Additionally, one can choose between stiff and unstiff solvers. Explore their differences by playing with the kinetic parameters and initial states of $\mathcal{CPN}_1$, $\mathcal{CPN}_2$, and $\mathcal{CPN}_3$.

(b) In contrast to $\mathcal{SPN}$, read arcs and two opposite arcs usually cause different behavior in the $\mathcal{CPN}$. Take the running example and compare its behavior for the $\mathcal{CPN}$ keeping the read arcs and the $\mathcal{CPN}$ where read arcs are replaced by two opposite arcs; see Exercise 7.17. Play also with the weights of the read arcs, which can now be nonnegative real numbers.

(c) Take the $\mathcal{CPN}$ of the running example and change δ_R to 0.08. How does the dynamic behavior change? Can oscillations still be obtained after this perturbation?

Solution.

(a) *Varying kinetic parameters*: See Figure S7.21 for a couple of simulation experiments.

Hint. Before simulating with Snoopy, try to find the expected plots with paper and pencil.

(b) *Read arcs versus two opposite arcs*: See Figure S7.22 for some plots for different values of the arc weight x of the read arcs, and the left column of Figure S7.23 for two plots when read arcs are replaced.

Hint. There are $\mathcal{CPN}$ scenarios where the switch-like behavior of read arcs is exactly what one wants to have.

(c) *Changing* δ_R to 0.08: See Figure S7.23 for some plots. We recommend comparing these results with the ones of Exercise 7.15.

□

Exercise 7.19 (Transforming ODEs into Continuous Petri Nets)**.** Given are the following ODEs:

$$\frac{dA_1}{dt} = k_2 * A_2 - k_1 * A_1 \qquad \frac{dB_1}{dt} = -k_3 * A_2 * B_1 + k_6 * B_2$$

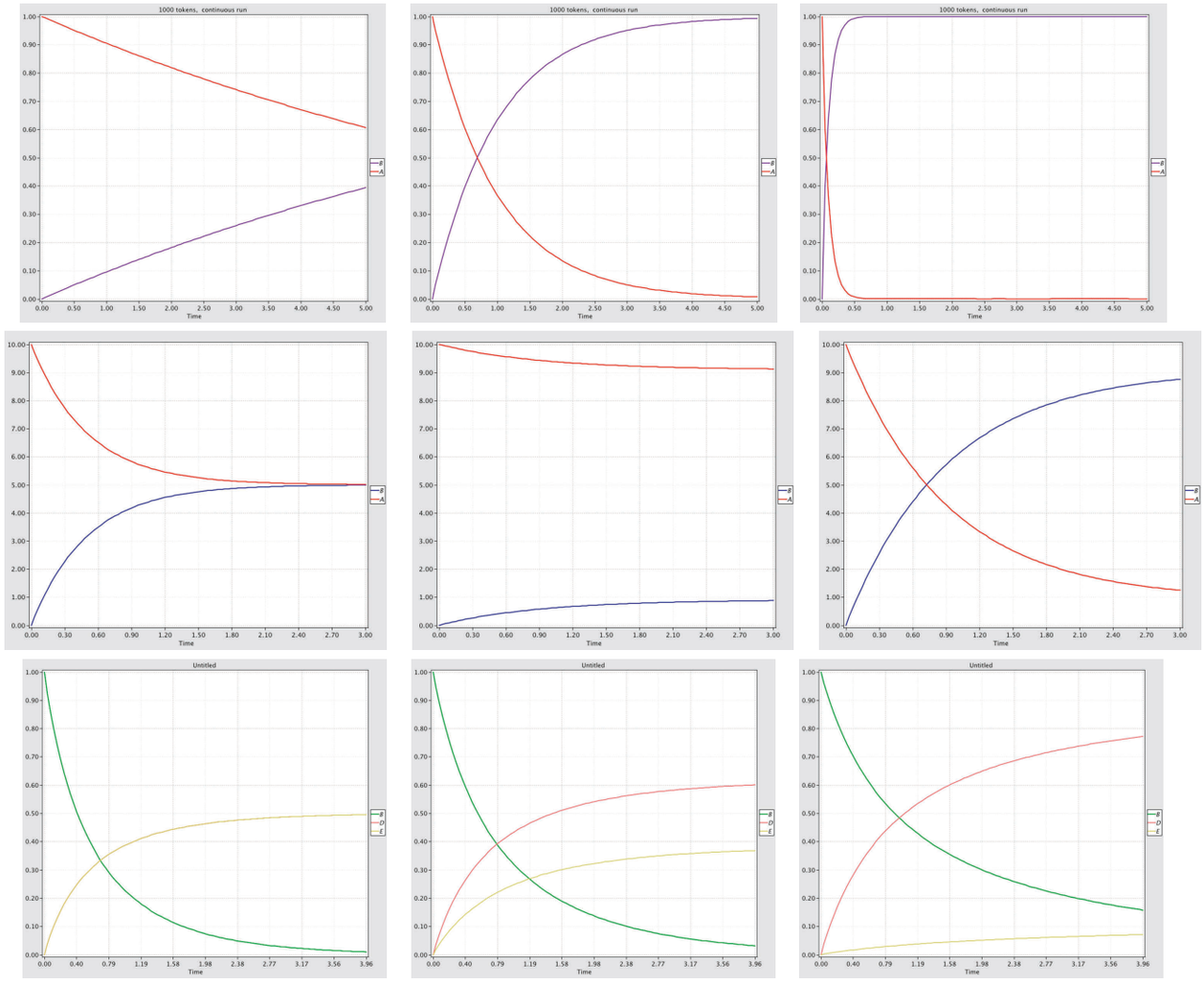


FIGURE S7.21 Playing with kinetic constants. (Top row) $\mathcal{CPN}_1$ with $k = 0.1$ (left), $k = 1$ (middle), $k = 10$ (right); (middle row) $\mathcal{CPN}_2$ with $k1 = 1, k2 = 1$ (left), $k1 = 1, k2 = 0.1$ (middle), $k1 = 0.1, k2 = 1$ (right); (bottom row) $\mathcal{CPN}_3$ with $k1 = 1, k2 = 1$ (left), $k1 = 1, k2 = 0.5$ (middle), $k1 = 1, k2 = 0.05$ (right).

$$\frac{\mathrm{d}A_2}{\mathrm{d}t} = k_1 * A_1 - k_2 * A_2$$

$$\frac{\mathrm{d}B_2}{\mathrm{d}t} = k_3 * A_2 * B_1 - k_4 * A_2 * B_2 + k_5 * B_3 - k_6 * B_2$$

$$\frac{\mathrm{d}B_3}{\mathrm{d}t} = k_4 * A_2 * B_2 - k_5 * B_3$$

(a) Reveal the structure hidden in this ODEs by deriving a corresponding $\mathcal{CPN}$. You can check your solution by drawing the $\mathcal{CPN}$ with Snoopy, and doing *export* $\rightarrow$ *ODEs to text* or *LaTeX*.

(b) Is the solution unique?

(c) Which structural and behavioral properties hold for the corresponding discrete Petri net?

(d) Construct a suitable initial state and explore the discrete and continuous behavior.

Solution. **(a)** *ODEs to* $\mathcal{CPN}$: See screenshot in Figure S7.24 for structure, kinetic parameters, and rate functions (all applying mass-action kinetics). Exporting this $\mathcal{CPN}$ to *txt* gives

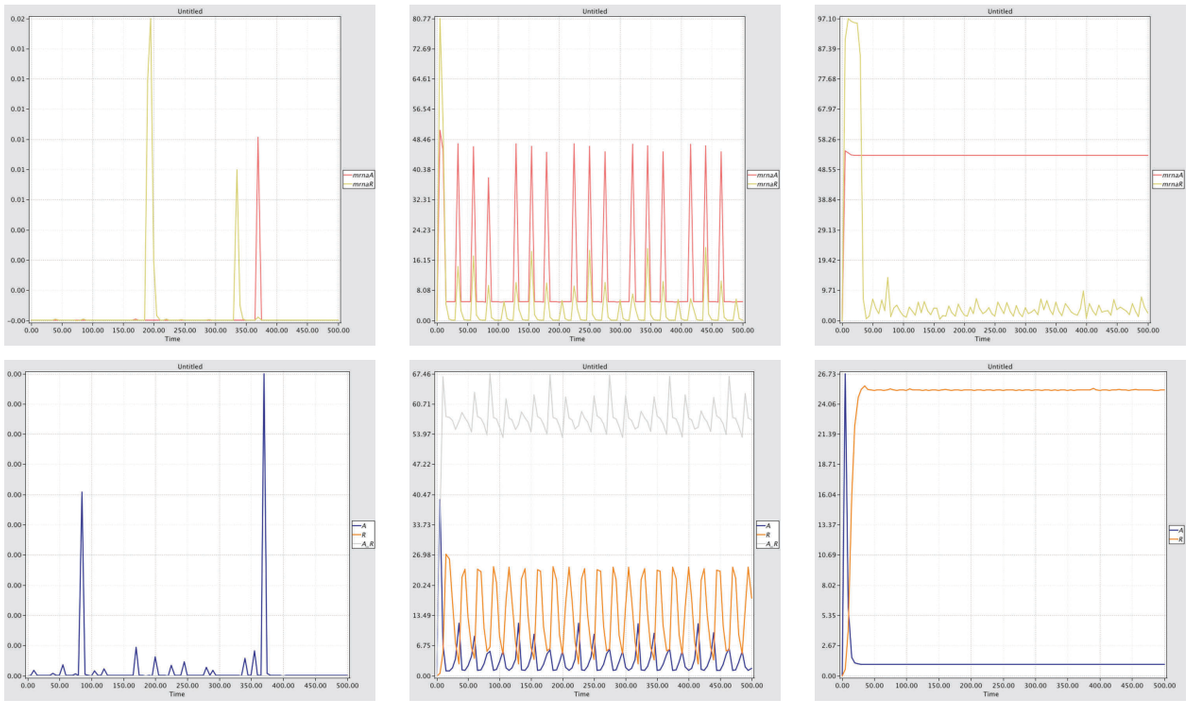


FIGURE S7.22 The effect of read arcs in the continuous paradigm for different arc weights x. (Left) $x = 1$, (middle) $x = 0.1$, (right) $x = 0.01$.

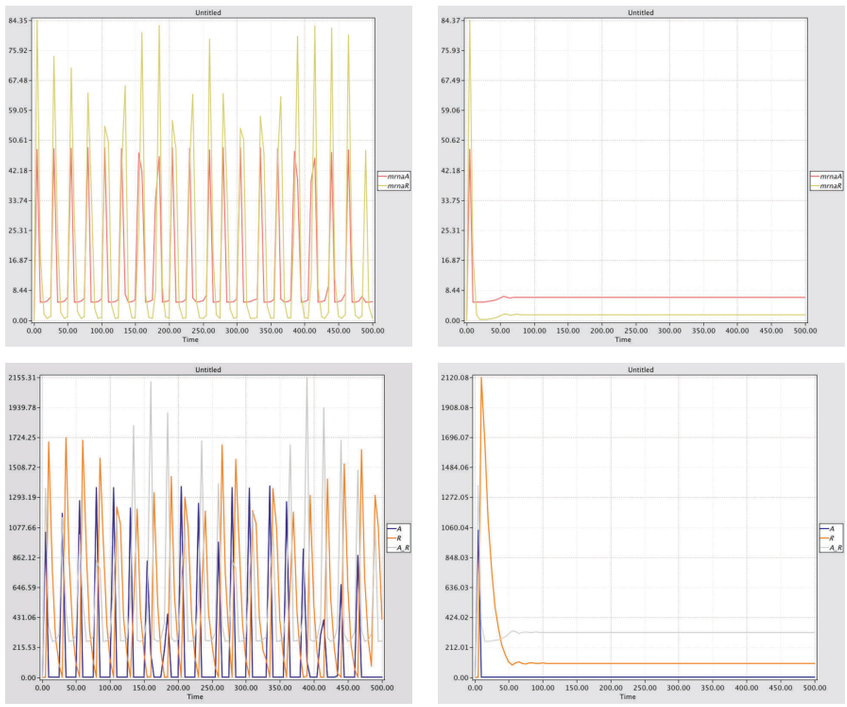


FIGURE S7.23 (Left) $\delta_R = 0.2$, (right) $\delta_R = 0.08$.

```
dA1/dt = (k2*A2)-(k1*A1)
dA2/dt = (k1*A1)+(k3*B1*A2)+(k4*B2*A2)-(k2*A2)-(k3*B1*A2)-(k4*B2*A2)
dB1/dt = (k6*B2)-(k3*B1*A2)
dB3/dt = (k4*B2*A2)-(k5*B3)
dB2/dt = (k5*B3)+(k3*B1*A2)-(k4*B2*A2)-(k6*B2)
```

which is the unreduced version of the given ODEs from which we started.

