FACULDADE DE INFORMÁTICA
PUCRS - Brazil

http://www.pucrs.br/inf/pos/

Why you should care about Generalized Tensor Algebra

L. Brenner, P. Fernandes, A. Sales

Technical Report Series

Number 037
November, 2003

Contact:
lbrenner@inf.pucrs.br
www.inf.pucrs.br/~lbrenner

paulof@inf.pucrs.br
www.inf.pucrs.br/~paulof

asales@inf.pucrs.br
www.inf.pucrs.br/~asales

Published by PPGCC - FACIN - PUCRS
Av. Ipiranga, 6681
90619-900 Porto Algre - RS - Brasil

1 Introduction

In the middle of the XIX century, the german mathematician Leopold Kronecker [16] proposed a new operation based on *tensors*, a generalization of the matrices where more than two dimension could be represented. This extension of the linear algebra was represented by one single operator called *Kronecker product*. Since then many researchers, mainly physicists, have used the Kronecker products to represent operations over multidimensional structures.

Since the very beginning of computer science, linear algebra is probably the most important application of numerical techniques. However, as far as the authors know, only in the late 70's computer scientists pay some attention to the Kronecker extension to the linear algebra. The works of Bellman [5], Brewer [7], Amoia, De Micheli and Santomauro [3], and Davio [11] are some of the first studies concerning the Kronecker product operation applied to the computer science. These authors and many of the subsequent works do not really agree about the name of the operation and the terms *direct product*, or *tensor product* are used to describe exactly the same operation. In many of these works, the first references to a new operation over tensors made its appearance: *tensor sum*. Logically, tensor sum is also known as *Kronecker sum* or *direct sum*. Tensor product and tensor sum operations, and their properties compose the *Classical Tensor Algebra* (CTA). In our work we prefer to use the terms *tensor product* and *tensor sum*, since they explicit the tensor (multidimensional) concept.

In the 80's, Plateau [19] proposed the first modeling formalism that use Tensor Algebra as representation: *Stochastic Automata Networks* (SAN). Sometime later, Donatelli [13] proposed another modeling formalism using Tensor Algebra: *Superposed Generalized Stochastic Petri Nets* (SGSPN). Both formalisms were at the base of the application of Kronecker-based representations. Comparing these Kronecker representations to the classical approach (a huge sparse matrix), it is obvious that the memory needs are dramatically reduced. This memory savings are quite important to reduce the impact of the classical state space explosion problem. Unfortunately, a similar reduction of the time spend to compute solutions is yet to come.

To cope with this problem, the research community has been working on numerical techniques to reduce the computational cost in computing exact solutions using both iterative [14] and direct methods [9]. Such works achieve some important gains, but the reduction is not as big as the reduction in memory needs. One of the main goals of our work is to stress some advantages of using *Generalized Tensor Algebra* (GTA), which is an extension to the CTA. In fact, we do believe that GTA can provide a more compact and more manageable representation from a numerical point of view.

For some years now, different structured approaches proposed more effective memory and computational gains [17, 18]. The main purpose of this work is turn back the attention to the Kronecker representations, stressing some promising characteristics due to the Generalized Tensor Algebra (GTA). Our first goal is to prove that GTA operators can be viewed as a compact form to describe complex CTA formulas. Therefore, models described with Kronecker representations can often be described by GTA operators in a more compact (and efficient) form. This report shows that SAN, which uses GTA, has the same application scope of SGSPN, which uses CTA.

We also show that any SAN model with functions has at least one equivalent representation without functions. In fact, the use of functions, and consequently the GTA, is not really a "need" since there is a formal equivalente of formalisms, but in some cases it represents, in a computational cost point of view, some irrefutable "advantages". Some modeling examples are presented in order to numerically support this affirmative.

Some segments of the research community made a considerable, and fruitful, effort to provide solutions for models with large and complex state space [8]. However, such efforts could provide satisfactory solutions for models with a relatively small number of reachable states. We do

not want to contest such gains achieved by state space analysis performed using *Multi-Decision Diagrams* (MDD) and *Matrix Diagrams* (MxD) techniques [18]. Nevertheless, we want to show that for models with a large number of reachable states the GTA approach has irrefutable gains.

The next section briefly describes Classical (CTA) and Generalized Tensor Algebra (GTA). Section 3 describes formally SAN and SGSPN formalisms. Section 4 describes the equivalence between SAN models with functions (needing GTA operators) and SAN models without functions (described only with CTA operators), as well as the representation equivalence between SAN and SGSPN formalisms. Some modeling examples are presented in Section 5. In Section 6, we show the required computational cost to achieve a transient or stationary solution of a SAN and a SGSPN model expressed in a tensor format. Section 7 shows the memory needs and CPU time to the generation, and the solution of the presented models in Section 5.

2 Tensor Algebra

In this section, the concepts of Classical Tensor Algebra [3, 11] and Generalized Tensor Algebra [19, 14] are presented.

