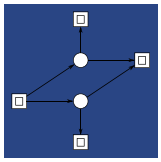

Brandenburg
University of Technology
Cottbus

The Manual for $\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$
- DRAFT -

Fei Liu and Monika Heiner

Data Structures and Software Dependability
Department of Computer Science
Brandenburg University of Technology at Cottbus
http://www-dssz.informatik.tu-cottbus.de/DSSZ
Please direct any comments for this manual to
snoopy@informatik.tu-cottbus.de.

July 29, 2011

Contents

Chapter 1

Introduction

Petri nets provide a formal and clear representation of systems based on their firm mathematical foundation for the analysis of system properties. However, standard Petri nets do not scale. So attempts to simulate systems by standard Petri nets have been mainly restricted so far to relatively small models. They tend to grow quickly for modeling complex systems, which makes it more difficult to manage and understand the nets, thus increasing the risk of modeling errors. Two known modeling concepts improving the situation are hierarchy and color. Hierarchical structuring has been discussed a lot, while the color has gained little attention so far. Thus, we investigate how to apply colored Petri nets to modeling and analyzing biological systems. To do so, we not only provide compact and readable representations of complex systems, but also do not lose the analysis capabilities of standard Petri nets, which can still be supported by automatic unfolding. Moreover, another attractive advantage of colored Petri nets for a modeler is that they provide the possibility to easily increase the size of a model consisting of many similar subnets just by adding colors.

In Snoopy, we have implemented $\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$ prototypes for editing, and animating/simulating colored qualitative Petri nets (QPN^{C}), colored stochastic Petri nets ($\mathcal{SPN}^{\mathcal{C}}$) and colored continuous Petri nets ($\mathcal{CPN}^{\mathcal{C}}$). In this manual, we will give relevant materials for understanding, constructing, simulating and analyzing colored qualitative/stochastic/continuous Petri nets, so that the user will have no difficulties in using colored Petri nets. In this manual, we concentrate on color-specific features. For a general introduction into Snoopy, see [HRR+08] and [RMH10].

1.1 Colored Petri nets

Colored Petri nets [GL79], [GL81], [Jen81], combine Petri nets with capabilities of programming languages to describe data types and operations, thus providing a flexible way to create compact and parameterizable models. In colored Petri nets, tokens are distinguished by the "color", rather than having only the "black" one. Additionally, arc expressions, an extended version of arc weights, specify which tokens can flow over the arcs, and guards that are in fact Boolean expressions define additional constraints on the enabling of transitions [JKW07].

For example, Figure 1.1 gives a colored Petri net, modeling dinning philoso-

phers. Around a round table sit some philosophers. Between each pair of philosophers there is one folk on the table. These philosophers either think or eat. In order to eat, they have to take the following steps: (1) take the left folk, (2) take the right folk and then start eating, (3) put the right folk back, and (4) put the left folk back. In the colored Petri net model, to change the number of philosophers means to change the number of colors in the net.

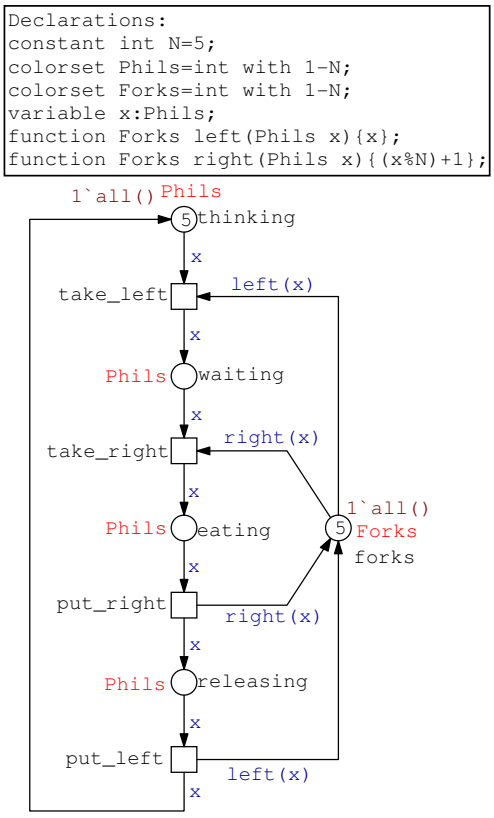

Figure 1.1: A colored Petri net, modeling dinning philosophers. All the declarations have been given on the top side (see Section 3 for how to read them). *all*() is a marking specification function, which means that all the colors in one color set are set to the same coefficient (here it is 1).

In our implementation, $\mathcal{QPN}^{\mathcal{C}}$ is a colored extension of extended qualitative place/transition net (extended by different kinds of arcs, e.g. inhibitor arc, read arc, reset arc and equal arc [HRR+08]), $\mathcal{SPN}^{\mathcal{C}}$ is a colored extension of biochemically interpreted stochastic Petri nets introduced in [GHL07] and extended in [HLG+09] and $\mathcal{CPN}^{\mathcal{C}}$ is a colored extension of continuous Petri nets introduced in [GHL07].

1.2 Some notes

In Snoopy, we provide a similar editing environment for $\mathcal{QPN}^{\mathcal{C}}$, $\mathcal{SPN}^{\mathcal{C}}$ and $\mathcal{CPN}^{\mathcal{C}}$; therefore the following descriptions will equally apply to $\mathcal{QPN}^{\mathcal{C}}$, $\mathcal{SPN}^{\mathcal{C}}$ and $\mathcal{CPN}^{\mathcal{C}}$, except those concerning animation, simulation and analysis, but all these differences will be noted clearly.

1.3 Features - overview

Before exploring all features in detail in the following chapters, we give a brief overview for the expected features here:

1.3.1 Features for modeling

- Drawing of the Petri net graph as usual.
- Rich data types for color set definition (See Section 3.1.1.):
  - Simple types: dot, int, string, bool, enum, index,
  - Compound types: product, union.
- User-defined functions.
- Concise specification of initial marking for larger color sets (See Section 2.4.1.).
- Rate function definition for each transition instance (only for $\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$) (See Section 2.4.2).
- Several extended arc types, such as inhibitor arc, read arc (often also called test arcs), equal arc, reset arc, and modifier arc, which are popular add-ons enhancing modeling comfort [HRR+08] (See Section 2.4.3).
- Several special transitions. Snoopy supports stochastic transitions with freestyle rate functions as well as three deterministically timed transition types: immediate firing, deterministic firing delay, and scheduled firing (see [HLG+09] for details.).
- Automatically colorizing some special subnets:
  - Colorizing any selected subnet,
  - Colorizing twin nets,
  - Colorizing T-invariants/master nets.

1.3.2 Features for animation (for $\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}$)

- The user can run animation automatically or control the animation manually:
  - Automatic animation,
  - Single-step animation by manually choosing a binding.

1.3.3 Features for simulation (for $\mathcal{SPN}^C/\mathcal{CPN}^C$)

- Simulation is done on an automatically unfolded Petri net.
- Show or export simulation results for colored or uncolored places/transitions separately or together.
- Several simulation algorithms to simulate $\mathcal{SPN}^C$, including the Gielespie stochastic simulation algorithm (SSA) [Gil77].
- Several simulation algorithms to simulate $\mathcal{CPN}^C$, including the Euler algorithm, Runge-Kutta algorithm etc.

1.3.4 Other features

- Highlighting the markings, color sets, guards, and expressions.
- $\mathcal{QPN}^C$, $\mathcal{SPN}^C$ and $\mathcal{CPN}^C$ are exported to different net formalisms within Snoopy, see Figure 1.2 (See Chapter 5 for details).
- Export $\mathcal{QPN}^C$ and $\mathcal{SPN}^C$ to APNN.
- Export/import beyond Snoopy, e.g., export to CPN tools(See Chapter 5 for details).

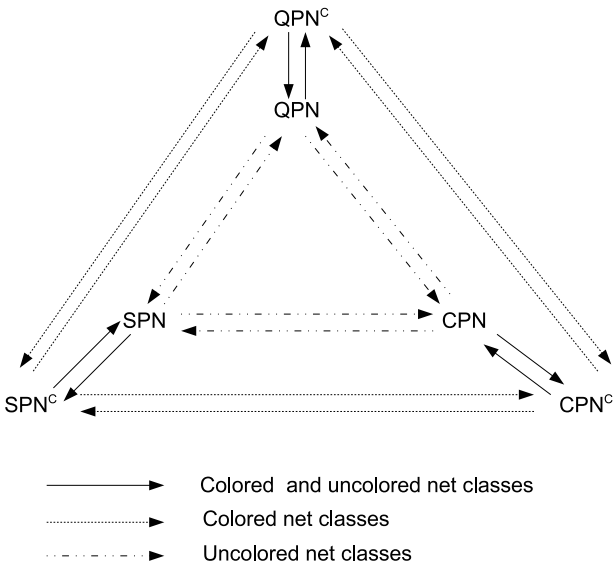

Figure 1.2: Export relationships among different net formalisms.

Chapter 2

Modeling

In this chapter, we will first demonstrate how to construct a colored Petri net ($\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$) and consider several key modeling problems afterwards.

2.1 General modeling procedure - an introductory example

This section will present a general step-by-step procedure of how to construct a $\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$ on the basis of a standard Petri net. A simple example will be used for the illustration of the procedure.

2.1.1 Transform a standard Petri net into a colored Petri net

One possibility to construct a colored Petri net is the transformation of an existing standard Petri net into a $\mathcal{QPN}^{\mathcal{C}}/\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$. The following sections will concentrate on $\mathcal{SPN}^{\mathcal{C}}$, but all steps can be applied to $\mathcal{QPN}^{\mathcal{C}}$ and $\mathcal{CPN}^{\mathcal{C}}$. We start with the following steps:

- Open a standard $\mathcal{SPN}$ (in our example "Copynet.spstochpn", see Figure 2.1) that should be transformed into a colored $\mathcal{SPN}^{\mathcal{C}}$.

- Go to the menu bar, select *File/Export* and choose “Export to colored stochastic Petri net" (see Figure 2.2). Define the path where you want to save the transformed Petri net. All Petri net elements (places, transitions, arcs) and their properties (markings, rate functions, arc weights) will we used for the construction of the corresponding colored Petri net.

2.1.2 Define similar subnets in the Petri net

We now need to subdivide the Petri net and fold it. We proceed as follows:

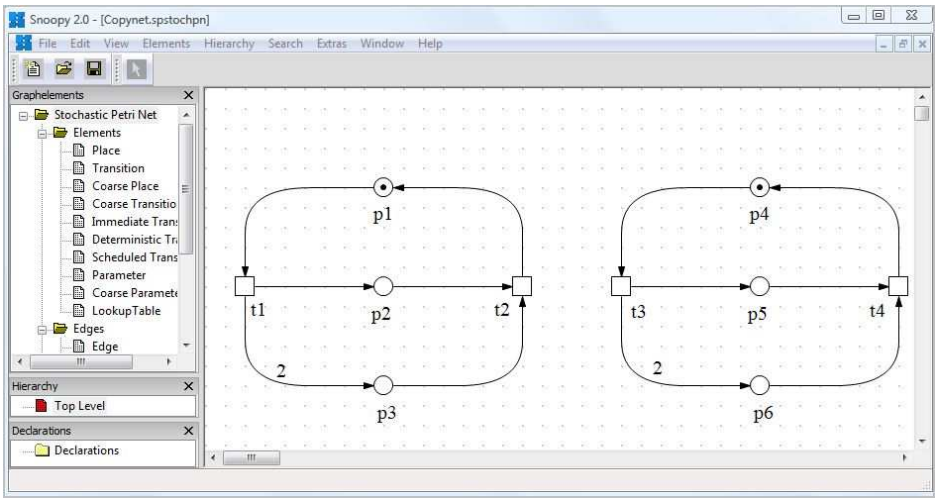

Figure 2.1: Open a stochastic Petri net.

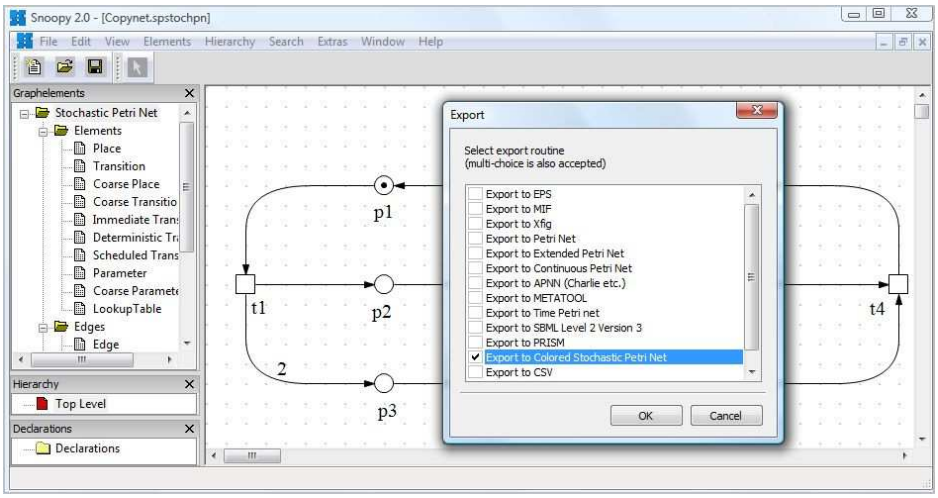

Figure 2.2: Export to colored stochastic Petri net.