(b) *Solution uniqueness*: Yes, the solution is unique, what is not necessarily the case; see [7] for three sufficient criteria to obtain a unique structure for a given ODEs.

(c) *Analyzing the corresponding* $\mathcal{PN}$: See Charlie's result vector:

```
PUR   ORD   HOM   NBM   CSV   SCF   FTO   TFO   FPO   PFO   CON   SC    NC
  N     Y     Y     Y     Y     N     Y     Y     Y     Y     Y     Y   nES
RKTH  STP   CPI   CTI  SCTI    SB   k-B   1-B   DCF   DSt   DTr  LIV   REV
  N     N     Y     Y     N     Y     0     Y     Y     1     N    N     Y
```

For more details, see Charlie's session protocol in the Supplementary Material.

The net is covered with P-invariant (CPI), thus it is bounded. It is also strongly connected (SC) and CPI, thus it has a chance to become live as well, but it needs a nonclean initial state for this. The given ODEs do not specify the initial state (initial condition). Thus, we construct it next by exploiting the structural knowledge we just obtained.

(d) *Initial state construction*: This is done best by P-invariant analysis: each minimal P-invariant needs initially some mass (tokens). We use two constants M, N to be flexible; see Figure S7.25. Filling each minimal P-invariant (i.e., $M > 0, N > 0$) makes the $\mathcal{PN}$ live, and the STP holds (which, however, does not allow the conclusion for liveness, as the net structure class is beyond extended simple (ES)); see Charlie's result vector:

```
PUR   ORD   HOM   NBM   CSV   SCF   FTO   TFO   FPO   PFO   CON   SC    NC
  N     Y     Y     Y     Y     N     Y     Y     Y     Y     Y     Y   nES
RKTH  STP   CPI   CTI  SCTI    SB   k-B   1-B   DCF   DSt   DTr  LIV   REV
  N     Y     Y     Y     N     Y     1     Y     N     0     Y    Y     Y
```

See Figure S7.26 for some simulation experiments with the initialized $\mathcal{CPN}$.

□

Exercise 7.20 (Transforming Continuous Petri Nets into ODEs)**.** Find at least three examples of two different $\mathcal{CPN}$s defining the same ODEs, up to some algebraic transformations. *Hint*: Arbitrary rate functions are allowed.

Solution. It is an easy exercise to imagine two reaction networks generating the same incidence matrix or ODEs, but differing in their discrete behavior.

1. Let's start with the introductory example in [8]; see Figure S7.27, left. The net on the right uses a modifier arc (dotted line), which is a standard feature of the Systems Biology Markup Language (SBML), also supported by Snoopy, to indicate that the rate of $r1$ depends on B, but B does not influence the enabledness of $r1$.

With the kinetic rates $v1 = v(r1) = k1 \cdot A \cdot B, v2 = v(r2) = k2 \cdot B$, we get the two equations $\mathrm{d}A/\mathrm{d}t = v2 - v1$, $\mathrm{d}B/\mathrm{d}t = v1 - v2$.

2. The example in Figure S7.28 has been triggered by the reaction network reported in [9]. The network on the left has the structure as suggested by the schematic representation in [9] and the list of reactions in the model's SBML format created by COPASI. The network on the right shows the correct structure, which is hidden in the kinetics of reactions $r23$ and $r25$.

Assigning to $r23$ ($r25$) on the left a rate function that is the sum of the rate functions of $r23a$ and $r23b$ ($r25a$ and $r25b$) on the right, while keeping all other rate functions equal, yields the same ODEs for both networks; compare discussion in [10].

3. See Figure S7.29 for another example, which might also serve as a golden prototype for a perfectly documented solution.

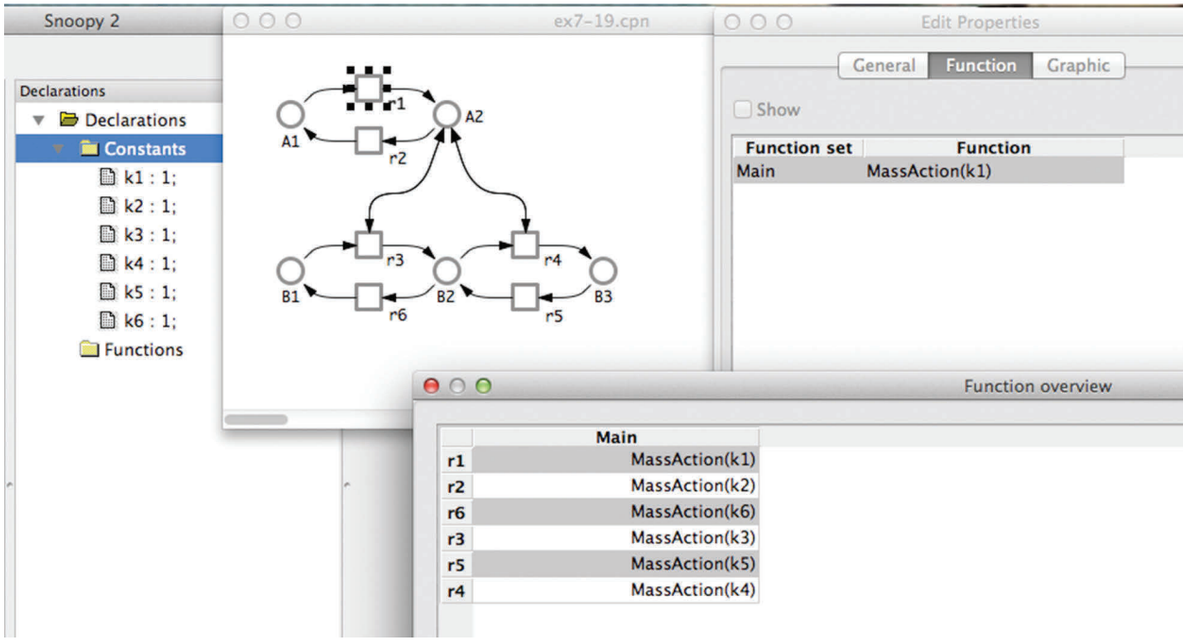


FIGURE S7.24 $\mathcal{CPN}$ for the given ODEs, which come without initial conditions.

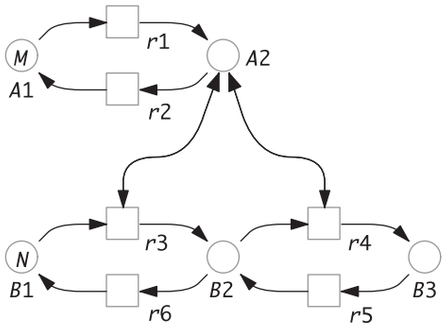


FIGURE S7.25 $\mathcal{CPN}$ for the given ODEs with an initial state constructed with the help of P-invariants. To be flexible, we use two constants M and N.

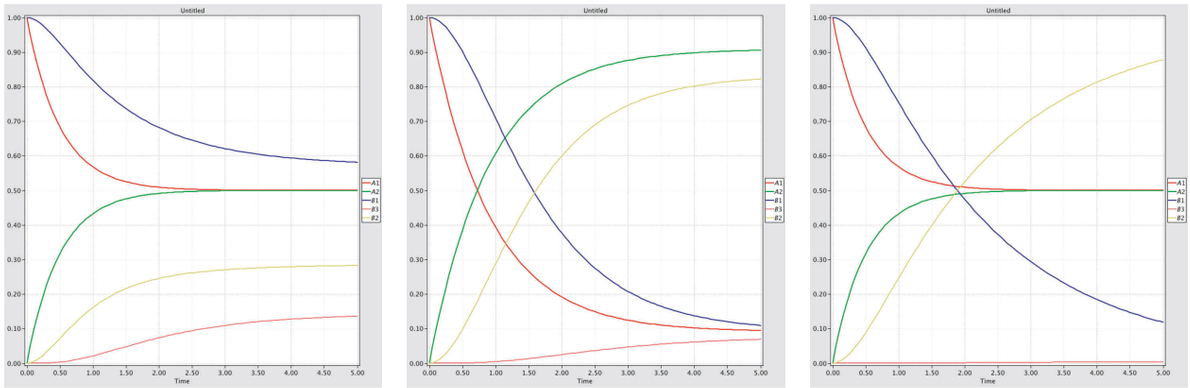


FIGURE S7.26 Varying kinetic parameters for the initialized $\mathcal{CPN}$ in Figure S7.25. (Left) all parameters set to 1; (middle) $k2, k4, k6 = 0.1$, all others 1; (right) $k4, k6 = 0.01$, all others 1.

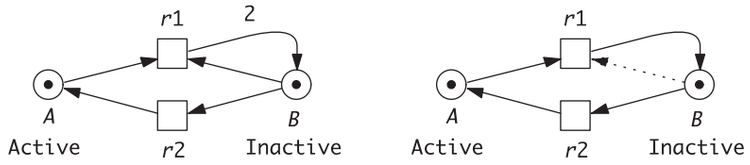


FIGURE S7.27 Two reaction networks generating the same incidence matrix and the same ODEs.

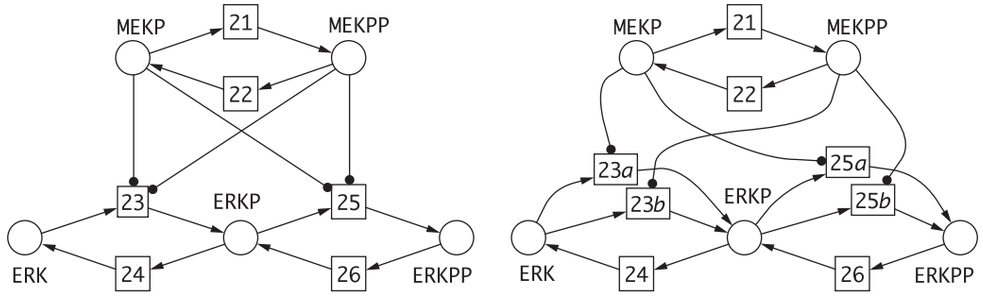


FIGURE S7.28 (Left) Structure as suggested by the schematic representation in [9] and the list of reactions in the model's SBML format. (Right) Correct structure hidden in the kinetics of reactions $r23$ and $r25$.

4. More examples can be found in [7]. These examples also illustrate that extracting the structure out of a given ODEs does not necessarily yield a unique solution.

Lesson learned: When analyzing ODEs, one actually considers a family of related models, varying in some structural details, but generating the same ODEs. □

Exercise 7.21 (Exploring the Limits)**.** We claim that any ODEs (no matter what it looks like) can be generated by a $\mathcal{CPN}$. Give a simple procedure to define such a $\mathcal{CPN}$, assuming the ODEs are given.