2.1 CTA - Classical Tensor Algebra

Define two matrices A and B as follows:

$$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \qquad B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \end{pmatrix}$$

The *tensor product* $C = A \otimes B$ is given by

$$C = \begin{pmatrix} a_{00}B & a_{01}B \\ a_{10}B & a_{11}B \end{pmatrix}$$

In general, to define the tensor product of two matrices: A of dimensions $(\rho_1 \times \gamma_1)$ and B of dimensions $(\rho_2 \times \gamma_2)$, it is convenient to observe that the tensor product matrix has dimensions $(\rho_1\rho_2 \times \gamma_1\gamma_2)$ and may be considered as consisting of $\rho_1\gamma_1$ blocks each having dimensions $(\rho_2\gamma_2)$, that is, the dimensions of B. To specify a particular element, it suffices to specify the block in which the element occurs and the position within that block of the element under consideration. Thus, as mentioned previously, element c_{36} $(a_{11}b_{02})$ is in the (1, 1) block and at position (0, 2) of that block. The tensor product $C = A \otimes B$ is defined by assigning to the element of C that is in the (k, l) position of block (i, j), the value $a_{ij}b_{kl}$, *i.e.*:

$$C_{[ik][jl]} = a_{ij}b_{kl}.$$

The *tensor sum* of two *square* matrices A and B is defined in terms of tensor products as:

$$A \oplus B = A \otimes I_{n_B} + I_{n_A} \otimes B$$

where n_A is the order of A; n_B the order of B; I_{n_i} the identity matrix of order n_i and "+" represents the usual operation of matrix addition. Since both sides of this operation (matrix addition) must have identical dimensions, it follows that tensor addition is defined for square matrices only. The value assigned to the element $C_{[ik][jl]}$ of the tensor sum $C = A \oplus B$ is defined as:

$$C_{[ik][jl]} = a_{ij}\delta_{kl} + b_{kl}\delta_{ij},$$

where δ_{ij} is the element of i^{th} row and j^{th} column of an identity matrix[1] defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Some important properties of tensor product and sum operations defined by Davio [11] are:

- Associativity:
  $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ and $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- Distributivity over (ordinary matrix) addition:
  $(A + B) \otimes (C + D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D)$
- Compatibility with (ordinary matrix) multiplication:
  $(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D)$
- Compatibility over multiplication:
  $A \otimes B = (A \otimes I_{n_B}) \times (I_{n_A} \otimes B)$
- Commutativity of normal factors[2]:
  $(A \otimes I_{n_B}) \times (I_{n_A} \otimes B) = (I_{n_A} \otimes B) \times (A \otimes I_{n_B})$

2.2 GTA - Generalized Tensor Algebra

Generalized Tensor Algebra is an extension from Classical Tensor Algebra. The main distinction of GTA with respect to CTA is the addition of the concept of the *functional elements*. However a matrix can be composed of constant elements (belonging to $\mathbb{R}$) or functional elements. A functional element is a function evaluated in $\mathbb{R}$ according to a set of parameters composed of the rows of the one or more matrices. Generalized tensor product is denoted by $\underset{g}{\otimes}$. The value assigned to the element $C_{[ik][jl]}$ of the generalized tensor product $C = A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$ is defined as:

$$C_{[ik][jl]} = a_{ij}(b_k)b_{kl}(a_i).$$

Generalized tensor sum is also analogous to the ordinary tensor sum, and it is denoted by $\underset{g}{\oplus}$. The elements of the generalized tensor sum $C = A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ are defined as:

$$C_{[ik][jl]} = a_{ij}(b_k)\delta_{kl} + b_{kl}(a_i)\delta_{ij}.$$

GTA properties defined by Fernandes, Plateau and Stewart [14] are:

- Associativity:
  $[\ A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} B(\mathcal{A},\mathcal{C})\ ] \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} [\ B(\mathcal{A},\mathcal{C}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B})\ ]$
  $[\ A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} B(\mathcal{A},\mathcal{C})\ ] \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} [\ B(\mathcal{A},\mathcal{C}) \underset{g}{\oplus} C(\mathcal{A},\mathcal{B})\ ]$
- Distributivity over addition:
  $[A(\mathcal{C},\mathcal{D}) + B(\mathcal{C},\mathcal{D})] \underset{g}{\otimes} [C(\mathcal{A},\mathcal{B}) + D(\mathcal{A},\mathcal{B})] = A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) + A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) + B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) + B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B})$

[1] δ is commonly known as Kronecker's operator.

[2] Although this property could be inferred from the *Compatibility with (ordinary matrix) multiplication*, it was defined by Fernandes, Plateau and Stewart [14].

- Compatibility over multiplication I:
  $A \underset{g}{\otimes} B(\mathcal{A}) = I_{n_A} \underset{g}{\otimes} B(\mathcal{A}) \times A \underset{g}{\otimes} I_{n_B}$
- Compatibility over multiplication II:
  $A(\mathcal{B}) \underset{g}{\otimes} B = A(\mathcal{B}) \underset{g}{\otimes} I_{n_B} \times I_{n_A} \underset{g}{\otimes} B$
- Decomposability of Generalized Tensor Product into Ordinary Tensor Product:

$$A \underset{g}{\otimes} B(\mathcal{A}) = \sum_{k=1}^{n_A} \ell_k(A) \otimes B(a_k)$$

3 Modeling Formalisms

Many formalisms can benefit from a structured tensor (Kronecker) representation, but in this section we describe the two formalisms that explicit define their tensor representations: Stochastic Automata Networks (SAN), and Superposed Generalized Stochastic Petri Nets (SGSPN). It is important to notice that the use of Kronecker representations is not limited to SAN and SGSPN, but can also be employed to any other structured formalism, *e.g.*, Process Algebra [15] and Stochastic Activity Networks [20].

Stochastic Automata Networks (SAN) formalism was proposed by Plateau [19]. The basic idea of SAN is to represent a whole system by a collection of subsystems with an independent behavior (*local transitions*) and occasional interdependencies (*functional rates* and *synchronizing events*). The framework proposed by Plateau define a modular way to describe continuous and discrete-time Markovian models[3] [4]. SAN formalism has exactly the same application scope as the Markov chains. In fact, any SAN model has one equivalent Markov chain model, and any Markov chain can be viewed as a SAN with only one automaton.