- Open the transformed Petri net (shown in Figure 2.3). Please note that the transformed Petri net is now opened in the QPN^C/SPN^C environment. The transformation of the Petri net can be recognized by the assigned default color set Dot to all places of the original Petri net.

- Define similar subnets contained in the Petri nets. The Petri net shown in Figure 2.3 can obviously be divided into two subnets. Therefore, the color set that we will assign to the Petri net consists of two colors. For example: colorset *Copy = int with 1-2*. See Section 3.1.1 for how to define color sets.

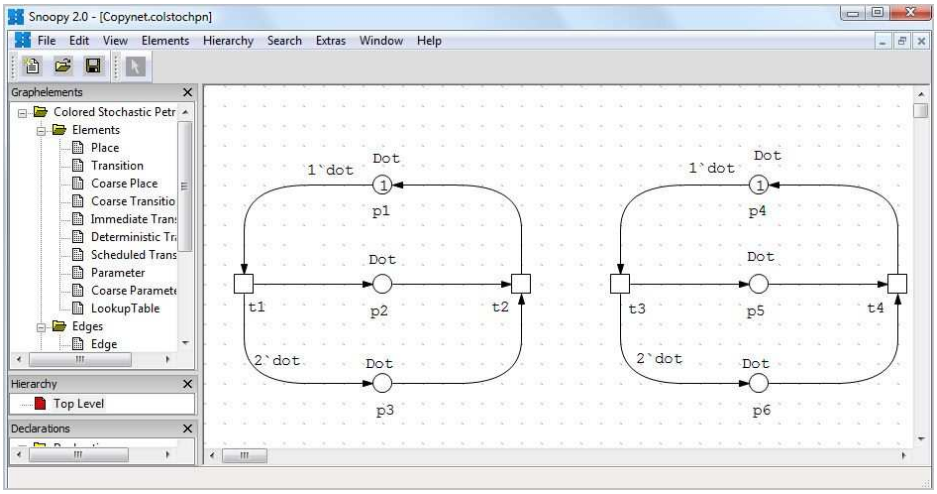

Figure 2.3: The transformed colored Petri net.

2.1.3 Define declarations

We have to declare and define the color sets, variables, constants and functions that he wants to apply to his SPN^C model. In the first step we define the color set according to the following procedure (see also section 5.1.1 for more information about color sets):

- Click on the tab "Colorsets" in declarations menu (left sidebar) and the color set definition dialogue will appear (shown in Figure 2.4).

- Define name, type (choose one in the drop down list) and colors of your color set.

- Check the syntax to proof your expressions.

For our running examples we will define the color set named "Copy" of the type integer (int) with the colors 1 and 2 (see Figure 2.4)

In the next step we define the variables (shown in Figure 2.5) that we want to use (see also section 5.1.2 for more information about variables). The procedure is analogous to the definition of the color set.

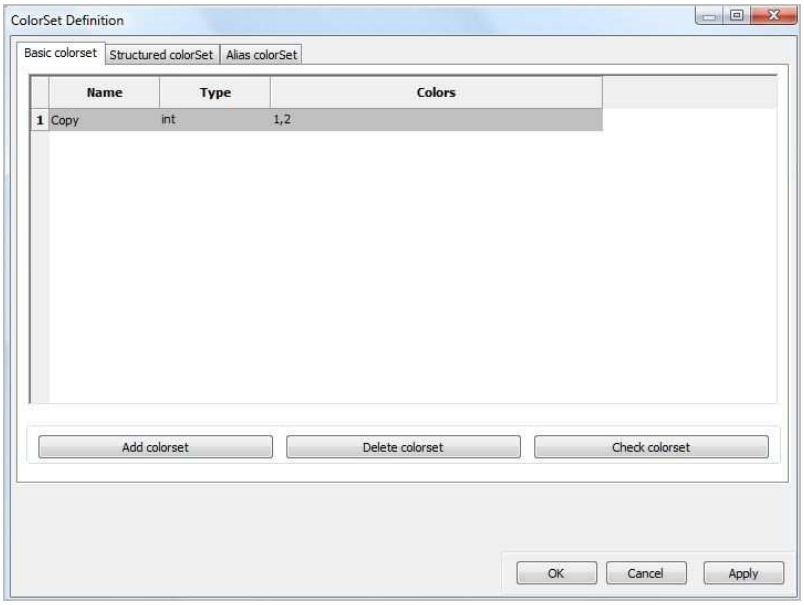

Figure 2.4: Define color sets.

In our running examples we define the variable "x", whose color set is "Copy" that can be chosen in the drop down list.

If you want to add any functions and constants, proceed according to the mentioned points (for more information about the declaration of functions and constants see section 5.1.3 and 5.1.4)

Following the same procedure to declare constants and functions.

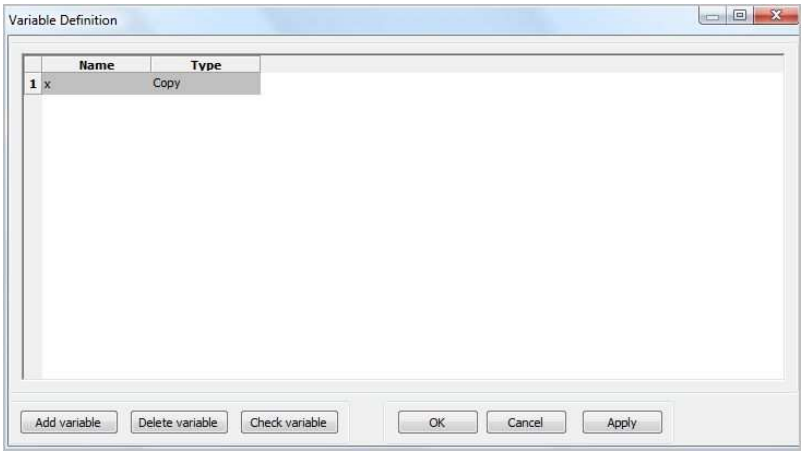

Figure 2.5: Declare variables.

2.1.4 Assign color sets to places and define initial markings

Now we need do apply the defined color set to the places of the colored Petri net.

- Open the "Edit Properties dialog" of a certain place.
- In the General tab, specify the name of a place.
- In the Marking tab, specify the color set in the "Colorset" box and edit the initial marking in the "MarkingList" (see Figure 2.6). You can always check the defined color sets with a click open the button "Colorset". If you want to apply the same marking for every color of this place use the function "all()", which means that all the colors in this color set are set to the same coefficient (here it is 1). (See Section 2.4.1 for more details on how to define initial markings.)

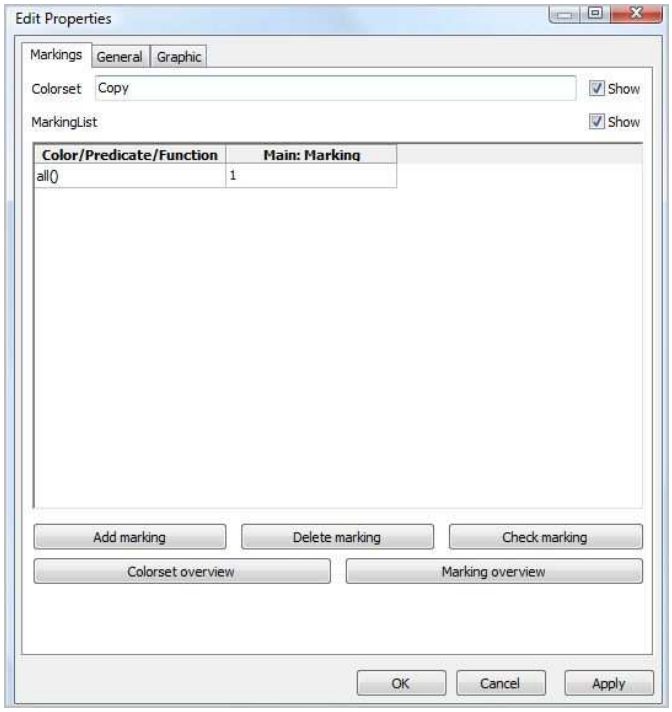

Figure 2.6: Specify initial marking.

It is also possible to edit a group of places and set their color set and marking at once. Just selected the places you want to edit and proceed like above.

- Select a group of places.
- Click *Edit/Edit selected elements,* and then a dialogue to specify the properties appear, e.g. see Figure 2.7 .

- In the Marking tab, specify the color set in the "Colorset" box and edit the initial marking in the "MarkingList".

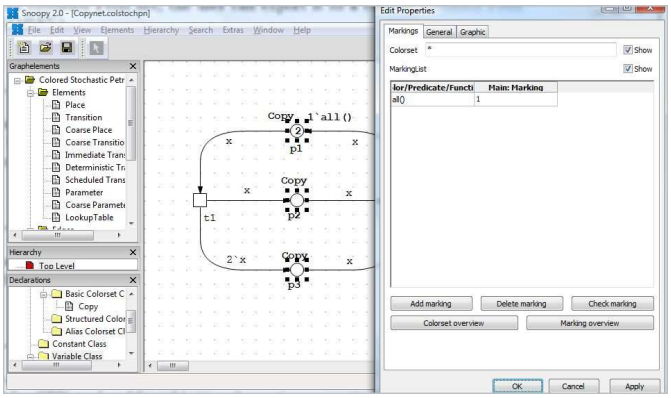

Figure 2.7: Specify inital markings for a group of places.

2.1.5 Define arc expressions

In the next step, we define the expression of every arc. (See Section for how to write arc expressions.)

- Open the "Edit Properties dialog" of a certain arc.
- In the Expression tab, write the expression, which can be aided by the expression assistant (Figure 2.8). Please note that this field should not be empty.

In our example, we will use the arc weight separated with "`" from the variable x.

You can also edit multiple arcs by selecting a group of arcs and edit them like above.

2.1.6 Define guards for transitions

The guards of a transition can be edited as follows, if they are needed (see also section Section 3.2.3).

- Open the "Eidt Properties dialog" of a transition.
- Write the guard expression in the "Guard" tab (see Figure 2.9). You have also the possibility to use the guard assistant to define the guard functions.

Again, you can edit multiple transitions by selecting a group of transitions.

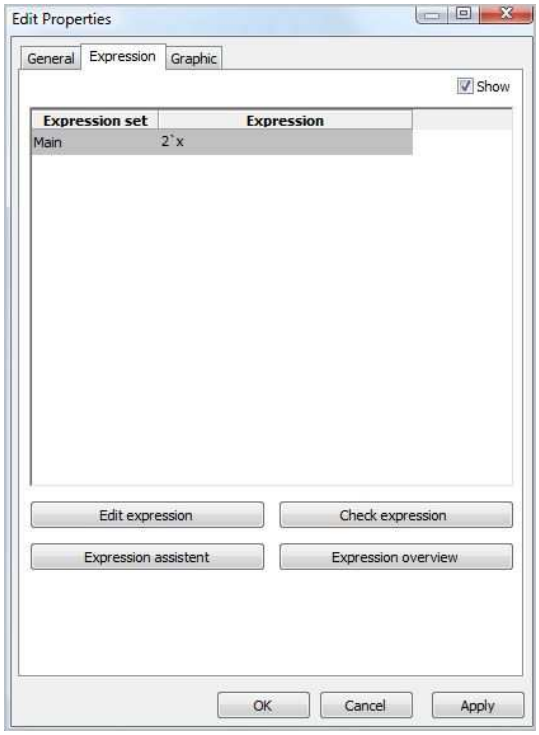

Figure 2.8: Write arc expressions.

2.1.7 Define rate functions for transitions (for $\mathcal{SPN}^{\mathcal{C}}/\mathcal{CPN}^{\mathcal{C}}$)

You can also edit the rate functions for transitions if they are needed, which follows the following procedure. You have also the possibility to use the rate function assistant to define the rate functions.

- Open the "Edit Properties dialog" of a transition.
- Write the rate function expression in the "Function" tab (see Figure 2.10).

See Section 2.4.2 for how to write rate functions. Rate functions are only available for $\mathcal{SPN}^{\mathcal{C}}$ and $\mathcal{CPN}^{\mathcal{C}}$.