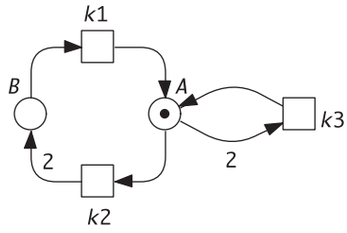
Solution. The answer can be found in [7]. □

Exercise 7.22 (Comparing Stochastic and Deterministic Behavior)**.** Find at least three Petri nets that, when read as $\mathcal{SPN}$ and $\mathcal{CPN}$ with mass-action kinetics, reveal substantially different behavior with respect to their general behavioral properties.

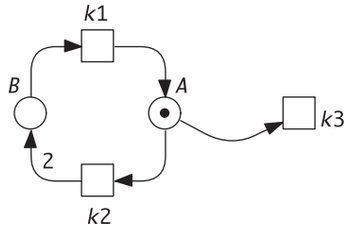
Solution. There are many examples, some of them quite entertaining. You might want to start your own collection.

1. Let's start with a simple in/out flow with a single place; see Figure S7.30. We set $k1 = k2$, and observe how A approaches in the deterministic case the steady-state value 1, without ever exceeding it, while in the stochastic case, A can arbitrarily peak, but with decreasing probability.
2. A popular example is given by the Lotka-Volterra equations, also known as the prey-predator system; see Figure S7.31. Appropriate kinetic parameters will let the $\mathcal{CPN}$ oscillate forever, while the corresponding $\mathcal{SPN}$ will always—independent of the kinetic parameters—have a chance to encounter a dead state sooner or later.
3. One of the counterintuitive effects observable when moving from the discrete to the continuous paradigm is that a trap—a set of places that can never become empty in the discrete case as soon as it has obtained a token—can be emptied in the continuous case; see [11].
4. Another example is the one discussed in Exercise 7.23; further examples can be found in [7, 12].

```
Two continuous Petri nets
- generating the same ODEs
- having different qualitative behavior
```




```
PUR ORD HOM NBM CSV SCF CON SC  FT0 TF0 FP0 PF0 NC
 N   N   N   N   N   N   Y   Y   N   N   N   N  SM
DTP CPI CTI SCTI SB k-B 1-B DCF DSt DTr LIV REV
 Y   N   Y   Y   -   N   N   -   0   -   -   -
```




```
PUR ORD HOM NBM CSV SCF CON SC  FT0 TF0 FP0 PF0 NC
 Y   N   Y   Y   N   N   Y   N   N   Y   N   N  FC
DTP CPI CTI SCTI SB k-B 1-B DCF DSt DTr LIV REV
 N   N   Y   Y   -   N   N   -  1  Y   N   -
```

```
dA/dt =  (k1.B)-(k2.A)-(k3.A.A)
dB/dt = -(k1.B)+(2.k2.A)
```

FIGURE S7.29 Two reaction networks generating the same incidence matrix and the same ODEs, and a demonstration of a well-documented solution.

Lesson learned: It is well known that time assumptions generally impose constraints on behavior. The qualitative and stochastic models consider all possible behavior under any timing, whereas the continuous model is constrained by its inherent determinism to consider a subset. □

Exercise 7.23 (Absolute Concentration Robustness)**.** A place of a $\mathcal{CPN}$ (a variable of an ODEs) is said to have absolute concentration robustness (ACR) if its concentration is the same in all positive steady states; that is, it does not depend on the initial state, but only on the kinetic constants. For illustration, let's explore the following toy two-species mass-action system [8]:

$A + B \xrightarrow{\alpha} 2B\,,\, B \xrightarrow{\beta} A$

(a) Consider this system as $\mathcal{PN}$. Determine the structural properties, and the general behavioral properties. *Hint.* There is a bad siphon.

(b) Consider this system as $\mathcal{CPN}$ and compare its behavior with the $\mathcal{PN}$, while varying the initial state.

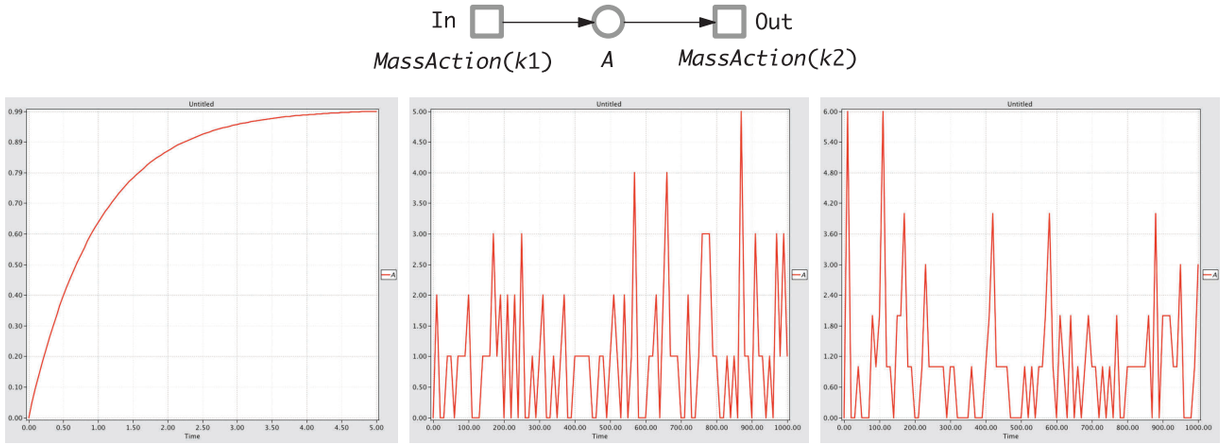


FIGURE S7.30 $\mathcal{CPN}$ ($\mathcal{SPN}$) for a simple in/out flow system, and some simulation experiments: $\mathcal{CPN}$ simulation (left) and two $\mathcal{SPN}$ simulations (middle, right).

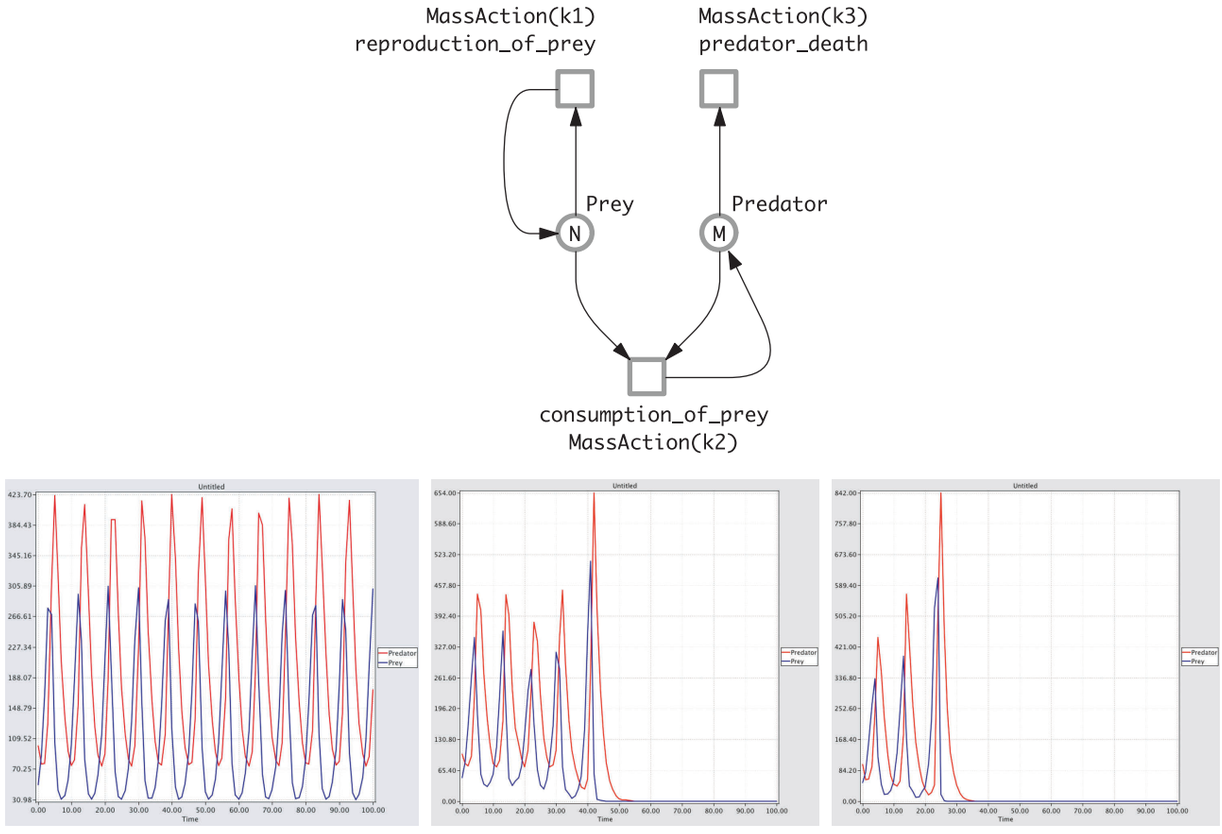


FIGURE S7.31 $\mathcal{CPN}$ ($\mathcal{SPN}$) for the Lotka-Volterra equations, also known as prey-predator system, and some simulation experiments: $\mathcal{CPN}$ simulation (left) and two $\mathcal{SPN}$ simulations (middle, right).

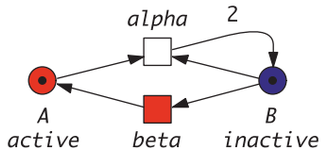


FIGURE S7.32 $\mathcal{PN}$ representation of the introductory example in [8]. A is an ACR place, and B a minimal bad siphon. Both places together form a P-invariant. The transition in red is the troublemaker—the transition responsible for emptying the bad siphon, and thus for the dead state; see [13] for more details.

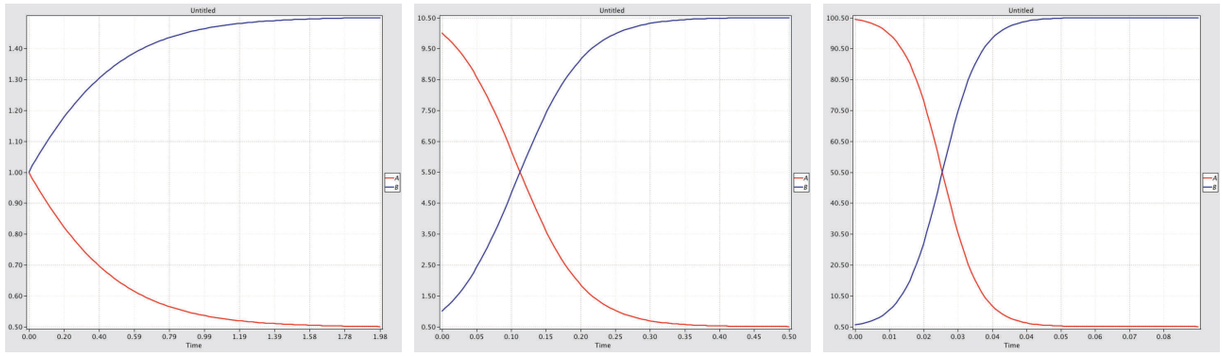


FIGURE S7.33 Some simulation experiments for the $\mathcal{CPN}$ in Figure S7.32, with $\alpha = 2$, $\beta = 1$, and the initial states $A = 1, 10, 100$, while B is always 1, that is, the total mass is 2, 11, or 101. Note the stable steady-state values for A: independent of the total mass, there is always 0.5 mass that accumulates on A, while all the other mass finally accumulates on B.

Solution. **(a)** *Considering as* $\mathcal{PN}$: The $\mathcal{PN}$ is given in Figure S7.32.