Donatelli [12] proposed a formalism called Superposed Stochastic Automata (SSA), a class of Stochastic Petri Nets (SPN). SSA is a set of stochastic state machines that interact through transition superposition, which the solution can be efficiently computed, since it can derive automatically a tensor expression for its infinitesimal generator. Later, Donatelli [13] extended the SSA formalism to SGSPN and show how the SSA solution can be adapted to work for this formalism. A SGSPN model is defined as a set of Generalized Stochastic Petri Nets (GSPN) synchronized by a common subset of transitions. GSPN [2] is derived from SPN and it contains two types of transitions: *timed* and *immediate*. An exponentially distributed random firing time is associated with each timed transition, whereas immediate transitions, by definition, fire in zero time. Immediate transitions always have precedence to fire over timed transitions.

3.1 SAN - Stochastic Automata Networks

In this report, it will be considered the formalization of a SAN model comprising N automata and E events.

Let

- $\mathcal{A}$ set of automata ($|\mathcal{A}| = N$);
- $\mathcal{E}$ set of events ($|\mathcal{E}| = E$);
- $\mathcal{F}$ reachability function.

Set of automata $\mathcal{A}$ comprises N automata named $\mathcal{A}^{(i)}$, where $i \in [1..N]$.

Let

[3] In the scope of this report, we will be restricted to continuous-time Markovian models, *i.e.*, every change among states is defined as a stochastic process (exponentially distributed) numerically expressed by a rate.

$\mathcal{S}^{(i)}$ set of (local) states of automaton $\mathcal{A}^{(i)}$;

$|\mathcal{S}^{(i)}|$ number of states in $\mathcal{S}^{(i)}$;

$\mathcal{S}$ product state space of a SAN model: $\mathcal{S}^{(1)} \times \ldots \times \mathcal{S}^{(N)}$;

$x^{(i)}$ a local state of automaton $\mathcal{A}^{(i)}$ ($x^{(i)} \in \mathcal{S}^{(i)}$).

Definition 1 *Global state $\tilde{x}$ of a SAN model is obtained by the combination of local states of the N automata, $\tilde{x} = (x^{(1)}, \ldots, x^{(N)})$, where $x^{(i)}$ is the local state of automaton $\mathcal{A}^{(i)}$ ($\tilde{x} \in \mathcal{S}$).*

Let

$\tilde{x}^{(\omega)}$ composition of local states $x^{(i)}$, where $i \in \omega$ and $\omega \subseteq [1..N]$;

$\tilde{x}(x^{(i)} \rightarrow y^{(i)})$ global state obtained by the substitution of local state $x^{(i)}$ for $y^{(i)}$ in automaton $\mathcal{A}^{(i)}$;

$\mathcal{S}^{(\omega)}$ product state space of local states of automaton $\mathcal{A}^{(i)}$, where $i \in \omega$;

$|\mathcal{S}^{(\omega)}|$ number of states in $\mathcal{S}^{(\omega)}$, where ω is the set of indices ($\omega \subseteq [1..N]$).

Definition 2 *Functional element $f(\mathcal{S}^{(\omega)})$ is a function of $\mathcal{S}^{(\omega)} \rightarrow \mathbb{R}^+$, where $\omega \subseteq [1..N]$.*

Automata $\mathcal{A}^{(i)}$ with $i \in \omega$ are the parameters of element $f(\mathcal{S}^{(\omega)})$. The functional elements define functional probabilities and functional rates. In this formal description, all rates and probabilities will be considered as functional elements, even though they are constant. Obviously, such choice represents no restriction, since constant elements can be viewed as (constant) functions with no parameters, *i.e.*, $\omega = \emptyset$.

Let

$f(\tilde{x}^{(\omega)})$ functional element[4] $f(\mathcal{S}^{(\omega)})$ evaluated to the composition of states $\tilde{x}^{(\omega)}$.

Definition 3 *An event in a SAN model is defined by: identifier e, where $e \in \mathcal{E}$; and the index of the master automaton[5] $\iota^{(e)}$, where $\iota^{(e)} \in [1..N]$.*

Definition 4 *An event tuple (e, τ) is composed of: event identifier e; and a functional element τ of $\mathcal{S} \rightarrow \mathbb{R}^+$, which defines the occurrence rate to event e.*

Definition 5 *Set $\mathcal{T}$ has all transition tuples (e, π). A transition tuple (e, π) is defined by: event identifier e; and functional element π of $\mathcal{S} \rightarrow [0, 1]$, which defines the probability from a transition.*

Definition 6 *$\mathcal{Q}^{(i)}$ is a transition function of $\mathcal{S}^{(i)} \times \mathcal{S}^{(i)} \rightarrow \mathcal{T}^*$, which has the transition labels of automaton $\mathcal{A}^{(i)}$.*

Definition 7 *$\tilde{\mathcal{Q}}$ is a transition function of $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{T}^*$, which has the transition labels of the global automaton.*

Let

$\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$ transition label from local state $x^{(i)}$ to $y^{(i)}$ in $\mathcal{Q}^{(i)}$, which has a list of transition tuples (e, π) in $\mathcal{T}$;

$\tilde{\mathcal{Q}}(\tilde{x}, \tilde{y})$ transition label from global state $\tilde{x}$ to $\tilde{y}$ in $\tilde{\mathcal{Q}}$, which has a list of transition tuples (e, π) in $\mathcal{T}$;

$\eta^{(e)}$ set of indices i ($i \in [1..N]$) such that automaton $\mathcal{A}^{(i)}$ has at least one transition tuple with event identifier e in any element of $\mathcal{Q}^{(i)}$;

$\mathcal{S}^{(\eta^{(e)})}$ product state space of set of indices $\eta^{(e)}$;

[4] A functional element can be a rate ($\tau(\mathcal{S}^{(\omega)})$) or a probability ($\pi(\mathcal{S}^{(\omega)})$), and it can be also expressed in the evaluated form: $\tau(\tilde{x}^{(\omega)})$ and $\pi(\tilde{x}^{(\omega)})$ respectively.