For every mentioned step above exist a check of the syntax. With the help of the check function you can find and avoid mistakes. You can find this function in each dialogue.

After applying all the steps to our running example, we obtain the following colored Petri net model (see Figure 2.11). We don't need the right subnetwork anymore, because we established this copy by assigning a color set consisting of two colors to the left one. This Petri net is equivalent to the original Petri net of Figure 2.1. With the help of high-level (colored) Petri nets we can easily increase the number of copies by changing the declaration of the color set instead of creating multiple graphical copies of one and the same subnet.

Figure 2.9: Write guards.

2.2 Contructing colored Petri nets

Colored Petri nets allow a more compact and parametric representation of a system by folding similar subnets. So it is possible to represent very concisely systems that would have required a huge uncoloured net. In this section, we will demonstrate how to construct basci colored Petri net components, so that we can build the whole model based on these components.

2.2.1 Basic colored Petri net components

The key step in the design of a colored Petri net is to construct basic colored Petri net units, through which we can obtain the whole colored Petri net model step by step. This process is also called folding. In the following we will introduce some folding ways to construct basic colored Petri net components, which are illustrated in Figure 2.12.

Figure 2.12 (a) shows the folding of two isolated subnets with the same structure. For this simple case, we only need to assign the color set "CS" to the place. We write the arc expression as x, where x is a variable of the type "CS". Thus, we get a basic colored Petri net component, illustrated on the right hand of Figure 2.12 (a).

In **Figure 2.12 (b)**, the net to be folded is extended by two extra arcs from p2 (p1) to t1 (t2), respectively. To fold it, we use the same color set, and just modify the arc expression to $x + +(+x)$, where the "+" in the $(+x)$ is the successor operator, which returns the successor of x in an ordered finite color

Figure 2.10: Write rate functions.

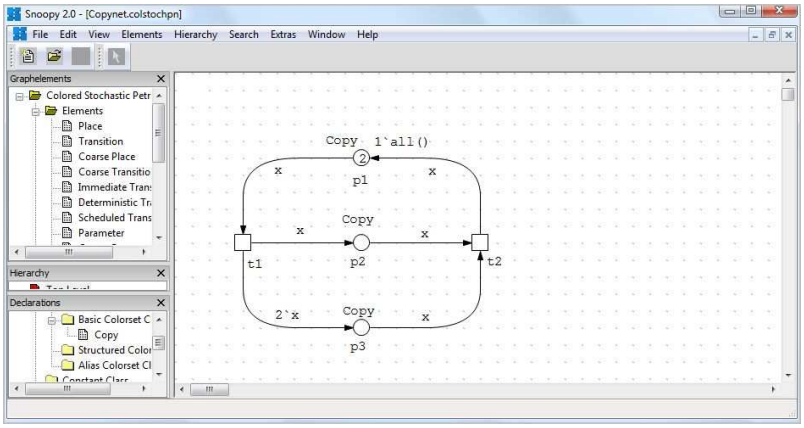

Figure 2.11: The colored Petri net model.

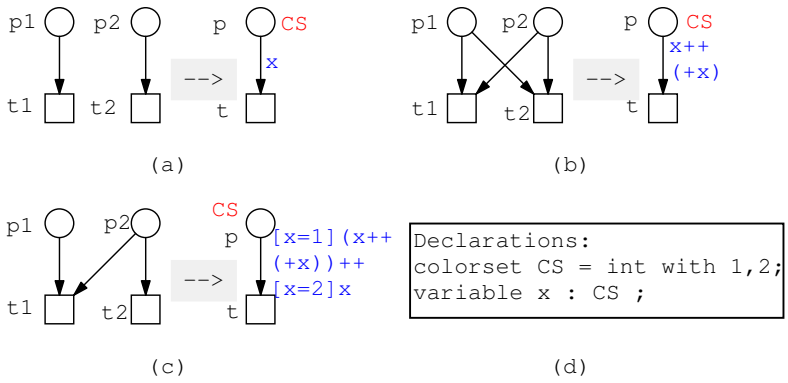

Figure 2.12: Basic colored Petri net components. For all these three cases, we define the color set as "CS" with two integer colors: 1 and 2. We use color "1" to represent the subnet containing p1 and t1, and color "2" to represent the subnet containing p2 and t2.

set. If x is the last color, then it returns the first color. The "++" is the multiset addition operator.

In **Figure 2.12 (c)**, the net to be folded gets one extra arc from p2 to t1. To fold it, we use the same color set, and just modify the arc expression to $[x = 1](x + +(+x)) + +[x = 2]x$, meaning: if $x = 1$, then there are two arcs connecting p with t, while if $x = 2$, then there is only one arc connecting p with t.

In summary, the following rules apply when folding two similar nets to a colored Petri net. If the two subnets share the same structure, we just have to define a color set and set arc expressions without predicates. If the subnets are similar, but do not have the same structure, we may need to use guards or arc expressions with predicates. However, in either case, if we want to continue to add other similar nets, what we should do is usually to add new colors, and slightly change arc expressions or guards. Using these basic colored Petri net components, we can construct the whole colored Petri net model step by step.

2.2.2 Modeling branch and conflict reactions

In this section, we demonstrate how to construct colored models for two special situations: a branching reaction (One reaction produces several products from reactants.) and conflicting reactions (Several reactions use the same reactants and produce their products independently or concurrently.) Figure 2.13 and Figure 2.14 illustrate how to model these two situations, respectively.

Figure 2.13 shows how to fold a branching reaction into a colored component. For this case, we define two color sets: *Dot* with one color *dot*, and CS with two colors b and c. We then assign the color set "Dot" to the place A, and CS to the place P. We define the expression *dot* for the arc from A to T and $b + +c$ for the arc from T to P, which means that when T fires two tokens with colors b and c will be added to P. Please note that the "++" is the multiset addition operator.

Figure 2.14 shows how to fold conflicting reactions into a colored component. For this case, we use the same color sets. We assign the color set "Dot" to the place A, and CS to the place P. We define the expression *dot* for the arc from A to T and x for the arc from T to P, where x is a variable of the type "CS".

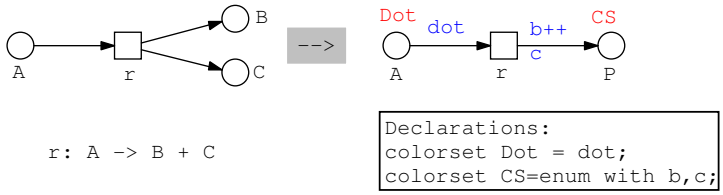

Figure 2.13: Petri net representation (on the left hand) and colored Petri net representation (on the right hand) of a branching reaction with reactant A and products B and C.

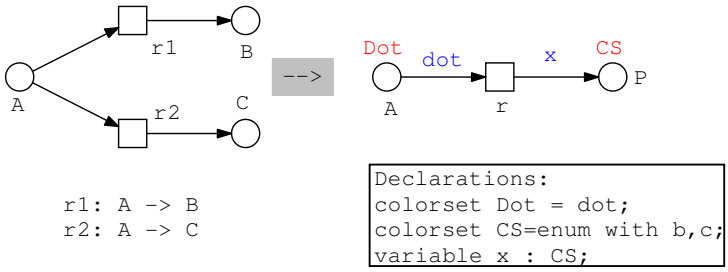

Figure 2.14: Petri net representation (on the left hand) and colored Petri net representation (on the right hand) of two conflicting reactions with reactant A producing B or C.

2.2.3 Modeling nets with logical nodes

In this section we will discuss how to deal with nets with logical nodes, illustrated in Figure 2.15 to Figure 2.19.

2.3 Automatic colorizing

Now, three ways are allowed to automatically colorize selected subnets.

2.3.1 Colorizing any subset

Go to the menu bar, select *Extras/Folding/Colorize* and then the user can colorize a selected subnet. During this process, the user can set a new color set

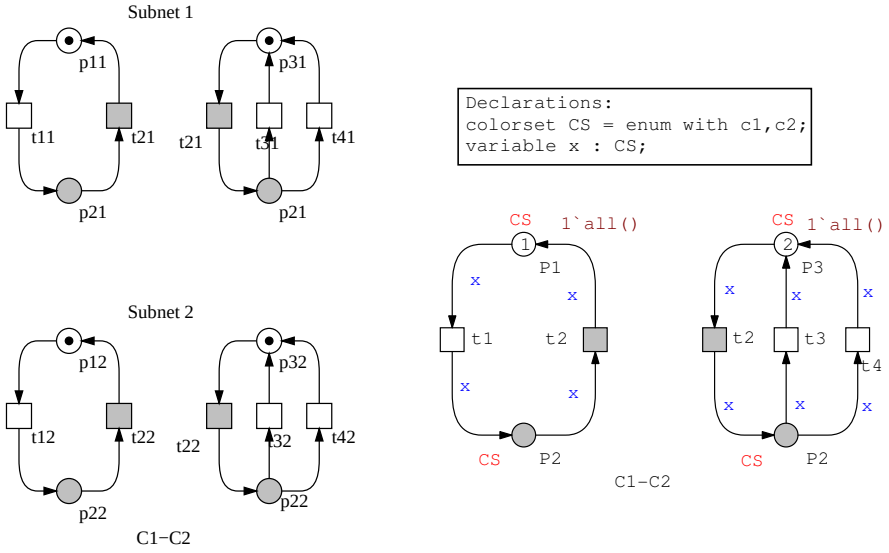

Figure 2.15: Case 1. In this case, we fold Subnet1 and Subnet2 (on the left hand) to a colored component (on the right hand). Each logical node has a unique copy in each subset.

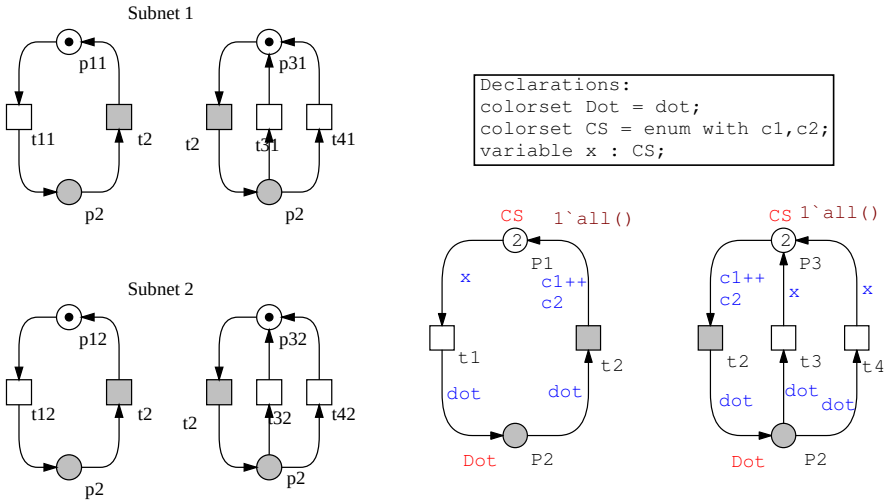

Figure 2.16: Case 2. In this case, either the transition $t2$ or place $p2$ only have one unique copy in both subnets.

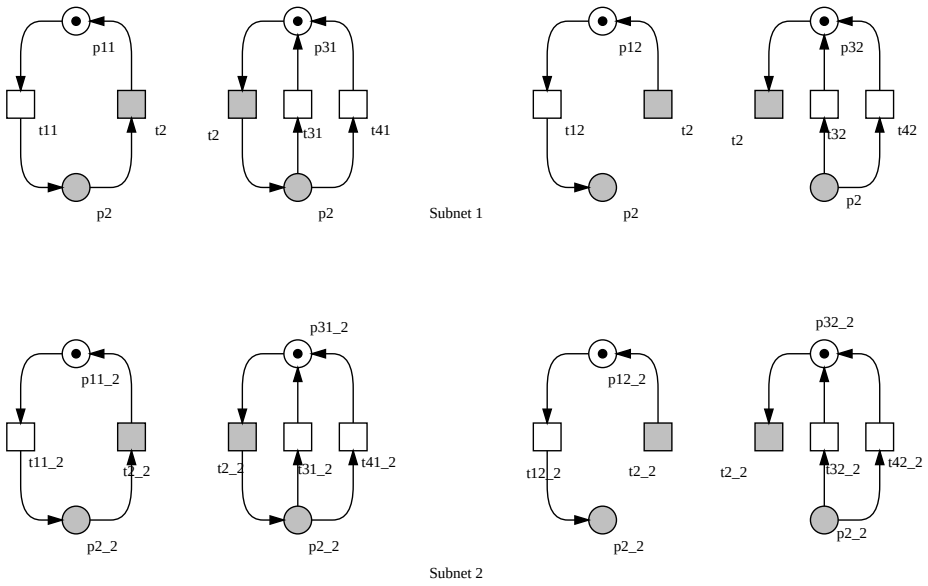

Figure 2.17: Case 3. Each logical node has a unique copy in each subset.