PUR	ORD	HOM	NBM	CSV	SCF	FTO	TFO	FPO	PFO	CON	SC	NC
N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	ES
RKTH	STP	CPI	CTI	SCTI	SB	k-B	1-B	DCF	DSt	DTr	LIV	REV
N	N	Y	Y	-	Y	2	N	N	1	Y	N	N

The place B forms on its own a bad siphon, that is, a siphon not containing a trap . It is easy to see that the net does not have a live initial marking in the discrete world. Thus, when the system is considered in the discrete world (which includes the stochastic paradigm), there is always a dead state reachable, in which the bad siphon, that is, B, is empty and all tokens reside on A.

Hint. Two given $\mathcal{PN}$ and $\mathcal{SPN}$ sharing structure also share all qualitative properties. Thus, talking about a $\mathcal{PN}$ immediately involves talking about the corresponding $\mathcal{SPN}$.

(b) *Considering as* $\mathcal{CPN}$: The place A is an ACR place; thus, its steady-state concentration is β/α (independent of the initial state), while B has the steady-state concentration $total - \beta/\alpha$, with *total* being the conserved total mass in the network, because the net is covered by one minimal P-invariant.

There is no (obvious) sign of the dead state when reading the net as $\mathcal{CPN}$. Moreover, increasing the mass in the closed system will increase the steady value of B, as A enjoys ACR. Thus, the continuous model may predict a simplified behavior that does not reflect the variety, which is possible in the discrete case, for example, in the given example that the siphon will eventually get empty.

Hint. Here, we confine ourselves to exploring $\mathcal{CPN}$ behavior by deterministic simulation, see Figure S7.33. A closed solution of the ODEs generated by the $\mathcal{CPN}$ actually gives some indication of the dead state.

Figure S7.34 gives some suggestions for further exercises of this type, taken from the Supplementary Material of [8]. □

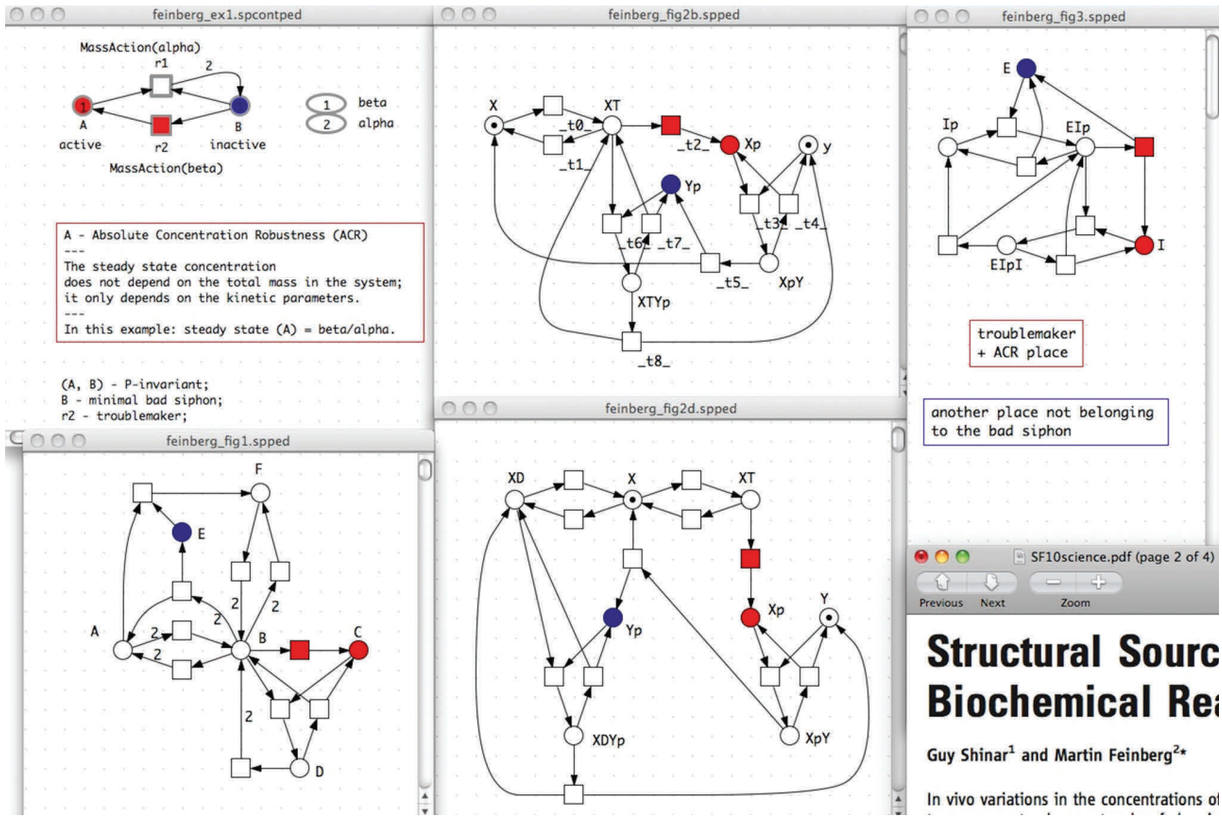


FIGURE S7.34 Some suggestions for further exercises of this type, taken from the supplementary material of [8], see lower right corner.

7.4 HYBRID PETRI NETS ($\mathcal{HPN}$)

Exercise 7.24 (Changing the Net Class)**.** As before, an $\mathcal{HPN}$ does not have to be built from scratch if a corresponding $\mathcal{SPN}$ or $\mathcal{CPN}$ already exists; compare Exercises 7.11 and 7.17. However, static partitioning additionally requires us to change the types of nodes and/or arcs appropriately. This option is found in *Edit* → *Convert To*.

Take the $\mathcal{SPN}$ and $\mathcal{CPN}$ of the running example and find for each a suitable order of changes to be made to produce the $\mathcal{HPN}$ shown in Figure 7.22.

Solution. When constructing an $\mathcal{HPN}$ from a given $\mathcal{SPN}$ or $\mathcal{CPN}$, it might be a challenge for beginners to find a suitable order of the changes to be made such that the $\mathcal{HPN}$ under construction obeys all the time the given syntactical constraint: discrete places can never be connected with continuous transitions by standard arcs. This constraint should be easy to memorize, as the continuous flow of a continuous transition obviously contradicts the discrete number of tokens allowed to be on discrete places.

But the other way around does not contradict any rules: a continuous place can be connected with discrete transitions by standard arcs. That's all we need to keep in mind to come up with the following construction steps. Do not worry; any attempted conversions contradicting the rules are refused, and a message appears in the log window.

- *Starting with $\mathcal{SPN}$:*
  1. Open the $\mathcal{SPN}$, export it to $\mathcal{HPN}$ by *File* → *Export as*, and select $\mathcal{HPN}$.
  2. Open the just generated $\mathcal{HPN}$.
  3. Select the discrete places $A, R, A\_R, mrnaA, mrnaR$, and convert the type of these places all at once by *Edit* → *Convert To*, choosing *continuous*; compare Figure S7.35.

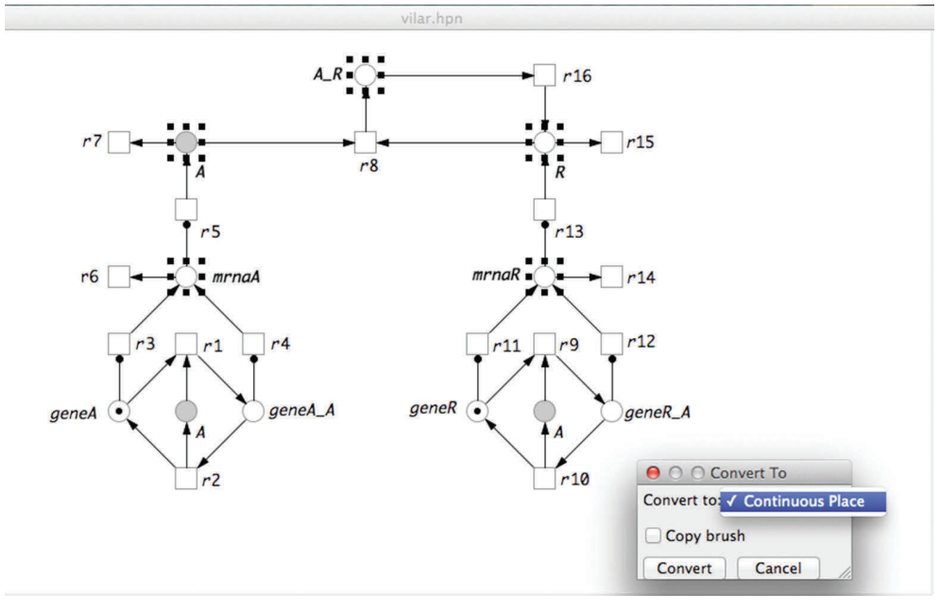


FIGURE S7.35 Converting a couple of places by *Edit* → *Convert To*, with just a few mouse clicks.

4. Analogously, select the discrete transitions $r5$-$r8$, and $r13$-$r16$, and convert the type of these transitions all at once by *Edit* → *Convert To*, choosing *continuous*.
5. Select the read arcs between *mrnaA* → $r5$, and *mrnaR* → $r13$, and convert the type of these arcs all at once by *Edit* → *Convert To*, choosing *edge* (which is just a synonym for arc). You get an arc each going from the place to the transition, which needs to be complemented by the opposite arcs.
6. Do not forget to save your result.

- *Starting with* $\mathcal{CPN}$:
  1. Open the $\mathcal{CPN}$, export it to $\mathcal{HPN}$ by *File* → *Export as*, and select $\mathcal{HPN}$.
  2. Open the just generated $\mathcal{HPN}$.
  3. Select the continuous transitions $r1$-$r4$, and $r9$-$r12$, and convert the type of these transitions all at once by *Edit* → *Convert To*, choosing *stochastic*; see Figure S7.36.
  4. Analogously, select the continuous places *geneA*, *geneA_A*, *geneR*, and *geneR_A*, and convert the type of these places all at once by *Edit* → *Convert To*, choosing *discrete*.
  5. Getting the read arcs as required is slightly more tricky, as we need to select the arcs going from the place to the transition, which are converted into a read arc, while the opposite arcs, going from the transition to the place, have to be deleted.
  6. Do not forget to save your result.

  Obviously, in the given case, it is slightly easier to start with the $\mathcal{SPN}$ due to the read arcs. □

Exercise 7.25 (Hybrid Simulation)**.** Familiarize yourself with Snoopy's hybrid simulator. Like the stochastic simulation of $\mathcal{SPN}$ (see Exercise 7.12) and the deterministic simulation of $\mathcal{CPN}$ (see Exercise 7.18), the hybrid simulation of $\mathcal{HPN}$ can be parameterized in several ways. For more details, see also Snoopy's FAQ webpage.

(a) Additionally, one can choose among the following synchronization principles.

- *Static*: Each transition type is kept as it has been determined by the user.
- *Dynamic*: The types of stochastic and continuous transitions are adjusted on the fly by evaluating the current transition rates.
- *Continuous*: The entire net is considered as $\mathcal{CPN}$ and thus simulated continuously. Any stochastic transition is automatically converted to a continuous one.