[5] The master/slave semantic is used to the formal definition of synchronizing events. However any semantics can be used without loss of generality.

$|\mathcal{S}^{(\eta^{(e)})}|$ number of states in $\mathcal{S}^{(\eta^{(e)})}$, in which a transition tuple $(e,\pi) \subset \tilde{Q}^{(\eta^{(e)})}(\tilde{x}^{(\eta^{(e)})}, \tilde{y}^{(\eta^{(e)})})$.

Definition 8 *An event e is classified as:*

8.1. local event, if $|\eta^{(e)}|= 1$*;*

8.2. synchronizing event, if $|\eta^{(e)}|> 1$.

Definition 9 *Set of local events* $\mathcal{E}_l$ *is defined as* $\mathcal{E}_l = \{e \in \mathcal{E} \; ||\eta^{(e)}|= 1\}$.

Definition 10 *Set of synchronizing events* $\mathcal{E}_s$ *is defined as* $\mathcal{E}_s = \{e \in \mathcal{E} \; ||\eta^{(e)}|> 1\}$.

Definition 11 *Set of events* $\mathcal{E}$ *is defined as* $\mathcal{E} = \mathcal{E}_l \cup \mathcal{E}_s$ *and* $\mathcal{E}_l \cap \mathcal{E}_s = \emptyset$.

Definition 12 *An automaton* $\mathcal{A}^{(i)}$ *is defined by: a set of states* $\mathcal{S}^{(i)}$*; and a transition function* $\mathcal{Q}^{(i)}$.

Let

$\tau_e(x^{(i)}, y^{(i)})$ occurrence rate of event e whose transition tuple (e,π) is associated to transition $\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$;

$\pi_e(x^{(i)}, y^{(i)})$ probability of event e whose transition tuple (e,π) is associated to transition $\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$;

$succ_e(x^{(i)})$ set of successor states $y^{(i)}$ such that transition $\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$ has a transition tuple with event identifier e and $\tau_e(x^{(i)}, y^{(i)}) \neq 0$, $\pi_e(x^{(i)}, y^{(i)}) \neq 0$. The set of successor states of event e in $x^{(i)}$ may be empty, if the transition can not fire in $x^{(i)}$ through event e.

A synchronizing event e is *executable* in global state $\tilde{x}$, if $\forall i \in \eta^{(e)}$ the set of states $y^{(i)} \in succ_e(x^{(i)})$ is not empty.

Reachability function $\mathcal{F}$ is a functional element of $\mathcal{S} \rightarrow [0..1]$. $\mathcal{F}$ associates to each global state of $\mathcal{S}$ the value 1 if it is reachable, otherwise $\mathcal{F}$ associates the value 0.

Let

$\mathcal{R}$ subset of $\mathcal{S}$ which comprises all global states $\tilde{x}$ such that $\mathcal{F}(\tilde{x}) = 1$.

Definition 13 *A SAN model composed of N automata and E events is defined by: each automaton* $\mathcal{A}^{(i)}$ *(*$i \in [1..N]$*); each event* $e \in \mathcal{E}$*; and a reachability function* $\mathcal{F}$.

3.1.1 Well defined SAN

To compute the stationary solution of a SAN model, some proprieties are necessary, *e.g.*, liveness, irreducibility, ergodicity, *etc.* Some restrictions must be respected to ensure those properties. SAN models that obey these restrictions are called *well defined* SAN.

Restriction 1 *Automaton* $\mathcal{A}^{(i)}$ *is well defined, if and only if* $\forall \tilde{x} \in \mathcal{S}$, $\forall x^{(i)} \in \mathcal{S}^{(i)}$ *and* $\forall e \in \mathcal{E}$ *such that* $succ_e(x^{(i)}) \neq \emptyset$*:*

1.1. $\forall y^{(i)}, z^{(i)} \in succ_e(x^{(i)})$

$$\tau_e(x^{(i)}, y^{(i)})(\tilde{x}) = \tau_e(x^{(i)}, z^{(i)})(\tilde{x});$$

1.2. $$\left(\sum_{y^{(i)} \in succ_e(x^{(i)})} \pi_e(x^{(i)}, y^{(i)})(\tilde{x}) \right) = 1$$ *or a functional element equal to* 0 *or* 1.

Restriction 2 *An event* $e \in \mathcal{E}$ *is well defined, if and only if:*

2.1. $\forall \pi_1, \pi_2 \in [0,1]$,
$\forall e_1, e_2 \in \mathcal{E}$,
$\forall x^{(i)}, y^{(i)} \in \mathcal{S}^{(i)}$ *such that*
$y^{(i)} \in succ_e(x^{(i)})$ *and* $x^{(i)} \neq y^{(i)}$,
$\forall (e_1, \pi_1),\ (e_2, \pi_2) \in \mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$
$e_1 \neq e_2$

Restriction 3 *Reachability function $\mathcal{F}$ is well defined, if and only if the set of reachability states $\mathcal{R}$ is a strongly connected transition graph.*

Restriction 4 *A SAN is well defined, if and only if:*

4.1. all its automata are well defined;

4.2. all its events are well defined;

4.3. its reachability function is well defined.

3.2 SGSPN - Superposed Generalized Stochastic Petri Nets

Let
$\mathcal{GSPN}$ set of components $\mathcal{GSPN}^{(i)}$, where $i \in [1..N]$;
$\mathcal{T}_s$ set of synchronized transitions.