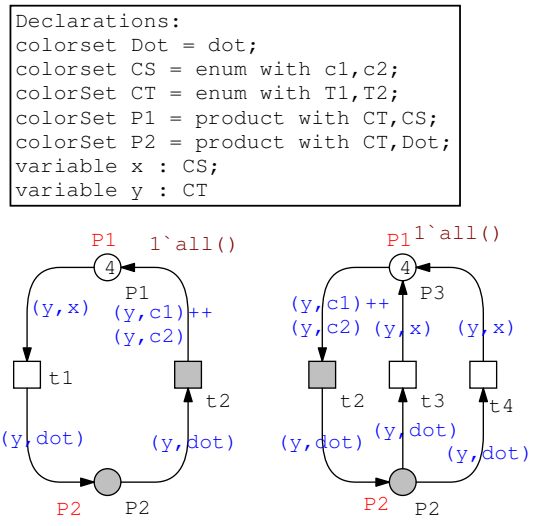

Figure 2.18: Colored Petri net model for Figure 2.17. Each logical node has a unique copy in each subset. Each subnet has the same structure and uses the same color set CS.

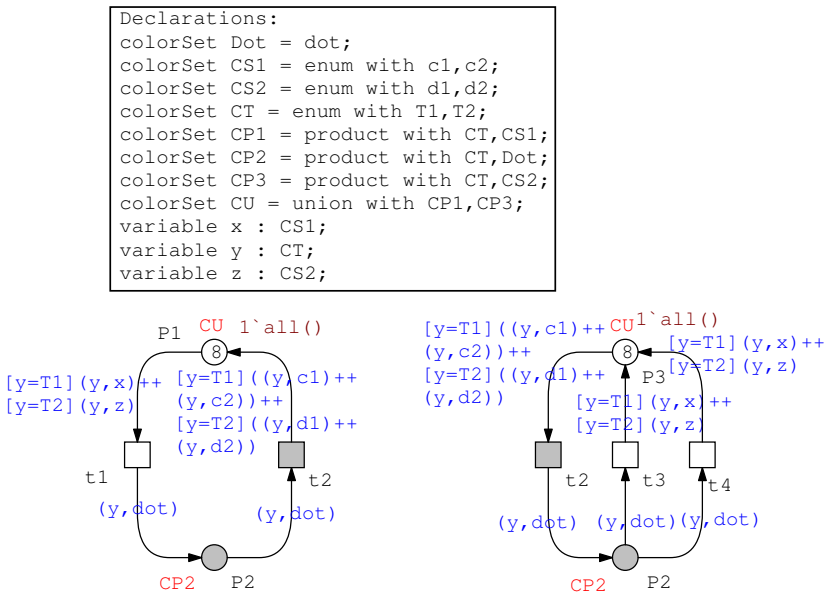

Figure 2.19: Colored Petri net model for Figure 2.17. Each logical node has a unique copy in each subset. Each subnet allows a different structure and uses a different color set. Here Subnet 1 uses the color set $CS1$, and Subnet 2 uses the color set $CS2$.

(for places) and variable name (for edges). After this is done, all places have the same color set and all edges the same expression.

2.3.2 Colorizing twin nets

Go to the menu bar, select *Extras/Folding/Generate twin nets* and then the user can create twin nets for a given net.

2.3.3 Modeling T-invariants

Go to the menu bar, select *Extras/Folding/Generate master nets* and then the user can create a color Petri net model for the given T-invariant file. The user then can demonstrate T-invariants on this colored net.

2.4 Some other key modeling problems

2.4.1 Specifying initial markings

We provide several ways for specifying initial markings:

- Specifying colors and their corresponding tokens as usual,
- Specifying a set of colors with the same number of tokens,

Table 2.1: Specification of initial markings. *Colorset* CS = *int with* $1-100$.

Color/Predicate/Function	marking
1	1
4,5,7	2
$x > 10$	3
all()	4

- Using a predicate to choose a set of colors and then specifying the same number of tokens,
- Using the *all*() function to specify for all colors a specified number of tokens.

Table 2.1 gives some examples for specifying initial markings.

2.4.2 Specifying rate functions

As there are four kinds of transitions (stochastic, immediate, deterministic and scheduled), we have to choose a suitable kind. Then we have to define the rate functions for the stochastic transitions, the weights for the immediate transition, the delays for the deterministic transitions, and the periodic for the scheduled transitions. But their specification has a similar procedure.

We start with the specification of predicates of rate functions. When writing predicates, there are some notes you should notice:

- For a same binding, only one predicate is allowed to be evaluated to true in the situation of more than one predicates. For example, in Table 2.2, we have two predicates, $x = 1$ and $x = 2$. For each binding, only one of these is evaluated to true. However, we are not allowed to write the predicates like this, $x = 1$ and $x \geq 1$, as these two predicates are evaluated to true for the binding $x = 1$.
- If the predicates of a transition do not cover all the instances of this transition, then the rate functions of these instances that are not covered are set to 0. For example, if we only use a predicate $x = 1$, this predicate will not cover the transition instance when x equals 2. During the syntax checking, there is a warning in the log window like this, "15:01:12: Notice: Transition: t1: predicates are not fully covered, where the rates are set to 0".

There are three ways for the specification of rate functions: at the colored level or at the instance level (Here we call each unfolded transition corresponding to a colored transition a transition instance of this colored transition.) or a combination of both of them. For any way, we should first use predicates to choose a or a set of transition instances and then specify rate functions.

Table 2.2: Specifying rate functions.

#	Predicate	Rate function
1	*true*	$P2 * P3$
2	$x = 1$	$P2 * P3$
	$x = 2$	$5 * P2 * P3$
3	*true*	$P1[1] * P1[2]$
4	*true*	$P1[1] * P1[2] * P2 * P3$
5	$x = 1$	$P1[1] * P1[2] * P2 * P3$
	$x = 2$	$5 * P1[1] * P1[2] * P2 * P3$

(1) Specifying rate functions at the colored level

We can specify rate functions by referencing names of colored places, just like specifying rate functions for stochastic Petri nets. For instance, in Figure 2.20 we can do it at the colored level like shown in the #1 and #2 of Table 2.2.

(2) Specifying rate functions at the instance level

We can also specify rate functions at the instance level. To do this, in a rate function, we reference a colored place, followed by [*color*/*variable*], which denotes the place instance by the specified "color" or "variable". For instance, in Figure 2.20 we can do it at the instance level as shown in the #3 of Table 2.2.

In addition, we can also combine the above ways to specify rate functions, like shown in the #4 and #5 of Table 2.2.

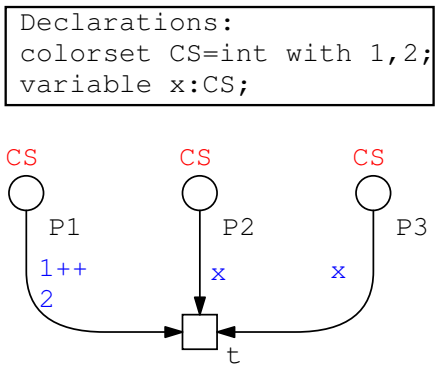

Figure 2.20: An example to demonstrate how to specify rate functions. The operator ++ in the arc expression 1++2 is the multiset addition operator.

2.4.3 Extended arc types

We support the following extended arc types, which are popular add-ons enhancing modeling comfort (see Figure 2.21 for graphical representation in Snoopy):

- inhibitor arc,
- read arc,
- equal arc,
- reset arc, and
- modifier arc.

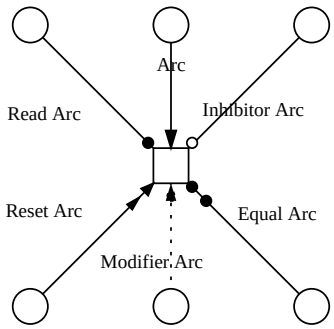

Figure 2.21: Special arcs in Snoopy.

Figure 2.22 gives an example for demonstrating the folding involving extended arcs, which contains two cases: 1) two special arcs are the same kind, and 2) two arcs are different kinds.

2.4.4 Consistency checks

In the rate function of a transition, only preplaces of this transition are allowed. However sometimes we may omit some preplaces in writing rate functions for different reasons. Therefore, we support to automatically check unused preplaces in rate functions, so that we can reexamine the rate funcitons. Consistency check is a part of syntax check. The principles we consider are as follows:

- If a rate function is constant, then we only check unused preplaces connected by modifier arcs,
- If a rate function contains places, then we check all unused preplaces.

The following is a consistency check result, which is taken from the Halo model.

- 11:08:35: Warning: The rate function for r31 has unused modifier places: SRI510

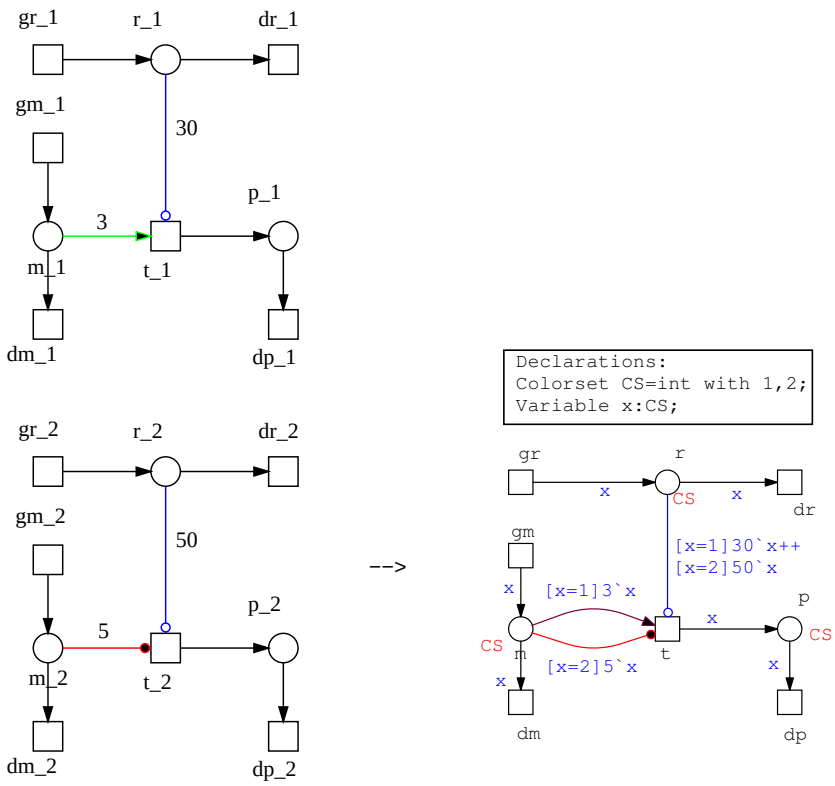

Figure 2.22: An example for demonstrating the folding involving extended arcs.

- 11:08:35: Warning: The rate function for r32 has unused modifier places: SRI510
- 11:08:35: Warning: The rate function for r36 has unused places: CheB
- 11:08:35: Warning: The rate function for r37 has unused places: CheB

Chapter 3

Annotation Language

In this chapter, we will describe the annotation language developed for colored Petri nets.

3.1 Declarations

3.1.1 Color sets

We provide two groups of data types to define color sets of colored Petri nets. The simple types can be directly used, but the compound ones must be based on defined color sets. The BNF form for the data type definition is given in Appendix A.2.

- Simple types: dot, int, string, bool, enum, index,
- Compound types: product, union.

Compared with CPN tools [CPN11], we do not support the list and record data types. The reason for not providing the record type is that the record type can be replaced by the product type. For the list type, the reason is that we only want to support finite color sets so as to get an unfolding Petri net from any color Petri net. In the following, we will describe each data type in detail.

(1) dot

We define a dot data type to declare a color set "Dot" with only one black color "dot".

(2) int

Integers are numerals without a decimal point. Here only non-negative integers are supported.

- Declaration Syntax:

  Integers seperated by "," or "-". Here some legal definitions:

  – 1,2,3

- 1-3
- 1,3,5-7
- 1-n

For example, "1,3,5-7" defines the color set that has the following colors: "1,3,5,6,7". We can also support a constant in the integer color set definition, for example, in the "1-n", n is a integer constant (See Section 3.1.4 for constant declarations.).

- Operations:
  - $i_1 + i_2$ addition
  - $i_1 - i_2$ subtraction
  - $i_1 * i_2$ multiplication
  - i_1 / i_2 division, quotient
  - i_1 % i_2 modulus, remainder

(3) string

Strings are specified by seqences of printable ASCII characters surrounded with double quotes.

- Declaration Syntax:

Strings separated by "," or "-". We also support regular expressions to define string, but they will be separated by "[ ]". Here some legal definitions:

- a, b, c
- a-c
- a, c, e-g
- $[a][e, f, g]$

For example, a, c, e-g defines the color set that has the following colors: a, c, e, f, g. $[a][e, f, g]$ defines the colors: ae, af, ag.