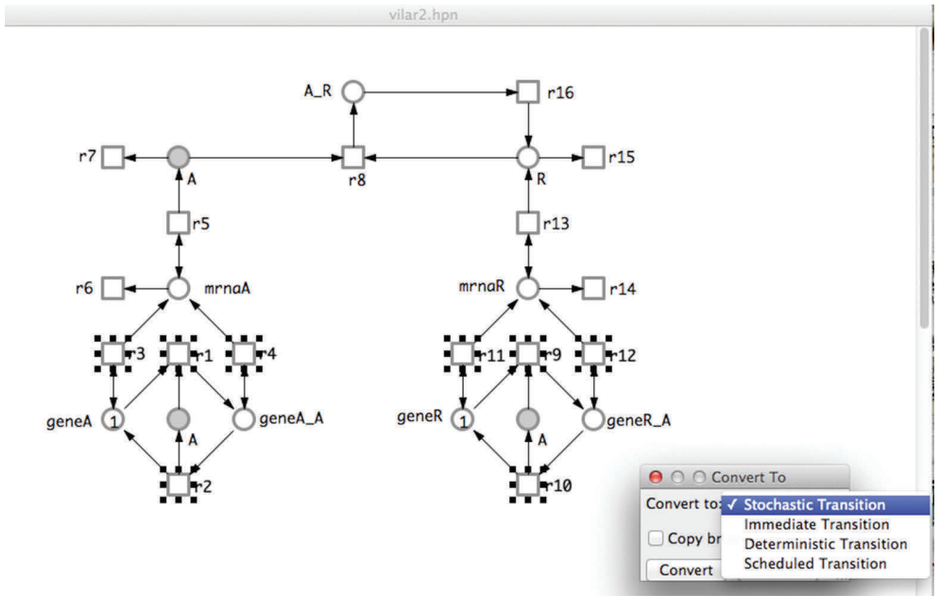


FIGURE S7.36 Converting a couple of transitions by *Edit* → *Convert To*, with just a few mouse clicks.

- *Stochastic*: The entire net is considered as $\mathcal{SPN}$ and thus simulated stochastically. Any continuous transition is automatically converted to a stochastic one.

Hint. Immediate transitions are not affected. Explore the differences between these synchronization principles by playing with $\mathcal{HPN}_1$ and $\mathcal{HPN}_2$. For this purpose, introduce more constants to increase the flexibility of the models, as demonstrated in Figure 7.18.

(b) The $\mathcal{HPN}$ of the running example uses read arcs for stochastic nodes, and two opposite arcs for continuous nodes; see Figure 7.22. Compare its behavior with the two extreme cases:

- All side conditions are modeled by read arcs. Play also with the weights of the read arcs, which can now be nonnegative real numbers.
- All side conditions are modeled by two opposite arcs.

(c) Take the $\mathcal{HPN}$ of the running example and change δ_R to 0.08. How does the dynamic behavior change? Can oscillations still be obtained after this perturbation?

Solution. **(a)** *Exploring* $\mathcal{HPN}_1$, $\mathcal{HPN}_2$: It does not make much sense to play with the initial marking, as exactly one token in the discrete control cycle is all we need to get the required effect, and a flexible initial state of A would require us to set the initial token in the control cycle appropriately.

But we can increase the flexibility by adding a couple of constants; see screenshot in Figure S7.37 and some plots of related computational experiments in Figure S7.38. For example, varying *k_off* in $\mathcal{HPN}_1$ influences the range of values of A and their probability, while varying $k1$ in $\mathcal{HPN}_2$ influences the time required to fill A.

While varying constants, try to find the plots with paper and pencil before asking Snoopy.

(b) *Exploring the running example in three versions*: Having practiced Exercise 7.24, it should not be time-consuming to produce the two additional versions of the running example (*vilar1.hpn*; see Figure 7.22) as suggested; let's call them *vilar2.hpn* (all read arcs) and *vilar3.hpn* (all read arcs replaced by two opposite arcs); see Figure S7.39.

Some related computational experiments are given in Figure S7.40.

(c) *Changing* δ_R: See Figure S7.40, first column, second row. An explanation for the changes in the oscillation is given in the solution of Exercise 7.15a.

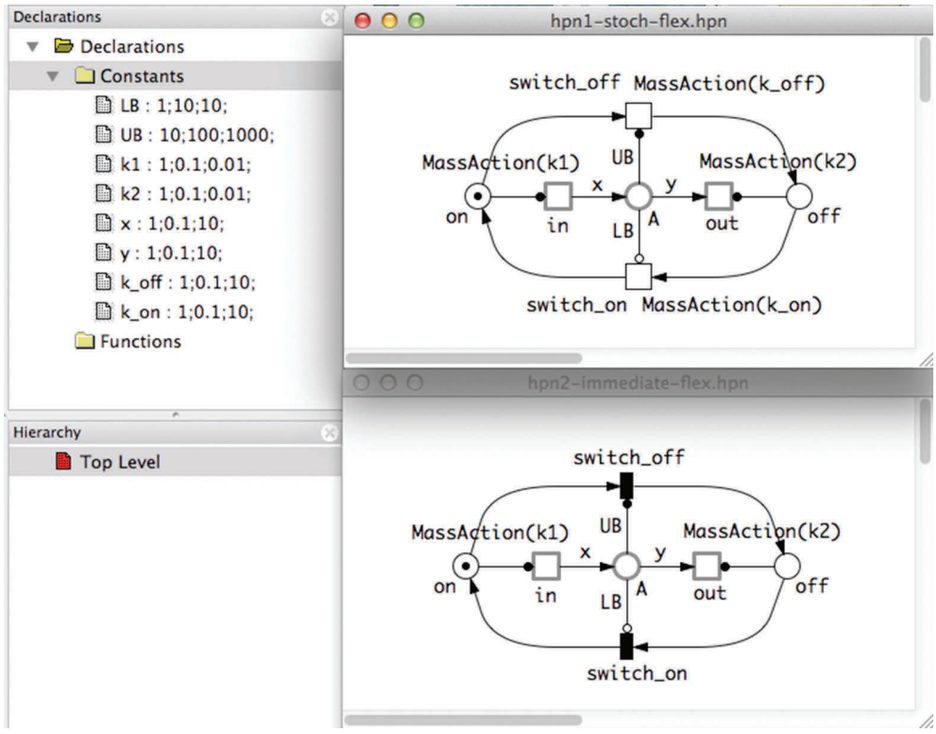


FIGURE S7.37 Flexible variants of $\mathcal{HPN}_1$ and $\mathcal{HPN}_2$ by introducing a couple of constants.

□

Exercise 7.26 (Transition Types)**.** The control cycles in $\mathcal{HPN}_1$ and $\mathcal{HPN}_2$ (see Figure 7.20) deploy stochastic or immediate transitions. There are two further transition types.

- *Deterministic transitions*: The firing occurs after a deterministic firing delay, which is specified by an integer constant. The delay is always relative to the time point where the transition gets enabled. The transition may lose its enabledness while waiting for the delay to expire.
- *Scheduled transitions*: The firing occurs according to a schedule specifying absolute points of the simulation time. A schedule can specify just a single time point, or equidistant time points within a given interval, triggering the firing once or periodically. However, transitions only fire at their scheduled time points if they are enabled at this time.

Create $\mathcal{HPN}_3$ and $\mathcal{HPN}_4$ by replacing the transitions in the control cycle with deterministic or scheduled transitions, and explore their behavior.

Solution. $\mathcal{HPN}_1$ and $\mathcal{HPN}_2$ come with the book chapter. Thus, creating $\mathcal{HPN}_3$ and $\mathcal{HPN}_4$ requires just a few clicks, when deploying the feature demonstrated in Figure S7.41. We obtain the four $\mathcal{HPN}$s shown in a screenshot in Figure S7.42. They can be easily explored by hybrid simulation; some simulation plots are given in Figure S7.43. □

7.5 COLORED PETRI NETS

Exercise 7.27 (Folding and Unfolding)**.** Let's assume we have a network consisting of five components being reversible reactions of the type given in Figure 7.11. We could construct it by copy, paste five times, and appropriate renaming. Instead, we want to use color.

(a) Turn the $\mathcal{SPN}_2$ in Figure 7.11 into a scalable $\mathcal{SPN}^{\mathcal{C}}$, allowing for a flexible number of reversible reactions.

1. Start by exporting the $\mathcal{SPN}$ to $\mathcal{SPN}^{\mathcal{C}}$, which applies a trivial coloring scheme: all places get assigned the color set *Dot*, and all arcs the color value *dot*.

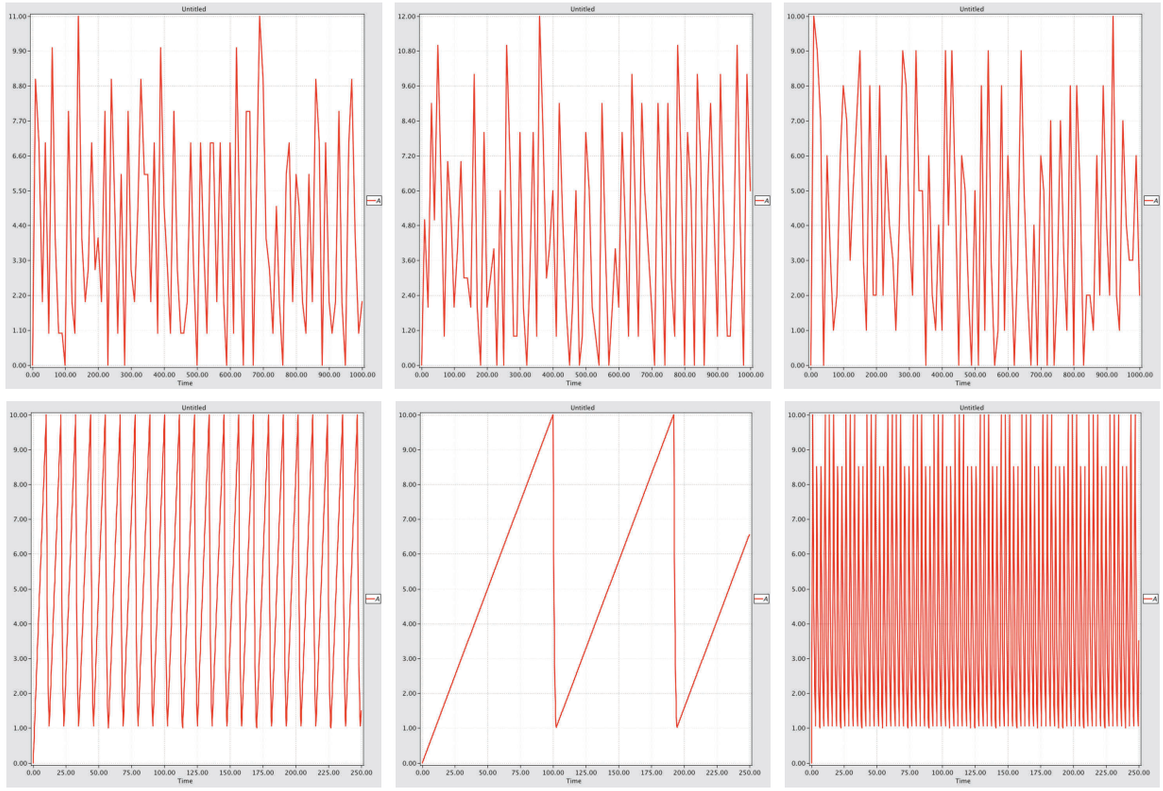


FIGURE S7.38 Some simulation experiments for the $\mathcal{HPN}$s in Figure S7.37. First row, $\mathcal{HPN}_1$ with $k\_off = 1, 0.1, 10$ (from left to right); second row, $\mathcal{HPN}_2$ with $k1 = 1, 0.1, 10$ (from left to right).

2. Open the generated $\mathcal{SPN}^{\mathcal{C}}$ and add color-related definitions as required. Use a constant to specify the number of components (reversible reactions) to keep your model flexible.
3. Simulate your $\mathcal{SPN}^{\mathcal{C}}$ for a varying number of components.
4. Explore different simple data types for this task. Which data type fits your purpose best?

(b) Verify the correctness of your solution by reversing the coloring.

1. Export your $\mathcal{SPN}^{\mathcal{C}}$ to $\mathcal{SPN}$, which involves unfolding.
2. Open the generated unfolded $\mathcal{SPN}$, and do *Edit* $\rightarrow$ *Layout* to automatically generate a more readable net layout.
3. Do steps 1 and 2 for different numbers of components.