Definition 14 *Set $\mathcal{GSPN}$ comprises N components named $\mathcal{GSPN}^{(i)}$, where $i \in [1..N]$.*

Let
$\mathcal{C}^{(i)}$ set of conditions associated to transitions of $\mathcal{T}^{(i)}$.

Definition 15 *$c \in \mathcal{C}^{(i)}$ is a condition that may be associated to a transition $t \in \mathcal{T}^{(i)}$, which depends on tokens of $p \in \mathcal{P}^{(i)}$. This condition is a function with domain on tokens of $p \in \mathcal{P}^{(i)}$ and counter-domain on* false *and* true.

Definition 16 *A component $\mathcal{GSPN}^{(i)}$ is defined by tuple ($\mathcal{P}^{(i)}$, $\mathcal{T}^{(i)}$, $\pi^{(i)}$, $I^{(i)}$, $O^{(i)}$, $W^{(i)}$, $G^{(i)}$, $M_0^{(i)}$), where:*

16.1. $\mathcal{P}^{(i)}$ set of non-empty places;

16.2. $\mathcal{T}^{(i)}$ set of non-empty transitions;

16.3. $\pi^{(i)}$: $\mathcal{T}^{(i)} \rightarrow \{0,1\}$ priority function of the transitions;

16.4. $I^{(i)}$ and $O^{(i)}$: $\mathcal{T}^{(i)} \rightarrow \mathcal{P}^{(i)*}$ input and output functions of the transitions;

16.5. $W^{(i)}$: $\mathcal{T}^{(i)} \rightarrow \mathbb{R}^+$ function that assigns a rate to each transition;

16.6. $G^{(i)}$: $\mathcal{T}^{(i)} \rightarrow \mathcal{C}^{(i)}$ function, called guard, that associates a necessary, but not sufficient, condition $c \in \mathcal{C}^{(i)}$ to the firing of each transition $t \in \mathcal{T}^{(i)}$;

16.7. $M_0^{(i)}$: $\mathcal{P}^{(i)} \rightarrow \mathbb{N}$ initial marking in each place.

Note that a guard $g \in G^{(i)}$ only depends on markings of component $\mathcal{GSPN}^{(i)}$. There is no dependency on markings of others components $\mathcal{GSPN}$.

Definition 17 *Set of timed transitions $\mathcal{T}_T^{(i)}$ of component $\mathcal{GSPN}^{(i)}$ is defined as $\mathcal{T}_T^{(i)} = \{t \in \mathcal{T}^{(i)} \mid \pi^{(i)}(t) = 0\}$.*

Definition 18 *Set of immediate transitions* $\mathcal{T}_I^{(i)}$ *of component* $\mathcal{GSPN}^{(i)}$ *is defined as* $\mathcal{T}_I^{(i)} = \{t \in \mathcal{T}^{(i)} \mid \pi^{(i)}(t) = 1\}$.

Definition 19 *Set of transitions* $\mathcal{T}^{(i)}$ *of component* $\mathcal{GSPN}^{(i)}$ *is defined as* $\mathcal{T}^{(i)} = \mathcal{T}_T^{(i)} \cup \mathcal{T}_I^{(i)}$ *and* $\mathcal{T}_T^{(i)} \cap \mathcal{T}_I^{(i)} = \emptyset$.

Let

$\mathcal{T}_l^{(i)}$ set of local transitions of component $\mathcal{GSPN}^{(i)}$;

$\mathcal{T}_s^{(i)}$ set of synchronized transitions of component $\mathcal{GSPN}^{(i)}$.

Definition 20 *Transition* $t \in \mathcal{T}_l^{(i)}$ *is defined as local transition, if and only if* $\forall j \in [1..N]$ *such that* $i \neq j$, $t \notin \mathcal{T}_l^{(j)}$.

Definition 21 *Transition* t *is defined as synchronized transition, if and only if* $t \in \mathcal{T}_s^{(i)}$ *and* $\exists j \in [1..N]$ *such that* $i \neq j, t \in \mathcal{T}_s^{(j)}$.

Definition 22 *Set of synchronized transition* $\mathcal{T}_s$ *of a SGSPN model is defined as* $\mathcal{T}_s = \mathcal{T}_s^{(1)} \cup \mathcal{T}_s^{(2)} \cup \ldots \cup \mathcal{T}_s^{(N)}$.

Definition 23 *A SGSPN model composed of* N *components* $\mathcal{GSPN}^{(i)}$ *is defined by each component* $\mathcal{GSPN}^{(i)}$, *where* $i \in [1..N]$.

Any GSPN model may be viewed as a SGSPN model with only one component $\mathcal{GSPN}$ ($N = 1$). Hence, set of synchronized transitions is empty.

3.2.1 Well defined SGSPN

Alike to a *well defined* SAN model, some proprieties are also necessary to compute the stationary solution of a SGSPN model, *e.g.*, liveness, irreducibility, ergodicity, *etc.* Some restrictions must be respected to ensure those properties. SGSPN models that obey these restrictions are called *well defined* SGSPN.

Restriction 5 *A component* $\mathcal{GSPN}^{(i)}$ *is well defined, if and only if:*

5.1. if $N > 1$, *set of synchronized transitions* $\mathcal{T}_s^{(i)}$ *must not be empty, or if there is only one component* $\mathcal{GSPN}^{(i)}$ *in set of components* $\mathcal{GSPN}$ *(*$N = 1$*), set of synchronized transitions* $\mathcal{T}_s^{(i)}$ *must be empty;*

5.2. $\forall t \in \mathcal{T}^{(i)}$, $\exists p \in \mathcal{P}^{(i)}$ *such that* $p \in I^{(i)}(t)$;

5.3. $\forall t \in \mathcal{T}^{(i)}$, $\exists p \in \mathcal{P}^{(i)}$ *such that* $p \in O^{(i)}(t)$;

5.4. set of tangible markings is finite;

5.5. in all markings exists at least one transition enabled lead to a distinct marking.