- Operations:
  - $s1 + s2$ concatenate the strings $s1$ and $s2$.

(4) bool

The boolean values are true and false.

- Declaration Syntax:

false, true.

- Operations:
  - ! b negation of the boolean value b,
  - $b1$ & $b2$ boolean conjunction, and,
  - $b1$ | $b2$ boolean disjunction, inclusive or.

(5) enum

Enumerated values are explicitly named as identifiers in the declaration.

- Declaration Syntax:

Strings separated by "," or "-". We also support regular expressions to define enum, but they will be separated by "[ ]". Here some legal definitions:

  - a, b, c
  - a-c
  - a, c, e-g
  - $[a][e, f, g]$

For example, a, c, e-g defines the color set that has the following colors: a, c, e, f, g. $[a][e, f, g]$ defines the colors: ae, af, ag.

The color set definition for enum is like that of string. The difference is that the initial character of an enum color is a letter or "_".

- Operations:

There are no standard operations.

(6) index

Indexed values are sequences of values composed of an identifier and an index-specifier

- Declaration Syntax:

index id with [intexp1 - intexp2]. For example, we can define an index color set as: *colorset Philosopher with index phil[1-5]*.

- Operations:

There are no standard operations.

(7) product

A product color set is a tuple of previously declared color sets.

- Declaration Syntax:

Defined color sets separated by ",". For example, we can define a product color set as: *colorset Philosopher with product H2O* $\times$ *Level*, where $H2O$ and *Level* are two previously defined color sets.

- Operations:

There are no standard operations.

(8) union

A union color set is a disjoint union of previously declared color sets.

- Declaration Syntax:

Defined color sets separated by ",". For example, we can define a union color set as: *colorset Salad with union Fruit, Dish*, where *Fruit* and *Dish* are two previously defined color sets.

- Operations:

There are no standard operations.

3.1.2 Subsets of color sets

We can also define subsets for a defined color set in the following two ways:

- Enumerate the colors that will appear in a subset, separated by ',',
- Using a logic expression (predicate) to select a group of colors, see Section 3.2.3 for how to define a predicate.

For example, suppose $Colorset\ CS = int\ with\ 1 - 10$, $Variable\ x : CS$ and then we can define a subset *CS_sub* for the color set CS using the logic expression $x <> 10$, which selects the colors, 1-9, for the subset *CS_sub*.

3.1.3 Variables

A variable is an identifier whose value can be changed during the execution of the model. They have the following characteristics:

- They are declared with a previously declared color set.
- They are bound to the variety of different values from their color set by the simulator as it attempts to determine if a transition is enabled.
- There can be multiple bindings simultaneously active on different transitions. These bindings can exist simultaneously because they have different scopes.
- They allow arc expressions with the ability to reference different values.

Variables can be used in the following situations (Suppose $Colorset\ CS = int\ with\ 1 - 10$; $Variable\ x : CS$):

- arc expressions, e.g., $x + 1$,
- guard, e.g., $x < 5$,
- marking predicate definition, e.g., $x < 6$,
- rate function predicate definition, e.g., $x < 7$.

3.1.4 Constants

A constant has a value and correponding data type or color set. For example, we can define a constant as follows: *constant* n = *int with* 5. Constants can be used in the arc expressions, guards, predicates and integer color set definition.

Constants can be used in the following cases (Suppose $Colorset\ CS$ = *int with* $1-10$; $Variable\ x : CS$; $Constant\ n : CS\ with\ 5$):

- arc expressions, e.g., $x + n$,
- guard, e.g., $x < n$,
- integer colorset definition, e.g., $x < n$,
- marking predicate definition, e.g., $x < n$,
- marking definition, e.g., we can set a color having a number of n,
- rate function predicate definition, e.g., $x < n$.

3.1.5 Functions

We can also define functions that are used in the whole net. A user-defined function contains the following components:

- Function name, which is an identifier,
- Parameter list, separated by ",",
- Function body, which is an expression, and
- Return type, which is the type of the return value.

When we write a function body, we can use all the defined constants and all the operators in Table 3.1. A function body should comply with the BNF forms in Appendix A.3. However, *please be careful when using the operator ++ and make sure that this will return only one single value or empty* as we at present do not support that the user-defined function returns more than one values (colors).

Specifically speaking, a user-defined function can be used in the following situations:

- expressions on arcs,
- guards on transitions,
- predicates in rate functions of transitions, and
- predicates in marking definitions of places.

In Figure 1.1, we use two user-defined functions. For example,

$$Forks\ Left(Phils\ x)\ \{\ x\ \}.$$

In this function, $Forks$ is the type of the return value, which is an integer color set. $Left$ is the function name. $Phils\ x$ defines the parameter of this function. x is the function body, which returns the left folk.

$$Forks\ Left(Phils\ x)\ \{\ (x\%N)+1\ \}.$$

This function returns the right fork. % is the modulus operator.

In Figure 6.5, we also use user-defined functions (See Table 6.1 for details.). For example, the function $Fun1$ is defined as follows:

$$P\ Fun1(HbO2\ x, Level\ y)\ \{\ [y=L]1`(x+1,y)++[y=H]1`(x,y)\ \}.$$

In this function, P is the type of the return value, which is a product color set. $Fun1$ is the function name. $HbO2\ x, Level\ y$ define two parameters of this function, where x is of the type $HbO2$ and y of $Level$. $[y=L]1`(x+1,y)++[y=H]1`(x,y)$ is the function body, which means when y equals L it will return one token with the color $(x+1,y)$ and when y equals H it will return one token with the color (x,y). See Section 3.2.2 for more details about how to read function bodies.

3.2 Expressions

3.2.1 Operators

We support the operators summarized in Table 3.1.

Table 3.1: Operators in the annotation language.

Priority	Operator	Executed operation
10	+	Successor, which returns the successor of the current color in an ordered finite color set. If the current color is the last color, then it returns the first color.
	−	Predecessor, which returns the predecessor of the current color in an ordered finite color set. If the current color is the first color, then it returns the last color.
	@	Index extracting, which returns the index of an index color.
	.	Extracting a component from a product color.
	!	Logical not.
9	$*, /, \%$	Arithmetic multiplicity, division, and modulus.
8	+	Arithmetic addition, or string concatenation.
	−	Arithmetic subtraction.
7	$<, <=$	Less than, or less than or equal to.
	$>, >=$	Greater than, or greater than or equal to.
6	$=, <>$	Equal, or unequal.
5	&	Logical and.
4	|	Logical or.
3	,	Used in a tuple expression.
2	`	Separating the coefficiency and the color.
1	++	Multiset addition, connecting two multiset expressions.

3.2.2 Arc expressions

Arc expressions can be defined according to the BNF forms illustrated in Appendix A.3. Arc expressions can use all the constants, variables and user-defined functions and all the operators in Table 3.1.

For example, in Figure 6.3 (see Table 6.1 for its declarations), we use three different expressions: *dot* e.g. on the arc from transiton t1 to place O2, x e.g. from t1 to HbO2L and $x + 1$ e.g. from HbO2L to t1. Among these, *dot* is a constant, x is a variable and $x + 1$ is an addition expression.

In Figure 6.4, we will see more complex expressions. For example, $[y = L]dot$ on the arc from place O2 to transition t1 means that if y equals L it will return a token with the color *dot*, otherwise an empty value. In fact, $y = L$ is a predicate of this expression.

3.2.3 Predicates/guards

Predicates/guards are in fact boolean expressions, which should be evaluated as boolean values. Guards are used for transitions, which decide which transition instances exist, while predicates are used in other situations. Predicates/guards can contain user-defined functions. Specifically speaking, we use the predicates in the following situations:

- Subset definition of color sets, where a predicate is used to select a group of colors to form a subset.
- Initial marking specification, where a predicate is used to select a group of colors.
- Rate function specification, where a predicate is used to select a group of transition instances.
- Arc expression specification, where a predicate is used to decide if the current arc is used or not.

For example, in Figure 6.3 (see Table 6.1 for its declarations), in the expression $[y = L]dot$, $y = L$ is a predicate of this expression, where when $y = L$ is evaluated to true, this expression will return one token *dot*, otherwise it will return empty. In addition, there is a guard $x <> 4$ e.g. on transitions t1, which means when this guard is evaluated to true, there exists a transition instance of t1.

Chapter 4

Animation, Simulation and Analysis

In this chapter, we will demonstrate how to animate/simulate/analyze $\mathcal{QPN}^{\mathcal{C}}$, $\mathcal{SPN}^{\mathcal{C}}$ and $\mathcal{CPN}^{\mathcal{C}}$.

4.1 Animation (for $\mathcal{QPN}^{\mathcal{C}}$ and $\mathcal{SPN}^{\mathcal{C}}$)

When the Petri net model is opened, then the user can click the *View|Start Anim-Mode* to prepare animation. Before opening the animation dialogue, the syntax will be checked automatically for this model. The user can choose automatic animation or a manual one, in which case the user can select a binding. In the following, we will in detail describe it. Figure 4.1 shows the animation interface.

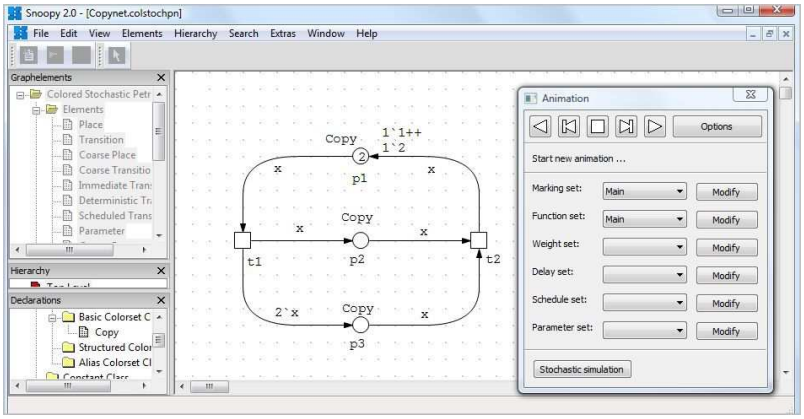

Figure 4.1: Animation interface.

4.1.1 Automatic animation

When the user clicks the *Play forward/Pause* button, the automatic animation will begin/pause. Figure 4.2 shows one animation snapshot.

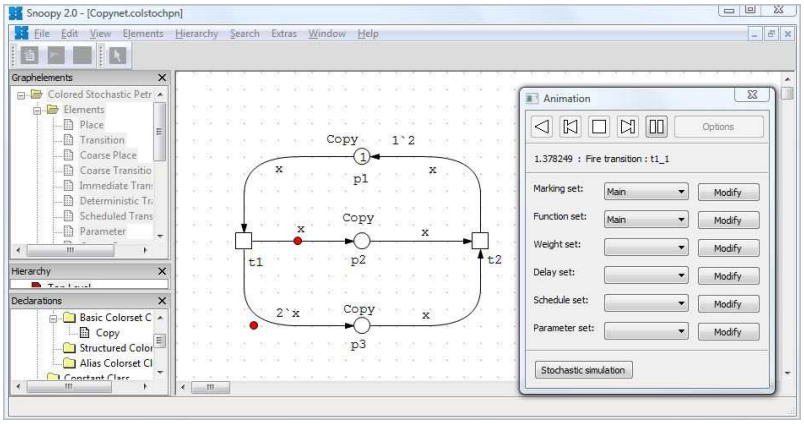

Figure 4.2: One animation snapshot.

4.1.2 Manual animation

When the user just clicks the transition to fire, then the binding selection dialogue will appear if this transition is enabled. For example, when we click the transition t1, we will get Figure 4.3. Then the user can select manual binding.

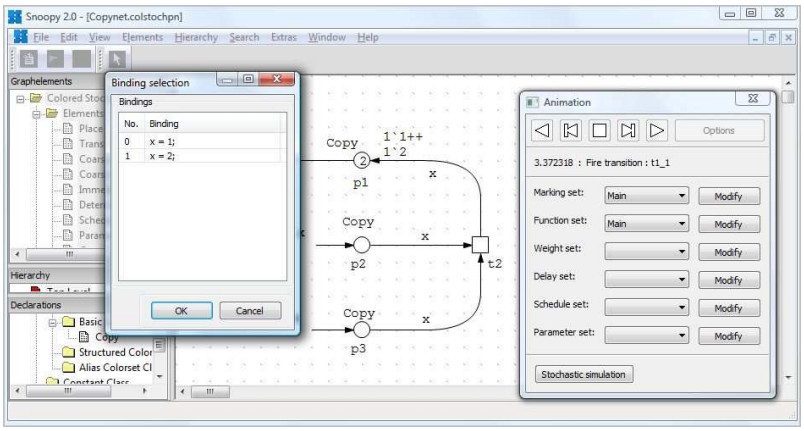

Figure 4.3: Manual animation snapshot.