Solution.

(a) Components can be added to the colored Petri net generated in Step 1 by following the procedure below.

1. Add a new constant, for example, N of type *int* with the value 5.
2. Add a new simple color set, for example, *component* of type *int* with colors $1 - N$, which gives you the colors $1, 2, 3, \ldots, N$.

   Hint. The type of the color set and the constant need to match each other.
3. Add a new variable, for example, x of the color set *component*.
4. Change the color set of all places from *Dot* to *component*, and replace the token of the color *dot* in the marking by the function *all*().

   Hint. The function *all*() selects all colors of the color set assigned to a place.
5. Change the arc expression 1‘*dot* to x (x is equivalent to 1‘x).

See also Figure S7.44 for an illustration of the modeling procedure. The simulation result of the finally colored model with $N = 5$ components is shown in Figure S7.45. The data type *int* fits best in order to use a

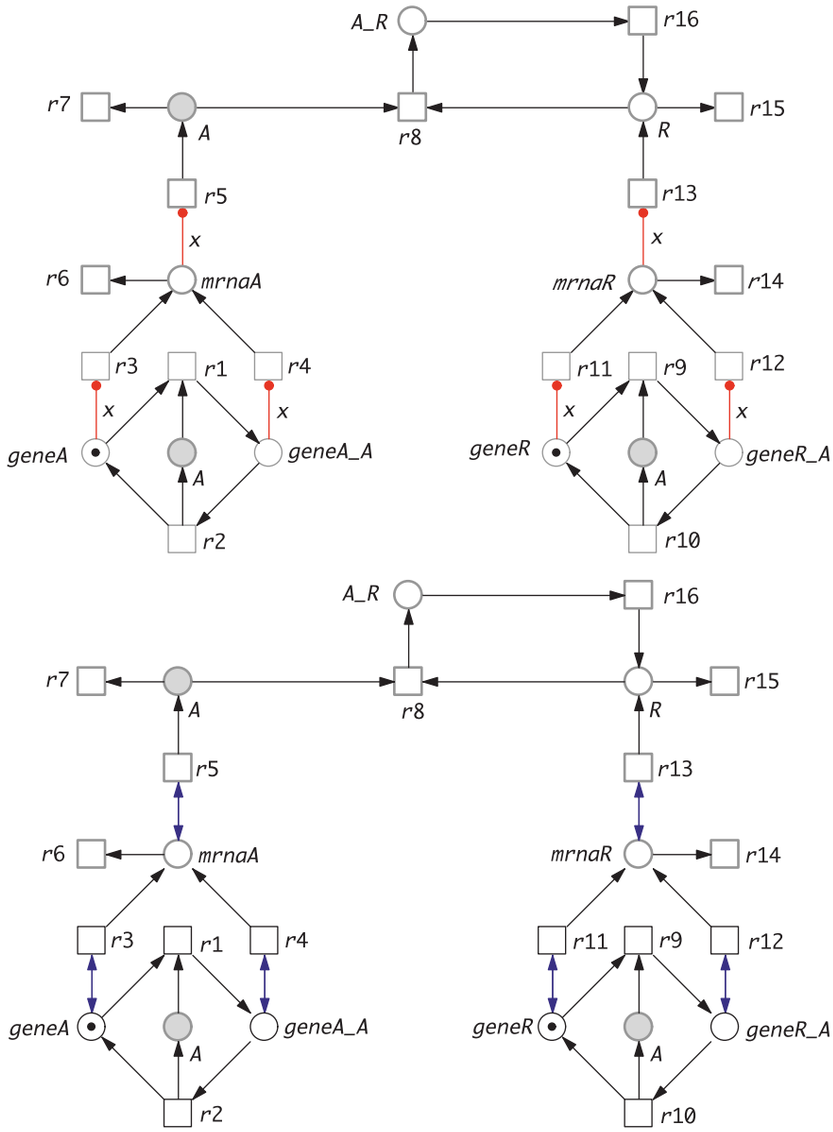


FIGURE S7.39 The running example in two $\mathcal{HPN}$ versions; *vilar2.hpn* (top), *vilar3.hpn* (bottom).

shorthand notation (e.g., $1 - N$) with a constant (e.g., N) to specify the number of colors. An integer constant cannot be used with type *string*, *enum*, and *index*. The type *Dot* only allows one single color, and the type *bool* restricts the number of colors to *true* and *false*.

(b) The simulation results of the unfolded low-level model, which are given in Figure S7.46, correspond to those given in Figure S7.45.

□

Exercise 7.28 (Running Example with Multiple Repressors)**.** Take the $\mathcal{HPN}^{\mathcal{C}}$ of the running example in Figure 7.24.

(a) To scale the number of repressors, follow the steps below.

1. Change the colors of the color set *Gene* to $A, R1, R2, R3$.
2. Assign the color set *Gene* to the place C.
3. Change the arc expression of $r8 \to C$, $C \to r16$, and $r16 \to P$ to x.

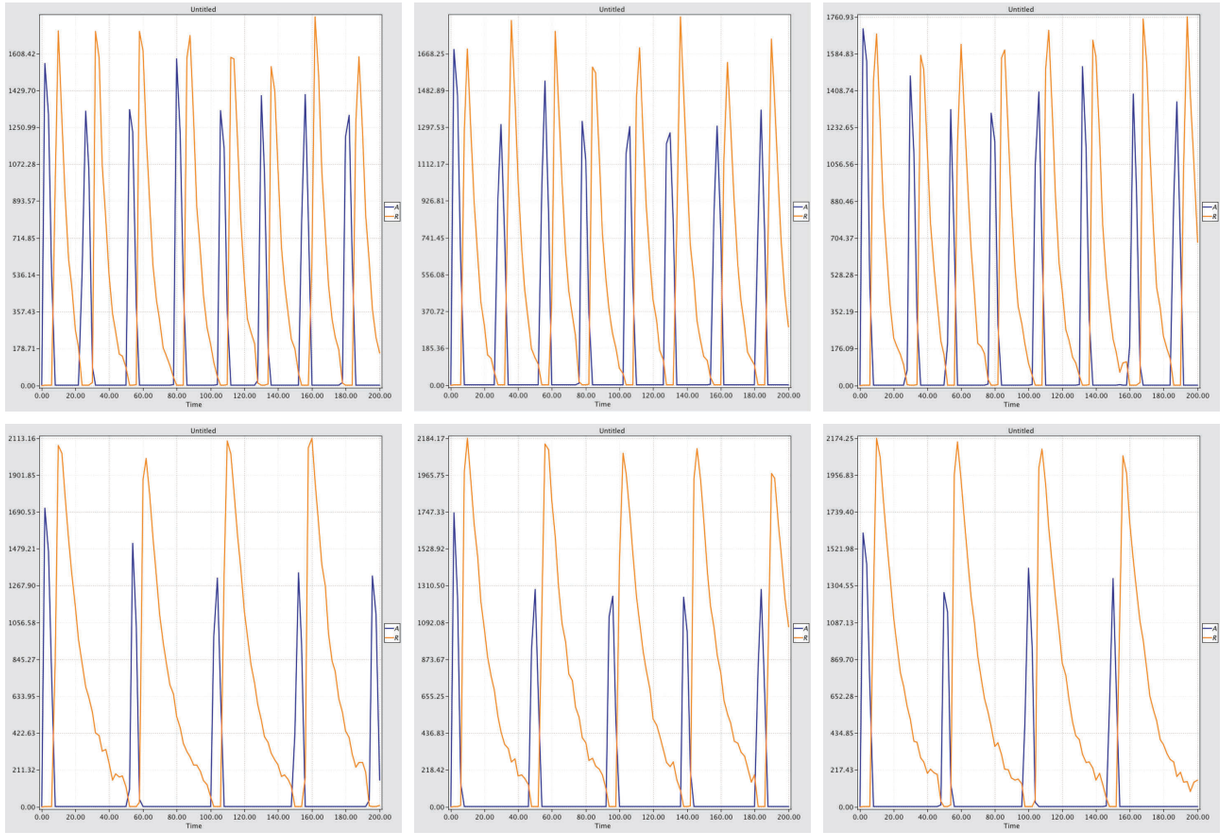


FIGURE S7.40 Some simulation experiments of the running example in three $\mathcal{HPN}$ versions: (from left to right) *vilar1.hpn*, *vilar2.hpn*, *vilar3.hpn*; (first row) $\delta_R = 0.2$, (second row) $\delta_R = 0.08$.

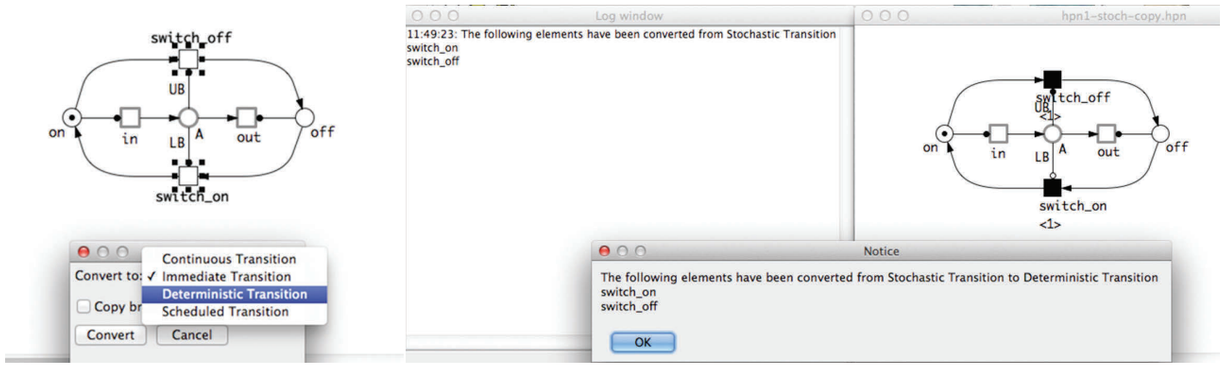


FIGURE S7.41 Converting transition types by *Edit* → *Convert To*. There is just one mouse click from the screenshot on the left to the screenshot on the right.

4. Change the arc expression of $P \rightarrow r8$ to $1'A + +x$.
5. Change all predicates of the firing rates for $x = R$ to $x <> A$.
6. Multiply the constants in the firing rates for the predicate $x = A$ with the number of repressors (e.g., $n = 3$, $MassAction(n * k)$).

(b) Explain the necessity of Steps 5 and 6.