Restriction 6 *Synchronized transition* $t \in \mathcal{T}_s$ *is well defined, if and only if:*

6.1. t *is a timed transition;*

6.2. $\forall i, j \in [1..N]$ *such that* $i \neq j$,

$$|I^{(i)}(t)| = |I^{(j)}(t)| \text{ and } |O^{(i)}(t)| = |O^{(j)}(t)|.$$

Restriction 7 *A SGSPN is well defined, if and only if:*

7.1. all its components $\mathcal{GSPN}^{(i)}$ *(*$i \in [1..N]$*) are well defined;*

7.2. all its synchronized transitions $t \in \mathcal{T}_s$ *are well defined.*

4 Representation Equivalence

In terms of infinitesimal generator description, the use of GTA operations is the main difference between SAN and SGSPN formalisms. Our first step is to show that any SAN model needing the use of GTA operators has at least one equivalent model that can be described only using CTA operators, *i.e.*, any SAN model with functional rates (or functional probabilities) can be represented by a SAN model with only constant rates[6], and vice-versa.

In Section 4.1, we demonstrate the equivalence between SAN (GTA) and SAN (CTA) models. Afterwards, in Section 4.2, we present the equivalence between SAN and SGSPN formalisms.

4.1 SAN (GTA) $\leftrightarrow$ SAN (CTA)

The proof idea for the equivalence between SAN models with and without functions is based on the substitution of functions by synchronizing events. In fact, each event with a functional rate (or functional probability) may be replaced by as many synchronizing events as its possible evaluations. The replacing synchronizing events must synchronize (often with *loop transitions*) all automata that the function depends.

Proposition 1 (SAN (GTA) $\rightarrow$ SAN (CTA)) *Any SAN (GTA) model has at least one equivalent SAN (CTA) model.*

Proof 1 *Any event which has a functional element associated to its occurrence rate may be represented through synchronizing events, i.e.:*

1. $\forall e \in \mathcal{E}, \forall \tilde{x}^{(\omega)}, \tilde{y}^{(\omega)} \in \mathcal{S}^{(\omega)}$ *such that it exists an event tuple* $(e, \tau(\mathcal{S}^{(\omega)}))$*, where* $(e, \tau(\tilde{x}^{(\omega)})) \neq (e, \tau(\tilde{y}^{(\omega)}))$*,*
2. $\forall i$ *such that* $i \in \eta^{(e)}$*,*
3. $\forall \mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$ *such that* $\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)}) \supset (e, \pi)$*, replace* (e, π) *by* $\mid \mathcal{S}^{(\omega - \eta^{(e)})} \mid$ *transition tuples* (e_j, π)*, where* $j \in [1.. \mid \mathcal{S}^{(\omega - \eta^{(e)})} \mid]$ *and*
4. $\forall \tilde{x}^{(\omega)} \in \mathcal{S}^{(\omega)}$, $\forall k \in (\omega - \eta^{(e)})$ *define an event tuple* $(e_j, \tau(\tilde{x}^{(\omega)}))$*, where* $x^{(k)}$ *is a part of* $\tilde{x}^{(\omega)}$*, and associate to* $\mathcal{Q}^{(k)}(x^{(k)}, x^{(k)})$ *a transition tuple* $(e_j, 1)$*.*

Any event which has a functional element associated to its probability may be represented through synchronizing events, i.e.:

1. $\forall e \in \mathcal{E}, \forall \tilde{x}^{(\omega)}, \tilde{y}^{(\omega)} \in \mathcal{S}^{(\omega)}$ *such that it exists an event tuple* $(e, \pi(\mathcal{S}^{(\omega)}))$*, where* $(e, \pi(\tilde{x}^{(\omega)})) \neq (e, \pi(\tilde{y}^{(\omega)}))$*,*
2. $\forall i$ *such that* $i \in \eta^{(e)}$*,*
3. $\forall \mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$ *such that* $\mathcal{Q}^{(i)}(x^{(i)}, y^{(i)}) \supset (e, \pi(\tilde{x}^{(\omega)}))$*, replace* $(e, \pi(\mathcal{S}^{(\omega)}))$ *by* $\mid \mathcal{S}^{(\omega - \eta^{(e)})} \mid$ *transition tuples* $(e_j, \pi(\tilde{x}^{(\omega)}))$*, where* $j \in [1.. \mid \mathcal{S}^{(\omega - \eta^{(e)})} \mid]$ *and*
4. $\forall \tilde{x}^{(\omega)} \in \mathcal{S}^{(\omega)}$, $\forall k \in (\omega - \eta^{(e)})$ *define an event tuple* $(e_j, \tau) = (e, \tau)$ *where* $x^{(k)}$ *is a part of* $\tilde{x}^{(\omega)}$*, and associate to* $\mathcal{Q}^{(k)}(x^{(k)}, x^{(k)})$ *a transition tuple* $(e_j, 1)$*.*

[6] GTA operators in the Markovian descriptor are used to represent the functional rates (or functional probabilities). Synchronizing events include new terms in the Markovian descriptor, but as long as there is no functional elements, they can be described using classical tensor products.