4.2 Simulation (for $\mathcal{SPN}^C$ and $\mathcal{CPN}^C$)

When the user clicks the *Play forward/Pause* button, and then clicks *stochastic simulation* button, the simulation dialogue will appear. During this process, an

implicit unfolding is done, which unfolds a colored Petri net to a standard Petri net.

4.2.1 Run simulation

In the simulation dialogue (Figure 4.4), the user can first set simulation parameters, and then click the *Start simulation* button to start simulation. The settings include:

- Setting a marking set,
- Setting a rate function/weight/delay/schedule set,
- Setting a parameter set,
- Setting a simulation run interval, output step count, and simulation run number, and
- Choosing a simulation algorithm.

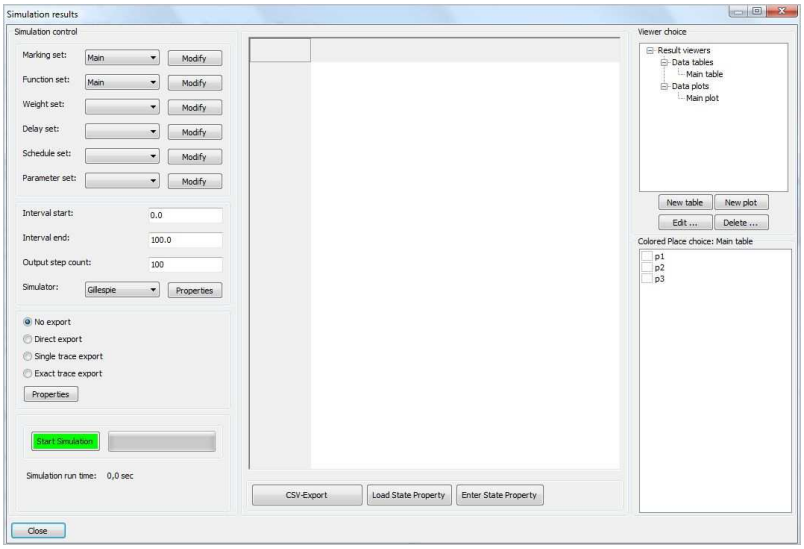

Figure 4.4: Simulation interface.

4.2.2 Show simulation results

The user can choose to show simulation results as a table or plot. Further, in a table or plot, the user can choose which information to be shown: colored, unfolded or both. For example, Figure 4.5 gives the plot show of colored places.

Plus, the user can eidt the table or plot to change settings of the informtion to be shown (Figure 4.6).

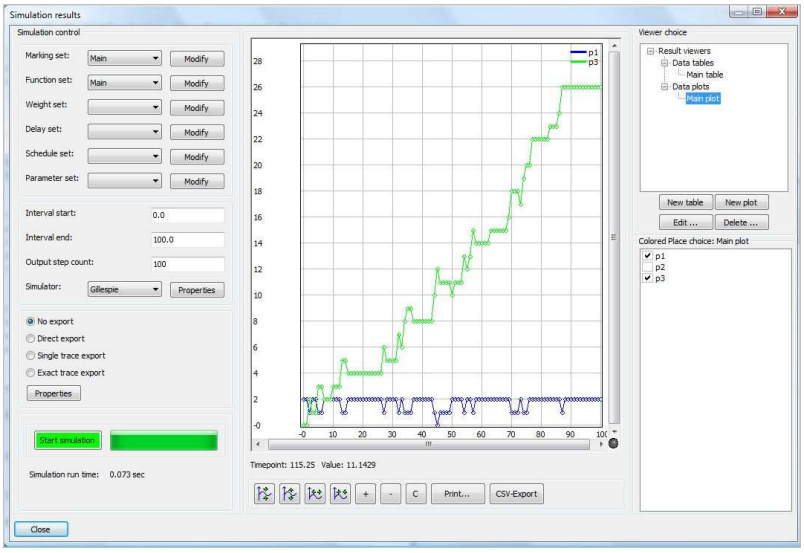

Figure 4.5: Plot for simulation results.

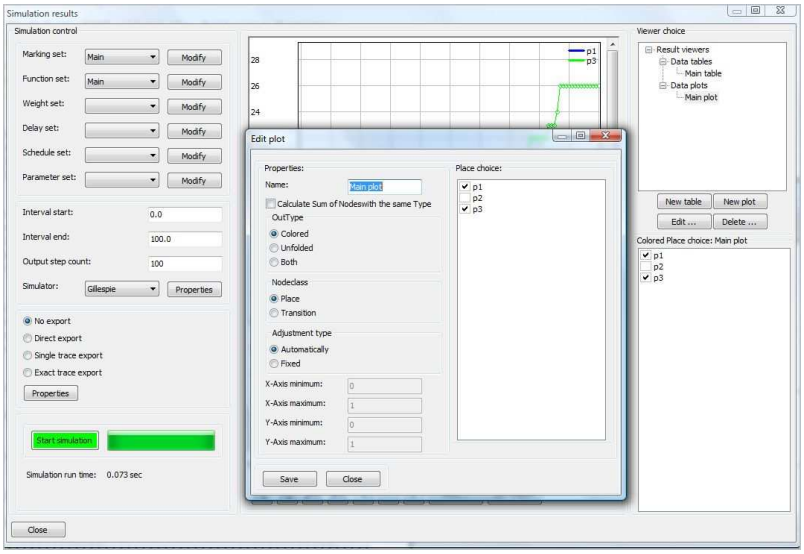

Figure 4.6: Edit plot.

4.2.3 Export simulation results

The user can choose which information to be exported to a file: colored, unfolded or both.

4.3 Analysis

4.3.1 Analysis using Charlie

We can export a colored Petri net to an uncolored Petri net, and then use Charlie [Cha11], [Fra09] to analyze its properties, e.g., P invariants and T invariants, or generate its reachability graph.

4.3.2 Analysis using Marcie

We can also export a colored stochastic Petri net to a stochastic Petri net, and then use the Marcie tool [Mar11], [SH09] to model check it.

4.3.3 Analysis using the MC2 tool

The MC2 tool [MC210] is used to analyze simulation traces of a stochastic model, so we can use MC2 to directly analyze simulation traces of a colored stochastic/continuous Petri net.

4.3.4 Analysis using CPN tools

We can export a colored Petri net produced by Snoopy to another colored Petri net readable by the CPN tools [CPN11]. So we can make use of the analysis tool of CPN tools [CPN11], [ASAP11] to analyze colored Petri nets at the colored level.

Chapter 5

Export/Import

5.1 $\mathcal{QPN}^{\mathcal{C}}$ export/import

5.1.1 Export to colored extended Petri nets

For an extended Petri net, the user can export it to a colored extended Petri net ($\mathcal{QPN}^{\mathcal{C}}$) by defining a color set *Dot*. After this transformation, the new net has the following features:

- All the places are set to the same color set, *Dot*.
- All the arcs are set to the same expression, *dot*.

5.1.2 Export to extended Petri nets

For a colored extended Petri net, the user can unfold it to an extended Petri net just by exporting it to an extended Petri net. During this process, all isolated nodes (places or transitions) are removed.

5.1.3 Export to colored stochastic Petri nets

For a colored extented Petri net ($\mathcal{QPN}^{\mathcal{C}}$), the user can transform it to a colored stochastic Petri net. All the information has been kept during this process, and all the rate functions for the transformed $\mathcal{SPN}^{\mathcal{C}}$ are set to $MassAction(1)$.

5.1.4 Export to CPN tools

For a colored extented Petri net ($\mathcal{QPN}^{\mathcal{C}}$), the user can transform it to a file read by CPN tools [CPN11]. After this transformation, sometimes we have to modify the arc or guard expressions to let them comply with the syntax of CPN tools. In summary, the following points should be noted:

- Modify user-defined functions in the declaration part,
- Change the syntax of predicates to the if-then-else syntax supported by CPN tools,

- Replace the operators of successor, predecessor etc. with user-defined functions,
- Modify arc expressions that belong to the union type.

5.1.5 Export declarations to a CSV file

We can export declarations of a colored Petri net to a csv file, which can be used for publication purposes or imported by other nets, i.e., when we creates a new colored net, we can import declarations from a CSV file for this new net.

5.1.6 Import declarations from a CSV file

Before defining a new colored Petri net, we can import declarations from a CSV file to reuse the declaration information which is defined before.

5.2 $\mathcal{SPN}^{\mathcal{C}}$ export/import

5.2.1 Export to colored stochastic Petri nets

For a stochastic Petri net, the user can export it to a colored stochastic Petri net by defining a color set *Dot*. After this transformation, the new net has the following features:

- All the places are set to the same color set, *Dot*.
- All the arcs are set to the same expression, *dot*.

5.2.2 Export to stochastic Petri nets

For a colored stochastic Petri net, the user can unfold it to a stochastic Petri net just by exporting it to a stochastic Petri net. During this process, all isolated nodes (places or transitions) are removed.

5.2.3 Export to colored extented Petri nets

For a colored stochastic Petri net, the user can transform it to a colored extented Petri net. After this transformation, all the information about rate functions is lost.

5.2.4 Export to CPN tools

For a colored extented Petri net ($\mathcal{QPN}^{\mathcal{C}}$), the user can transform it to a file read by CPN tools [CPN11]. After this transformation, sometimes we have to modify the arc or guard expressions to let them comply with the syntax of CPN tools.

5.2.5 Export declarations to a CSV file

We can export declarations of a colored Petri net to a csv file, which can be used for publication purposes or imported by other nets, that is, when we creates a new colored net, we can import declarations from a CSV file for this new net.

5.2.6 Import declarations from a CSV file

Before defining a new colored Petri net, we can import declarations from a CSV file to reuse the declaration information which is defined before.

5.3 CPN^C export/import

5.3.1 Export to colored continuous Petri nets

For a continuous Petri net, the user can export it to a colored continuous Petri net by defining a color set *Dot*. After this transformation, the new net has the following features:

- All the places are set to the same color set, *Dot*.
- All the arcs are set to the same expression, *dot*.

5.3.2 Export to continuous Petri nets

For a colored continuous Petri net, the user can unfold it to a continuous Petri net just by exporting it to a continuous Petri net. During this process, all isolated nodes (places or transitions) are removed.

5.3.3 Export to colored stochastic Petri nets

For a colored continuous Petri net, the user can transform it to a colored stochastic Petri net. After this transformation, the equations are transformed to rate functions.

5.3.4 Export declarations to a CSV file

We can export declarations of a colored continuousPetri net to a csv file, which can be used for publication purposes or imported by other nets, that is, when we creates a new colored net, we can import declarations from a CSV file for this new net.

5.3.5 Import declarations from a CSV file

Before defining a new colored Petri net, we can import declarations from a CSV file to reuse the declaration information which is defined before.

Chapter 6

Examples

6.1 Cooperative ligand binding

We consider an example of the binding of oxygen to the four subunits of a hemoglobin heterotetramer. The hemoglobin heterotetramer in the high and low affinity state binds to none, one, two, three or four oxygen molecules. Each of the ten states is represented by a place and oxygen feeds into the transitions that sequentially connect the respective places. The qualitative Petri net model is illustrated in Figure 6.1 (taken from [MWW10]).

Now we begin to construct a colored Petri net model for Figure 6.1. For this, we first partition Figure 6.1 into five subnets, each of which is embraced by a rectangle and is defined as a color. So we can use five integers, 0-5, to represent these five subnets. We then group similar places, which are marked with an identical color. The places in each group (with a specific color) are considered as a colored place. The net after partitioning and grouping is shown in Figure 6.2.

Now we obtain for Figure 6.1 a $\mathcal{QPN}^{\mathcal{C}}$ model, illustrated in Figure 6.3, and further a more compact $\mathcal{QPN}^{\mathcal{C}}$ model (Figure 6.4) by continuing folding the left and right parts. From Figure 6.3, we can see that the colored Petri net model reduces the size of the corresponding standard Petri net model. Moreover, comparing Figure 6.3 with Figure 6.4, we can also see that we can build colored Petri net model with different level of structural details, which is especially helpful for modeling complex biological systems. After automatic unfolding, these two colored models yield exactly the same Petri net model as given in Figure 6.1, i.e., the colored models and the uncolored model are equivalent. The declarations for these two $\mathcal{QPN}^{\mathcal{C}}$ models of the cooperative ligand binding are given in Table 6.1.

Besides, we give another colored model (see Figure 6.5), which uses user-defined functions and is equivalent to Figure 6.4. In this model, we define two functions $Fun1$ and $Fun2$ to replace lengthy expressions. See Table 6.1 for details about these two functions.