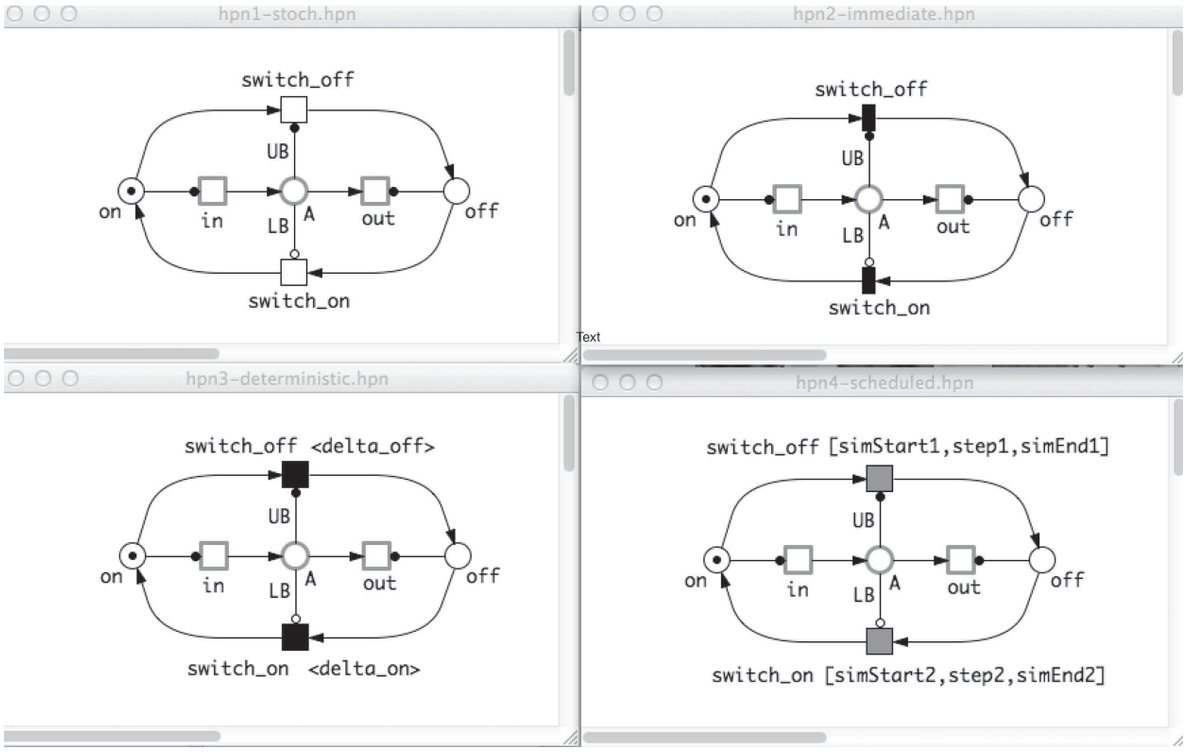


FIGURE S7.42 Control cycle in four versions: $\mathcal{HPN}_1$—stochastic transitions, $\mathcal{HPN}_2$—immediate transitions, $\mathcal{HPN}_3$—deterministically delayed transitions, and $\mathcal{HPN}_4$—scheduled transitions.

(c) Simulate the scalable $\mathcal{HPN}^{\mathcal{C}}$ constructed in (a) to create traces as shown in Figure 7.25. Try different simulation options.

(d) Explore different simple data types for this task. Which data type fits your purpose best?

Solution. **(a)** Detailed instructions for constructing the solution are already given in the exercise itself; compare Figure S7.47.

(b) In Step 5, the boolean expression $x <> A$ serves as a predicate to extract the colors representing the repressor species. The firing rates, which were previously assigned to the single repressor in the basic model, are now also valid for the additionally added repressors. Step 6 has to be done in order to adjust the kinetics of the activator A with respect to the number of different repressor species given by n. In other words, to preserve the behavior of the basic model, the activator A has to act n-times faster. Therefore, the constants in the firing rates are multiplied by n.

(c) See Figure S7.48.

(d) In this case, the data types *int*, *enum*, and *string* are equally good choices. A suitable color set could be defined in the following ways:

- *Genes* := *int* with $0, 1, 2, 3$ (color 0 representing the activator, colors $1, 2, 3$ representing the repressor species.
- *Genes* := *string* with $A, R1, R2, R3$.
- *Genes* := *enum* with $A, R1, R2, R3$.

All other expressions given in (a) have to be changed according to the used color set definition.

□

Exercise 7.29 (Running Example with Activators/Repressors in a Row or Circle)**.** Instead of having several repressors that act on one activator, let us assume that each protein activates its upstream neighbor and inhibits its downstream neighbor. Two interaction patterns are possible.

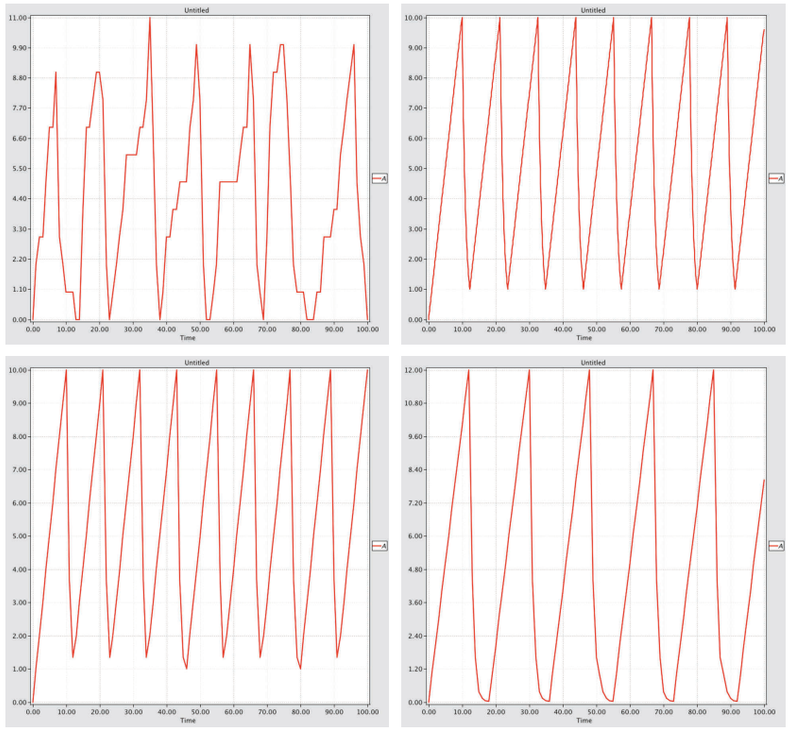


FIGURE S7.43 Some simulation experiments for $\mathcal{HPN}_1$-$\mathcal{HPN}_3$ in Figure S7.42, with *delta_off* $= 0$, *delta_on* $= 0$ (left), and *delta_off* $= 1$, *delta_on* $= 2$ (right).

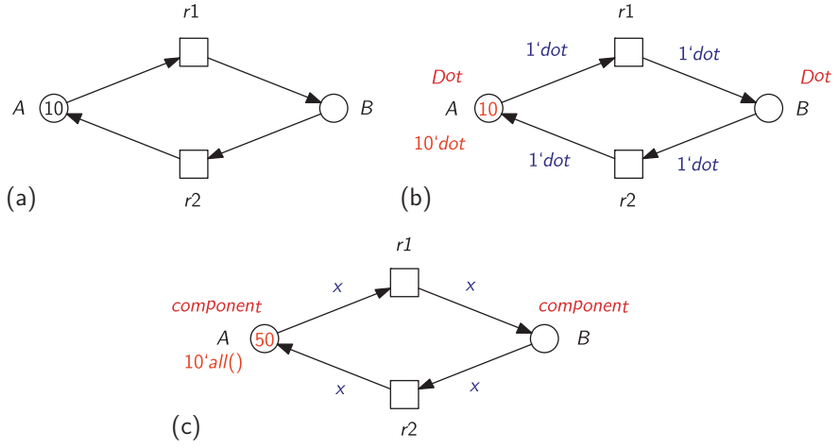


FIGURE S7.44 Coloring procedure. (a) Low-level Petri net; (b) colored Petri net with default declarations; (c) colored Petri net with color set *component* of type *int* with colors $1 - N$, constant $N = 5$, and the variable x of color set *component*.

(a) *Row*: The first protein has no downstream neighbor, and the last protein has no upstream neighbor.
(b) *Circle*: The last protein is the downstream neighbor of the first protein, and thus the first protein is the upstream neighbor of the last protein.

Start with one of the colored models in Figure 7.24 and modify it to represent the cases (a) and (b). *Hint*: For a variable x of integer data type, $val(x)$ yields the value of the currently bound color.

Solution. Starting with the $\mathcal{HPN}^{\mathcal{C}}$ of the running example in Figure 7.24, the following steps have to be performed.

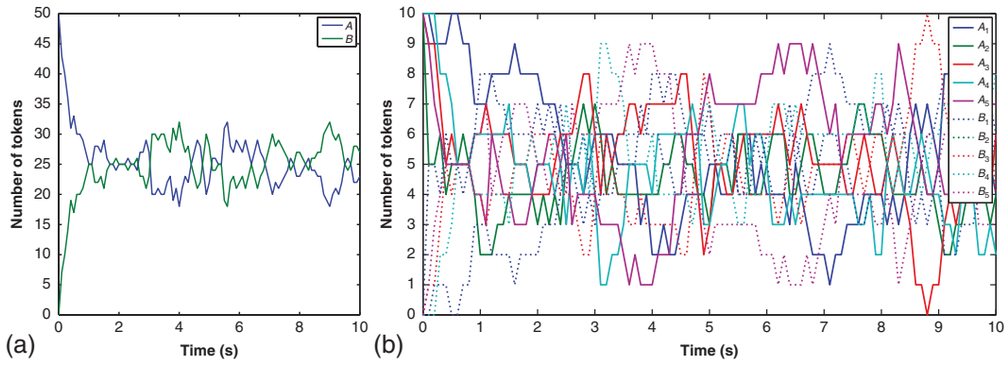


FIGURE S7.45 Simulation results. The plots show the simulation results for the (a) colored places and (b) uncolored places.

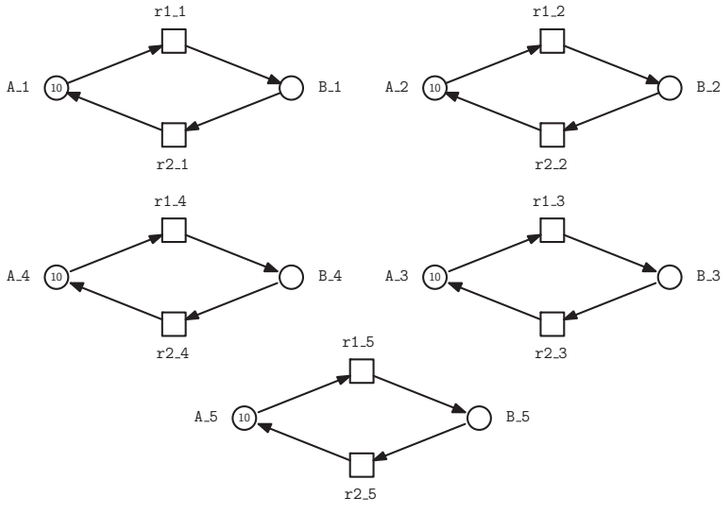


FIGURE S7.46 Unfolded model of the $\mathcal{SPN}^{\mathcal{C}}$ in Figure S7.44c with $N = 5$.

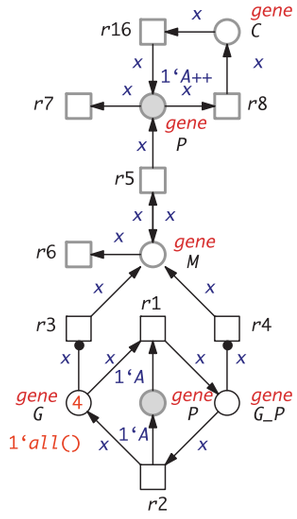


FIGURE S7.47 $\mathcal{HPN}^{\mathcal{C}}$ representing a single-activator protein acting on three repressor proteins.