Proposition 2 (SAN (CTA) → SAN (GTA)) *Any SAN (CTA) model has at least one equivalent SAN (GTA) model.*

Proof 2 *Any occurrence rate of an event may be viewed as a constant functional element, i.e.:*

1. *$\forall e \in \mathcal{E}$ such that $\forall \tilde{x}, \tilde{y} \in \mathcal{S}$ does not exist an event tuple $(e, \tau(\mathcal{S}))$, where $(e, \tau(\tilde{x})) \neq (e, \tau(\tilde{y}))$.*

Any probability of an event may be viewed as a constant functional element, i.e.:

1. *$\forall e \in \mathcal{E}$ such that $\forall \tilde{x}, \tilde{y} \in \mathcal{S}$ does not exist a transition tuple $(e, \pi(\mathcal{S}))$, where $(e, \pi(\tilde{x})) \neq (e, \pi(\tilde{y}))$.*

Theorem 1 (SAN (GTA) ↔ SAN (CTA)) *Any SAN model needing the use of GTA operators has at least one equivalent model that can be described only using CTA operators, and vice-versa.*

Proof 3 *Directly by Proposition 1 and Proposition 2.*

4.2 SAN ↔ SGSPN

Since any SAN (GTA) model has at least one equivalent SAN (CTA) model (Theorem 1), it is possible to prove the equivalence between SAN and SGSPN formalisms. So any SAN model is viewed as a SAN (CTA) model. Figure 1 presents the equivalence diagram between SAN and SGSPN formalisms.

First of all, to prove equivalence between SAN and SGSPN formalisms, it is necessary to remind the classical definitions of *Stochastic State Machine* (SSM), *Reachability Set* (RS), *Tangible Reachability Set* (TRS), and *Tangible Reachability Graph* (TRG).

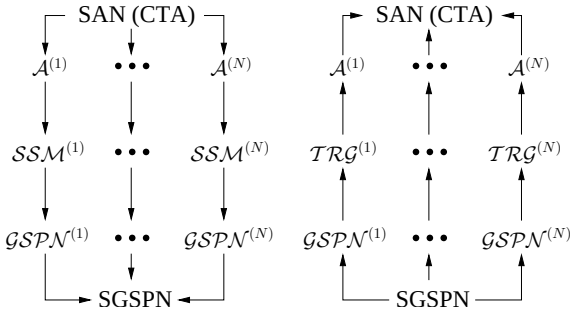


Figure 1: Equivalence diagram between SAN and SGSPN formalisms

Definition 24 *A Stochastic State Machine (SSM) is defined by tuple $(\mathcal{P}, \mathcal{T}, F, \Lambda)$, where:*

24.1. $\mathcal{P}$ set of non empty places;

24.2. $\mathcal{T}$ set of non empty transitions;

24.3. $F \subseteq ((\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}))$ with $dom(F) \cup codom(F) = \mathcal{P} \cup \mathcal{T}$ is the flow relation. It has to satisfy the following restriction[7]: $\forall t \in \mathcal{T}$: $|^\circ t| = |t^\circ| = 1$;

24.4. $\Lambda : \mathcal{T} \to \mathbb{R}^+$, where $\Lambda(t)$ is the rate of the exponential probability distribution associated to transition t.

[7] $|^\circ t|$ and $|t^\circ|$ indicate the number of input and output places of t.

Let

$M_k[t > M_l$ change from marking M_k to M_l due to the firing of t.

Definition 25 *Reachability Set $\mathcal{RS}^{(i)}(M_0^{(i)})$ of component $\mathcal{GSPN}^{(i)}$ is defined as the smallest set of markings, such that:*

25.1. $M_0^{(i)} \in \mathcal{RS}^{(i)}(M_0^{(i)})$

25.2. $M_k^{(i)} \in \mathcal{RS}^{(i)}(M_0^{(i)})$, *if and only if* $\exists t \in \mathcal{T}^{(i)}$ *such that* $M_k^{(i)}[t > M_l^{(i)} \rightarrow M_l^{(i)} \in \mathcal{RS}^{(i)}(M_0^{(i)})$

Definition 26 *Tangible Reachability Set $\mathcal{TRS}^{(i)}(M_0^{(i)})$ of component $\mathcal{GSPN}^{(i)}$ is composed of all tangible markings of $\mathcal{RS}^{(i)}(M_0^{(i)})$.*

Definition 27 *Tangible Reachability Graph $\mathcal{TRG}^{(i)}(M_0^{(i)})$ of component $\mathcal{GSPN}^{(i)}$ given an initial marking $M_0^{(i)}$ is a labelled directed multigraph whose set of nodes $\mathcal{TM}^{(i)}$ is composed of markings of Tangible Reachability Set $\mathcal{TRS}^{(i)}(M_0^{(i)})$ and whose set of arcs $\mathcal{TARC}^{(i)}$ is defined as follow:*

27.1. $\mathcal{TARC}^{(i)} \subseteq \mathcal{TRS}^{(i)}(M_0^{(i)}) \times \mathcal{TRS}^{(i)}(M_0^{(i)}) \times \mathcal{T}_T^{(i)} \times \mathcal{T}_I^{(i)*}$

27.2. $a_j^{(i)} = [M_k^{(i)}, M_l^{(i)}, t_0, \sigma] \in \mathcal{TARC}^{(i)}$, *if and only if* $M_k^{(i)}[t_0 > M_1^{(i)}, \sigma = t_1, \ldots, t_n, (n \geq 0)$ *and*

27.3. $\exists M_2^{(i)}, \ldots, M_n^{(i)}$ *such that* $M_1^{(i)}[t_1 > M_2^{(i)}[t_2 > \ldots M_n^{(i)}[t_n > M_l^{(i)}$

Proposition 3 (SAN → SGSPN) *Any SAN model has at least one equivalent SGSPN model.*