From these colored nets, we can also see that the folding operation does reduce the size of the net description for the prize of more complicated inscriptions. The graphic complexity is reduced, but the annotations of nodes and edges creates a new challenge. This is not unexpected since a more concise

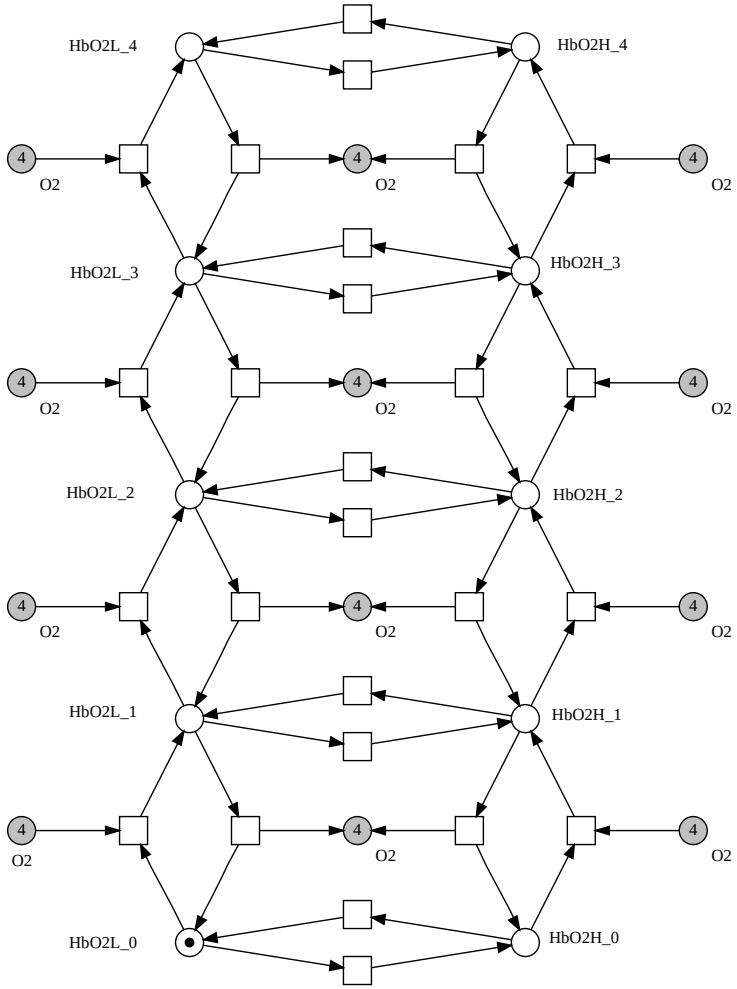

Figure 6.1: Cooperative binding of oxygen to hemoglobin represented as a Petri net model [MWW10]. For clarity, oxygen is represented in the form of multiple copies (logical places) of one place.

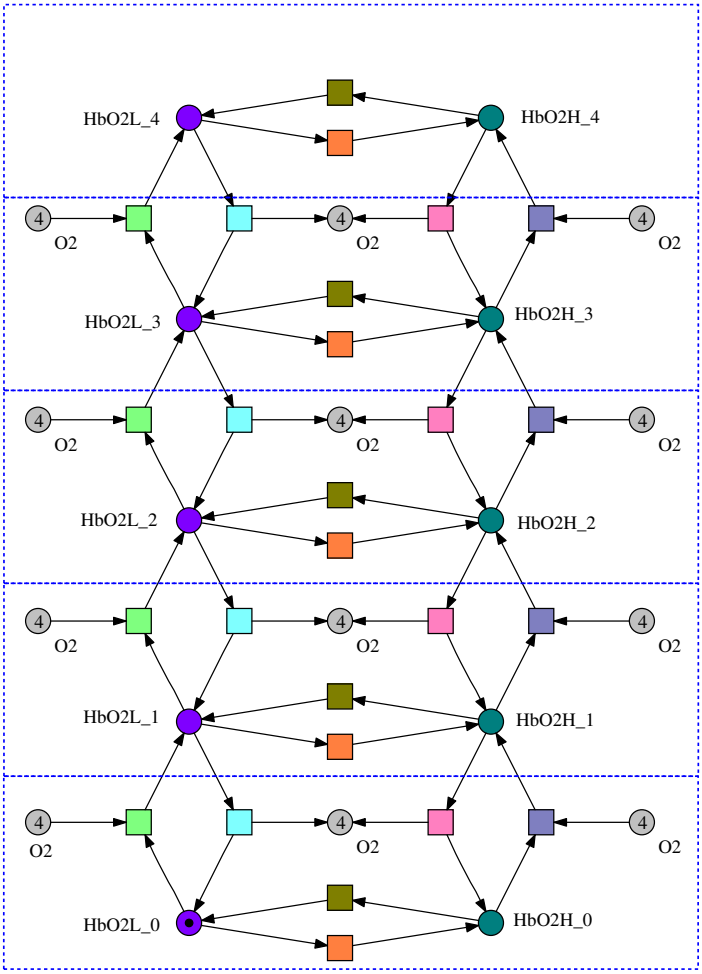

Figure 6.2: Cooperative binding of oxygen to hemoglobin represented as a Petri net model, which has been partitioned into subnets.

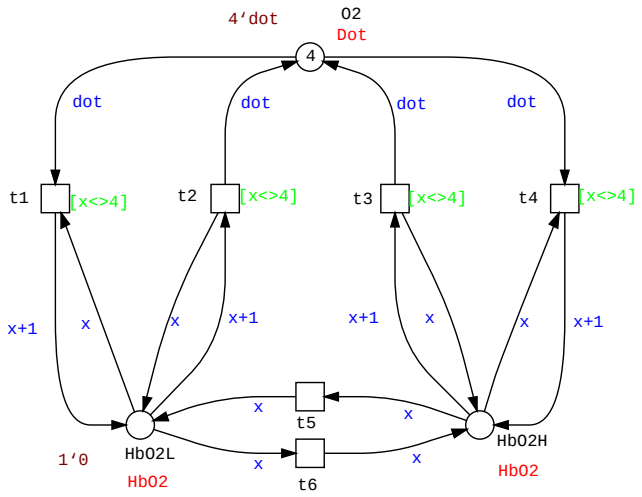

Figure 6.3: $\mathcal{QPN}^{\mathcal{C}}$ model for the cooperative binding of oxygen to hemoglobin, given as a standard Petri net in Figure 6.1. For declarations of color sets and variables, see 6.1.

Table 6.1: Declarations for the $\mathcal{QPN}^{\mathcal{C}}$ models of the cooperative ligand binding.

Declarations
colorset Dot = dot;
colorset HbO2 = int with 0-4;
colorset Level = enum with H,L;
colorset P = product with HbO2 × Level;
variable x: HbO2;
variable y: Level;
Function P Fun1(HbO2 x, Level y) {[y=L]1‘(x+1,y)++[y=H]1‘(x,y)};
Function P Fun2(HbO2 x, Level y) {[y=H]1‘(x+1,y)++[y=L]1‘(x,y)};

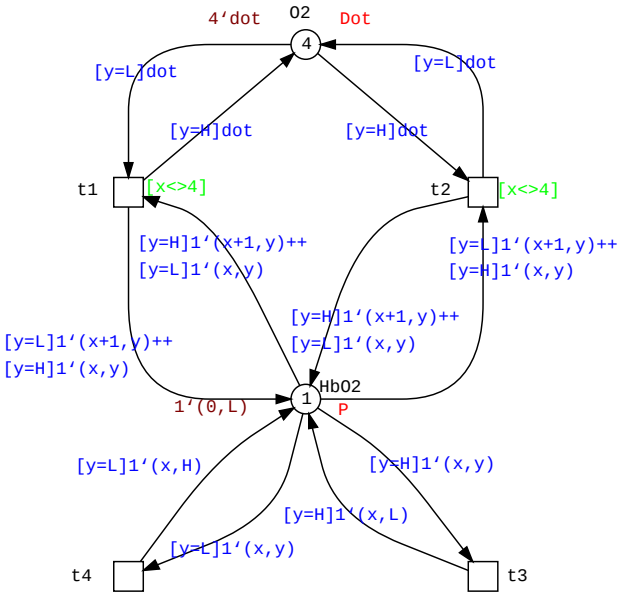

Figure 6.4: $\mathcal{QPN}^{\mathcal{C}}$ model for the cooperative binding of oxygen to hemoglobin, given as a standard Petri net in Figure 6.1. For declarations of color sets and variables, see Table 6.1.

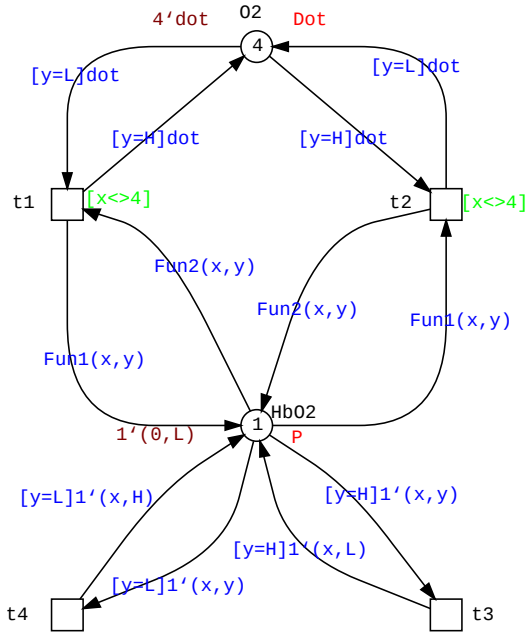

Figure 6.5: Another $\mathcal{QPN}^{\mathcal{C}}$ model for the cooperative binding of oxygen to hemoglobin, which uses user-defined functions and is equivalent to Figure 6.4. For declarations of color sets and variables, see Table 6.1.

write-up must rely on more complex components. Therefore, it is necessary to build a colored Petri net model at a suitable level of structural details.

6.2 Repressilator

In this section, we will demonstrate the $\mathcal{SPN}^{\mathcal{C}}$ using an example of a synthetic circuit - the repressilator, which is an engineered synthetic system encoded on a plasmid, and designed to exhibit oscillations [EL00]. The repressilator system is a regulatory cycle of three genes, for example, denoted by g_a, g_b and g_c, where each gene represses its successor, namely, g_a inhibits g_b, g_b inhibites g_c, and g_c inhibites g_a. This negative regulation is realized by the repressors, p_a, p_b and p_c, generated by the genes g_a, g_b and g_c respectively [LB07].

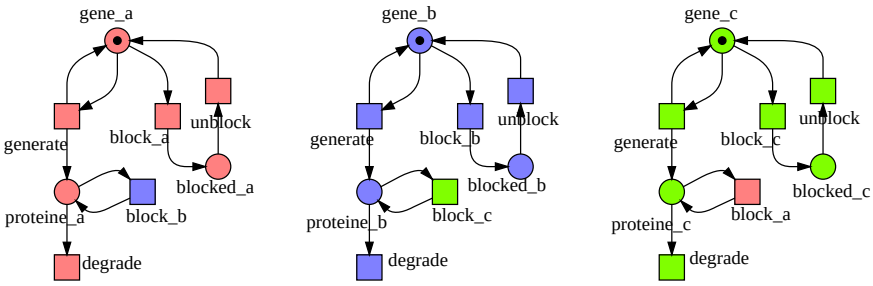

Figure 6.6: Stochastic Petri net model for the repressilator. The highlighted transitions are logical transitions.

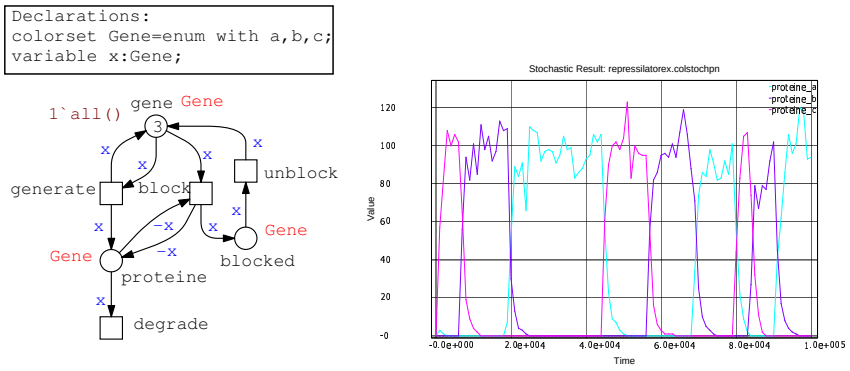

Figure 6.7: $\mathcal{SPN}^{\mathcal{C}}$ model of the standard Petri net given in Figure 6.6, and one simulation run plot for the repressilator. For rate functions, see Table 6.2.