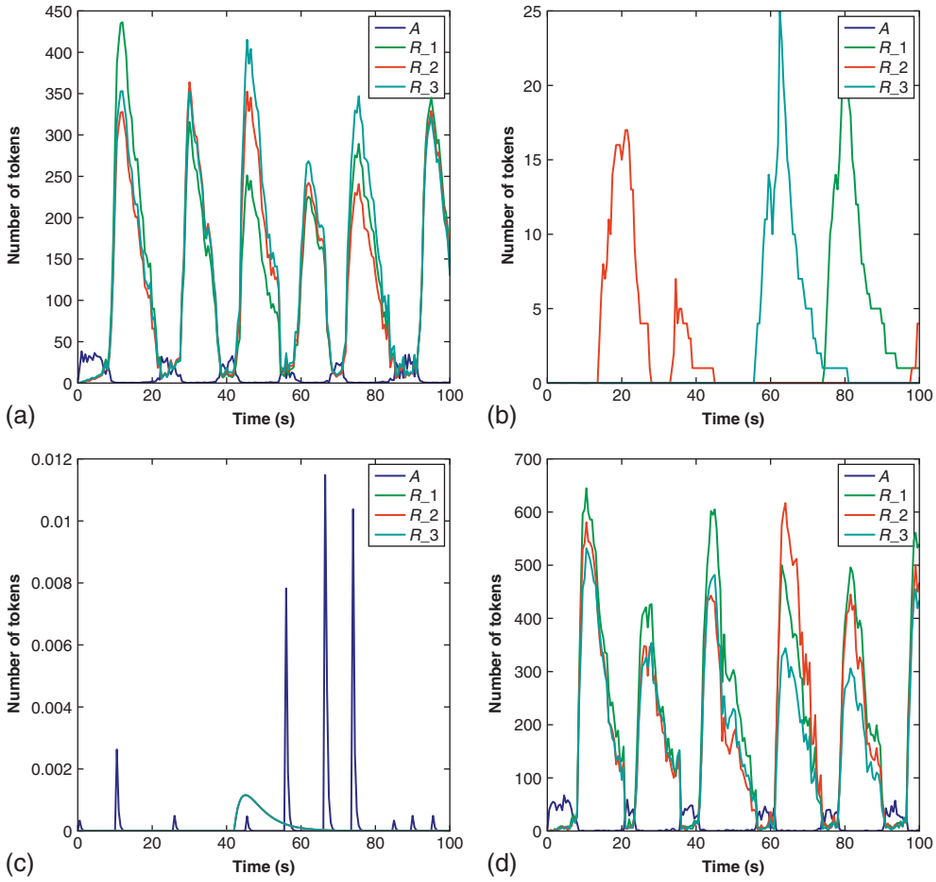


FIGURE S7.48 Comparison of simulation methods applied to a scalable $\mathcal{HPN}^{\mathcal{C}}$: (a) static synchronization, (b) dynamic synchronization, (c) continuous synchronization, and (d) stochastic synchronization.

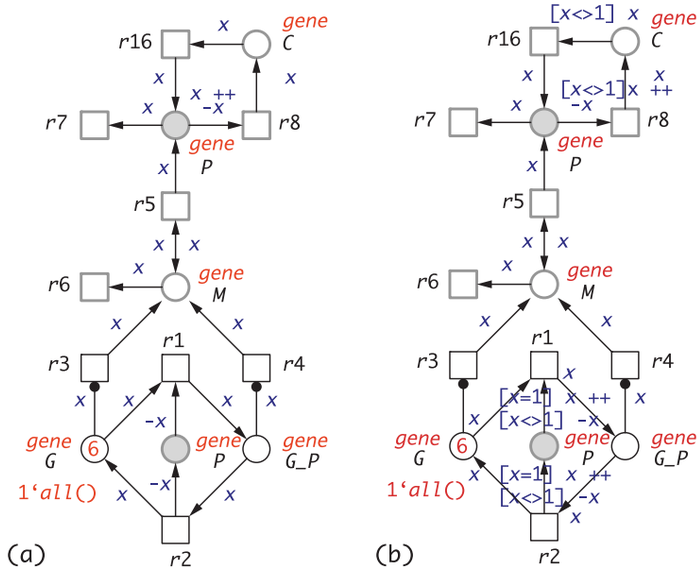


FIGURE S7.49 $\mathcal{HPN}^{\mathcal{C}}$ for linear and circular configuration of regulatory proteins: (a) row and (b) circle of proteins with positive effect on their upstream neighbor and negative effect on their downstream neighbor.

1. Create a new constant N of type *int* with, for example, value 6.
2. Edit the color set *Gene* by changing its type to *int* with colors $1 - N$.
3. Assign the color set *Gene* to the place C.
4. Change all predicates of the firing rates for $x = A$ to *true*.
5. Delete the firing rates with the predicate $x = R$.
6. Divide the constants in the firing rates for the predicate *true* by $[val(x)]$ (e.g., $MassAction(k/[val(x)])$).
7. Change the arc expression of $r8 \rightarrow C$ and $r16 \rightarrow P$ to x.
   (a) *Row*:
      8.1. Change the arc expression of $C \rightarrow r16$ to x.
      8.2. Change the arc expression of $P \rightarrow r8$ to $x + + - x$.

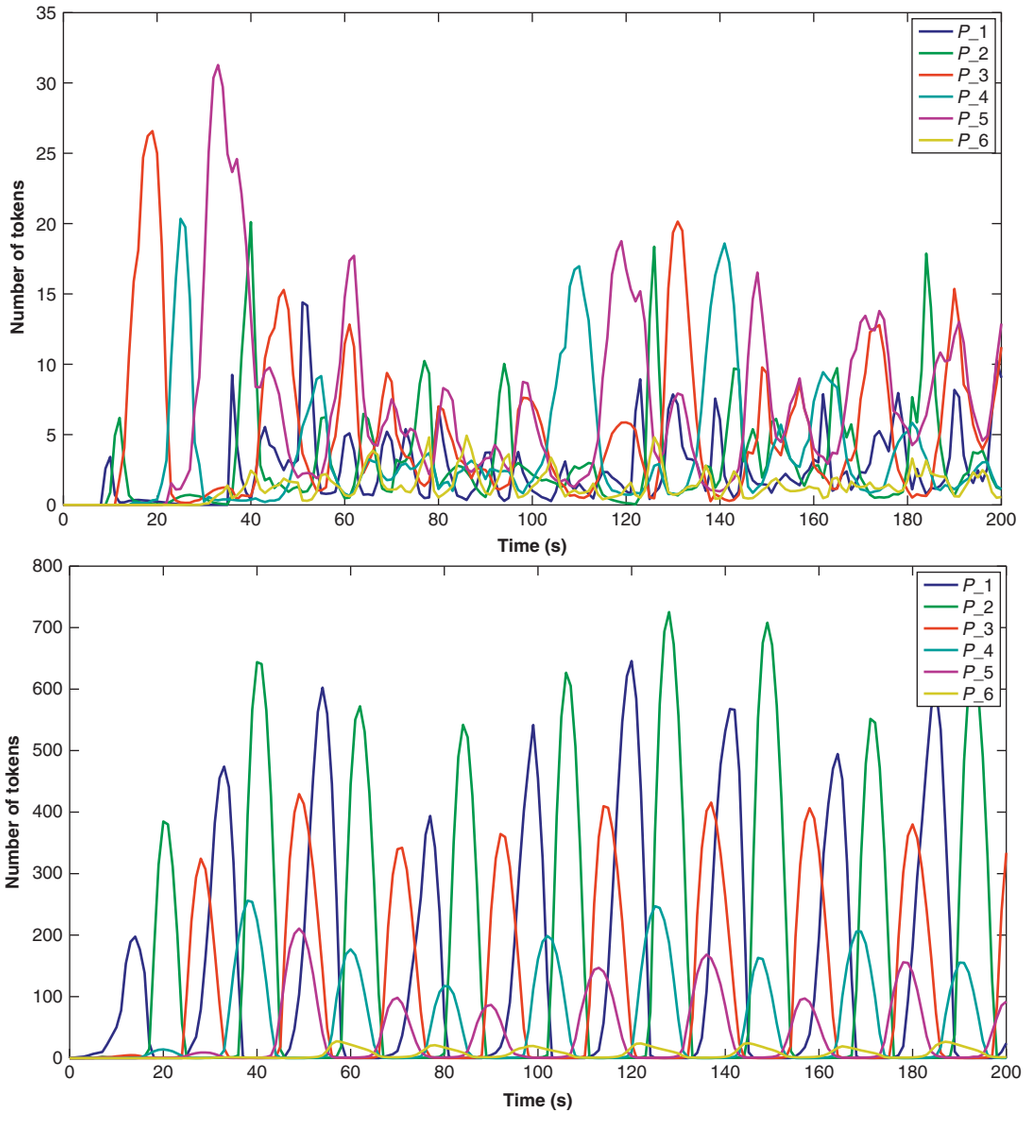


FIGURE S7.50 Hybrid simulation of linear and circular configuration of regulatory proteins. Hybrid simulation results with static synchronization of (a) a row and (b) a circle of proteins with positive effect on their upstream neighbor and negative effect on their downstream neighbor.

8.3. Change the arc expression of $P \rightarrow r1$ to $-x$.
8.4. Change the arc expression of $r2 \rightarrow P$ to $-x$.

a. *Circle*:
8.1. Change the arc expression of $C \rightarrow r16$ to $[x <> 1]x$.
8.2. Change the arc expression of $P \rightarrow r8$ to $[x <> 1](x + + - x)$.
8.3. Change the arc expression of $P \rightarrow r1$ to $[x = 1](1`x) + +[x <> 1](-x)$.
8.4. Change the arc expression of $r2 \rightarrow P$ to $[x = 1](1`x) + +[x <> 1](-x)$.

Compare Figure S7.49 for the structure of the colored Petri nets, and Figure S7.50 for hybrid simulation results with static synchronization. □

ACKNOWLEDGMENTS

We would like to thank Mostafa Herajy, Fei Liu, Christian Rohr, and Martin Schwarick for their contributions in developing and supporting the use of Snoopy, Marcie, and Charlie; and Robin Donaldson and David Gilbert for developing and supporting the use of MC2. We appreciate countless productive discussions with all of them.

This work has been partly supported by the Germany Federal Ministry of Education and Research (FKZ0316177E).

REFERENCES

[1] Blätke MA, Heiner M, Marwan W. Tutorial—Petri nets in systems biology. Tech. rep. Otto von Guericke University and Magdeburg, Centre for Systems Biology; 2011.
[2] Murata T. Petri nets: properties, analysis and applications. Proc IEEE 1989;77:541-80.
[3] Heiner M, Gilbert D, Donaldson R. Petri nets for systems and synthetic biology; 5016 of LNCS. Berlin: Springer; 2008. p. 215-64.
[4] Heiner M. Understanding network behaviour by structured representations of transition invariants—a Petri net perspective on systems and synthetic biology. Natural computing series. Heidelberg: Springer; 2009. p. 367-89.
[5] Schwarick M, Rohr C, Heiner M. MARCIE manual—an analysis tool for extended stochastic Petri nets. Tech. Rep. 03-14. BTU Cottbus-Senftenberg, Dep. of CS; 2014.
[6] Wilkinson DJ. Stochastic modelling for system biology. 1st ed. New York: CRC Press; 2006.
[7] Soliman S, Heiner M. A unique transformation from ordinary differential equations to reaction networks. PLoS ONE 2010;5:e14284.
[8] Shinar G, Feinberg M. Structural sources of robustness in biochemical reaction networks. Science 2010;327:1389-91.
[9] Brightman FA, Fell DA. Differential feedback regulation of the MAPK cascade underlies the quantitative differences in EGF and NGF signalling in PC12 cells. FEBS Lett 2000;482:169-74.
[10] Heiner M, Gilbert D. How might Petri nets enhance your systems biology toolkit; 6709 of LNCS. Berlin: Springer; 2011. p. 17-37.
[11] Silva M, Recalde L. Petri nets and integrality relaxations: a view of continuous Petri net models. IEEE Trans Syst Man Cybern Part C Appl Rev 2002;32:314-27.
[12] Angeli D. Boundedness analysis for open chemical reaction networks with mass-action kinetics. Nat Computing 2011;10:751-74.
[13] Heiner M, Mahulea C, Silva M. On the Importance of the Deadlock Trap Property for Mono-tonic Liveness in *Int. Workshop on Biological Processes & Petri Nets (BioPPN), satellite event of Petri Nets 2010*:39-54 2010.