As our purpose is to demonstrate the $\mathcal{SPN}^{\mathcal{C}}$, we only consider a relatively simple model of the repressilator, which was built as a stochastic π-machine in [BCP08]. Based on that model, we build a stochastic Petri net model (Figure 6.6), and further a $\mathcal{SPN}^{\mathcal{C}}$ model for the repressilator (shown on the left hand of Figure 6.7). This colored model when unfolded yields the same uncolored Petri net model in Figure 6.6.

Table 6.2: Rate functions for the $\mathcal{SPN}^{\mathcal{C}}$ model of the repressilator.

Transition	Rate function
generate	$0.1 * gene$
block	$1.0 * proteine$
unblock	$0.0001 * blocked$
degrade	$0.001 * proteine$

For the $\mathcal{SPN}^{\mathcal{C}}$ model in Figure 6.7, there are three colors, a, b, and c to distinguish three similar components in Figure 6.6. The predecessor operator "-" in the arc expression $-x$ returns the predecessor of x in an ordered finite color set. If x is the first color, then it returns the last color.

As described above, the $\mathcal{SPN}^{\mathcal{C}}$ will be automatically unfolded to a stochastic Petri net, and can be simulated with different simulation algorithms. On the right hand of Figure 6.7 a snapshot of a simulation run result is given. The rate functions are given in Table 6.2 (coming from [PC07]). The $\mathcal{SPN}^{\mathcal{C}}$ model exhibits the same behavior compared with that in [PC07].

From Figure 6.7, we can see that the $\mathcal{SPN}^{\mathcal{C}}$ model reduces the size of the original stochastic Petri net model to one third. More importantly, when other similar subnets have to be added, the model structure does not need to be modified and what has to be done is only to add extra colors.

For example, we consider the generalized repressilator with an arbitrary number n of genes in the loop that is presented in [MHE+06]. To build its $\mathcal{SPN}^{\mathcal{C}}$ model, we just need to modify the color set as n colors, and do not need to modify anything else. For example, Figure 6.8 gives the conceptual graph of the generalized repressilator with $n = 9$ (on the left hand), and one simulation plot (on the right hand), whose rate functions are the same as in Table 6.2. Please note, the $\mathcal{SPN}^{\mathcal{C}}$ model for the generalized repressilator is the same as the one for the three-gene repressilator, and the only difference is that we define the color set as n colors rather than 3 colors. This demonstrates a big advantage of color Petri nets, that is, to increase the colors means to increase the size of the net.

6.3 Where to find more examples

In [GLG+11], [GLT+11], colored (both stochastic and continuous) Petri nets have been used to describe the phenomenon of Planar Cell Polarity (PCP) signaling in Drosophila wing. Two colored models (abstract and refined) has been developed, which model a group of cells on a two-dimensional grid, corresponding to a fragment of the wing tissue. Moreover each cell is partitioned into seven virtual compartments, so these two models has a two-level hierarchy. In addition, these models involves product color sets, subsets of color sets, user-defined functions and etc.

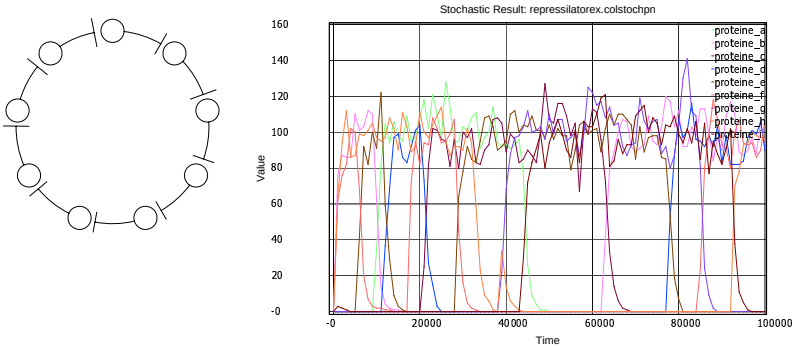

Figure 6.8: Conceptual graph and one simulation run plot for the repressilator with 9 genes.

Bibliography

[ASAP11] ASAP: http://www.daimi.au.dk/~ascoveco/asap.html (2011)

[BCP08] R. Blossey, L. Cardelli, A. Phillips: Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation. HFSP Journal. 2(1), 17-28 (2008)

[BNF11] BNF: http://en.wikipedia.org/wiki/Backus_Naur_Form (2011)

[Cha11] Charlie - a Software Tool to Analyze Place/Transition Nets: http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Charlie (2011)

[CPN11] CPN tools: http://cpntools.org/ (2011)

[EL00] M. B. Elowitz, S. Leibler: A Synthetic Oscillatory Network of Transcriptional Regulators. Nature. 403, 335-338 (2000)

[Fra09] A. Franzke: Charlie 2.0 - a Multi-Threaded Petri Net Analyzer. Diploma Thesis, Brandenburg University of Technology Cottbus. (2009)

[GHL07] D. Gilbert, M. Heiner, S. Lehrack : A Unifying Framework for Modelling and Analysing Biochemical Pathways Using Petri Nets. Proc. International Conference on Computational Methods in Systems Biology. LNCS/LNBI, 4695, 200-216 (2007)

[Gil77] D. T. Gillespie: Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry. 81(25), 2340-2361 (1977)

[GL79] H. J. Genrich, K. Lautenbach: The Analysis of Distributed Systems by Means of Predicate/Transition-Nets. Semantics of Concurrent Computation. LNCS, 70, 123-146 (1979)

[GL81] H. J. Genrich, K. Lautenbach: System Modelling with High-Level Petri Nets. Theoretical Computer Science. 13(1), 109-135 (1981)

[GLG+11] Q. Gao, F. Liu, D. Gilbert, M. Heiner, D. Tree: A Multiscale Approach to Modelling Planar Cell Polarity in Drosophila Wing using Hierarchically Coloured Petri Nets. Proc. 9th International Conference on Computational Methods in Systems Biology (CMSB 2011). ACM digital library (2011)

[GLT+11] Q. Gao, F. Liu, D. Tree, D. Gilbert: Multi-cell Modeling Using Coloured Petri Nets Applied to Plannar Cell Polarity. Proc. 2th International Workshop on Biological Processes & Petri nets. CEUR Workshop Proceedings, 724, 135-150 (2011)

[HLG+09] M. Heiner, S. Lehrack, D. Gilbert, W. Marwan: Extended Stochastic Petri Nets for Model-based Design of Wetlab Experiments. Transaction on Computational Systems Biology XI. LNCS/LNBI, 5750, 138-163 (2009)

[HRR+08] M. Heiner, R. Richter, C. Rohr, M. Schwarick: Snoopy - A Tool to Design and Execute Graph-Based Formalisms. [Extended Version]. Petri Net Newsletter. 74, 8-22 (2008)

[Jen81] K. Jensen: Coloured Petri Nets and the Invariant-Method. Theoretical Computer Science. 14(3), 317-336 (1981)

[JKW07] K. Jensen, L. M. Kristensen, L. M. Wells: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer. 9(3/4), 213-254 (2007)

[LB07] A. Loinger, O. Biham: Stochastic Simulations of the Repressilator Circuit. Phisical Review. 76(5), 051917(9) (2007)

[Mar11] Marcie - a Tool for Qualitative and Quantitative Analysis of Generalized Stochastic Petr Nets. http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie (2011)

[MC210] MC2 website: MC2 - A PLTL Model Checker. University of Glasgow, http://www.brc.dcs.gla.ac.uk/~drg/courses/sysbiomres/software/mc2 (2010)

[MHE+06] S. Muller, J. Hofbauer, L. Endler, C. Flamm, S. Widder, P. Schuster: A Generalized Model of the Repressilator. Journal of Mathematical Biology. 53, 905-937 (2006)

[MWW10] W. Marwan, A. Wagler, R. Weismantel: Petri Nets as a Framework for the Reconstruction and Analysis of Signal Transduction Pathways and Regulatory Networks. Natural Computing. 10(2), 639-654 (2010)

[PC07] A. Phillips, L. Cardelli: Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-Calculus. Proc. Computational Methods in Systems Biology. LNCS, 4695, 184-199 (2007)

[RMH10] C Rohr, W Marwan, M Heiner: Snoopy - a Unifying Petri Net Framework to Investigate Biomolecular Networks: Bioinformatics. 26(7), 974-975 (2010)

[SH09] M. Schwarick, M. Heiner: CSL Model Checking of Biochemical Networks with Interval Decision Diagrams. Proc. 7th International Conference on Computational Methods in Systems Biology (CMSB 2009), LNBI 5688, 296-312 (2009)

Appendix A

Annotation Language

A.1 Introduction to BNF

A BNF specification is a set of derivation rules, written as [BNF11]

$$symbol ::= expression$$

where:

1. *symbol* is a nonterminal; *expression* consists of one or more sequences of symbols; more sequences are separated by the vertical bar, $'|'$, indicating a choice, the whole being a possible substitution for the symbol on the left.
2. Symbols that never appear on a left side are terminals, which are notated by using the single quotation marks ' '.
3. Symbols that appear on a left side are non-terminals.
4. ::= means "is defined as".

A.2 BNF for the data type definition

type	::=	*simple_type*
	|	*compound_type*
simple_type	::=	*type_identifier*
	|	*structured_type*
type_identifier	::=	*unsigned_integer*
	|	*boolean*
	|	*string*
unsigned_integer	::=	$'int'$
boolean	::=	$'bool'$
string	::=	$'string'$
structured_type	::=	*enumeration*
	|	*index*
enumeration	::=	*identifier_list*
identifier_list	::=	*identifier*
	|	*identifier_list* $','$ *identifier*
index	::=	*identifier* $'['$ *index_specifier* $']'$
index_specifier	::=	$'int'$
compound_type	::=	*product*
	|	*union*
product	::=	*type* $'\times'$ *type*
	|	*product* $'\times'$ *type*
union	::=	*type*
	|	*union* $','$ *type*

A.3 BNF for the annotation language

CPN_expr	::=	*Multiset_expr*
Multiset_expr	::=	*Predicate_expr*
	|	*Multiset_expr MSAdditionOp Predicate_expr*
MSAdditionOp	::=	$'++'$
Predicate_expr	::=	*Separate_expr*
	|	$'['$ *Or_expr* $']'$ *Separate_expr*
Separate_expr	::=	*Tuple_expr*
	|	*Separate_expr SeparatorOp Tuple_expr*
SeparatorOp	::=	$'`'$
Tuple_expr	::=	*Or_expr*
	|	$'('$ *Comma_expr* $')'$
Comma_expr	::=	*Tuple_expr*
	|	*Comma_expr CommaOp Tuple_expr*
CommaOp	::=	$','$
Or_expr	::=	*And_expr*
	|	*Or_expr OrOp And_expr*
OrOp	::=	$'\|'$
And_expr	::=	*Equal_expr*
	|	*And_expr AndOp Equal_expr*
AndOp	::=	$'\&'$
Equal_expr	::=	*Relation_expr*
	|	*Equal_expr EqualOp Relation_expr*
EqualOp	::=	$'='$
	|	$'<>'$
Relation_expr	::=	*Add_expr*
	|	*Relation_expr RelationOp Add_expr*
RelationOp	::=	$'<'$
	|	$'<='$
	|	$'>='$
	|	$'>'$

Add_expr	::=	*Multiplicity_expr*
	|	*Add_expr AddOp Multiplicity_expr*
AddOp	::=	$'+'$
	|	$'-'$
Multiplicity_expr	::=	*Unary_expr*
	|	*Multiplicity_expr MultiplicityOp Unary_expr*
MultiplicityOp	::=	$'*'$
	|	$'/'$
	|	$'\%'$
Unary_expr	::=	*Postfix_expr*
	|	*UnaryOp Postfix_expr*
UnaryOp	::=	$'+'$
	|	$'-'$
	|	$'@'$
	|	$'!'$
Postfix_expr	::=	*Atom_expr*
	|	*Postfix_expr* $'['$ *Atom_expr* $']'$
	|	*Postfix_expr DotOp Atom_expr*
DotOp	::=	$'.'$
Atom_expr	::=	*Constant*
	|	*Variable*
	|	*Function*
	|	$'('$ *CPN_expr* $')'$
Constant	::=	*Integer*
	|	*String*
Variable	::=	*Identifier*
Function	::=	*Identifier*$'('$ *ArgumentList* $')'$ $'\{'$*Function_body*$'\}'$
ArgumentList	::=	*Or_expr*
	|	*ArgumentList CommaOp Or_expr*
Function_body	::=	*Multiset_expr*
Integer	::=	*Digit*
	|	*Integer Digit*
String	::=	*LetterOrDigit*
	|	*String LetterOrDigit*
Identifier	::=	*Letter*
	|	*Identifier LetterOrDigit*
LetterOrDigit	::=	*Letter*
	|	*Digit*
Digit	::=	$"0-9"$
Letter	::=	$"A-Za-z\_"$