Proof 4 *A SAN model has N automata $\mathcal{A}^{(i)}$, where $i \in [1..N]$. Each automaton $\mathcal{A}^{(i)}$ has an equivalent stochastic state machine $\mathcal{SSM}^{(i)}$ such that:*

1. *Each $x^{(i)} \in \mathcal{S}^{(i)}$ corresponds to $p^{(i)} \in \mathcal{P}^{(i)}$;*
2. *Each transition tuple $(e, \pi) \subset Q^{(i)}(x^{(i)}, y^{(i)})$ corresponds to $|\mathcal{S}^{(\eta^{(e)})}|$ synchronized transitions $t_j^{(i)}$ $(j \in [1..|\mathcal{S}^{(\eta^{(e)})}|])$, if and only if exist $(p^{(i)}, t_j^{(i)}) \in F$ and $(t_j^{(i)}, q^{(i)}) \in F$ such that $x^{(i)}$ corresponds to $p^{(i)}$, $y^{(i)}$ corresponds to $q^{(i)}$, and $\Lambda(t_j^{(i)}) = \tau_e(\tilde{x}^{(\eta^{(e)})}, \tilde{y}^{(\eta^{(e)})}).\pi_e(\tilde{x}^{(\eta^{(e)})}, \tilde{y}^{(\eta^{(e)})})$;*
3. *There is only one token in any place $p^{(i)} \in \mathcal{P}^{(i)}$.*

Each $\mathcal{SSM}^{(i)}$ may be viewed as a component $\mathcal{GSPN}^{(i)}$ which has only timed transitions. SGSPN model is composed of all those components $\mathcal{GSPN}^{(i)}$ $(i \in [1..N])$.

Proposition 4 (SGSPN → SAN) *Any SGSPN model has at least one equivalent SAN model.*

Proof 5 *A SGSPN model has N components $\mathcal{GSPN}^{(i)}$, where $i \in [1..N]$. Each component $\mathcal{GSPN}^{(i)}$ has a tangible reachability graph $\mathcal{TRG}^{(i)}$ (Definition 27). Each tangible reachability graph $\mathcal{TRG}^{(i)}$ has an equivalent automaton $\mathcal{A}^{(i)}$ such that:*

1. *Each node $M_j^{(i)} \in \mathcal{TM}^{(i)}$ corresponds to $x^{(i)} \in \mathcal{S}^{(i)}$ of automaton $\mathcal{A}^{(i)}$;*
2. *Each $t^{(i)} \subset a_j^{(i)} \in \mathcal{TARC}^{(i)}$ corresponds to $(e, \pi) \subset \mathcal{Q}^{(i)}(x^{(i)}, y^{(i)})$, if and only if exist $[M_k^{(i)}, M_l^{(i)}, t, \sigma]$ such that $M_k^{(i)}$ corresponds to $x^{(i)}$, $M_l^{(i)}$ corresponds to $y^{(i)}$, $t \in \mathcal{T}_T^{(i)}$, $\sigma \in \mathcal{T}_I^{(i)*}$, $\tau_e(x^{(i)}, y^{(i)}) = W(t).W(\sigma)$[8], and $\pi_e(x^{(i)}, y^{(i)}) = 1$.*

[8]The computation of $W(\sigma)$ - the weight of a sequence of immediate transitions - can be found in [1].

Theorem 2 (SAN $\leftrightarrow$ SGSPN) *Any model described by SAN formalism has at least one equivalent model described by SGSPN formalism, and vice-versa.*

Proof 6 *Directly by Proposition 3 and Proposition 4.*

5 Modeling Examples

We now present five modeling examples. The first one presents a SAN and SGSPN model of an *Open Queueing Network with Loss and Blocking*. The second one, modeled by SAN (CTA and GTA) and SGSPN, describes a *Resource Sharing* model. The third one shows a SAN (CTA and GTA) and SGSPN model of an *Alternate Service Pattern*. The fourth one also presents a SAN (CTA and GTA) and SGSPN model of a *First Available Server* queueing. And the last one shows a SAN and SGSPN model of a queueing network with *Limited Capacity Subnets*.

It is important to notice that SAN (CTAand GTA) and SGSPN models presented in this section, even though equivalents, were not obtained using the conversion steps presented in Figure 1. In fact, the equivalence diagram is used to prove the existence of an equivalent SGSPN model to each and every SAN model, and vice-versa. However, the models obtained in such way are often not compact or readable ones. An human-made conversion may provide better (more compact) models. The models presented in this section were converted using our best knowledge on the formalisms. Therefore, a more skilled modeler may develop even better models.

5.1 OQN - Open Queueing Network

Figure 2 is an example of an Open Queueing Network with three queues and only one single class of customers.

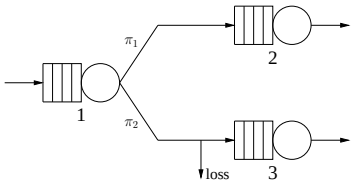


Figure 2: OQN with Loss and Blocking

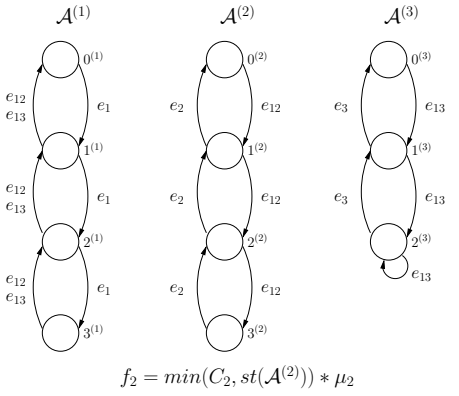
This model has one queue with a *blocking* behavior (for customers from queue 1 to queue 2) and another queue with a *loss* behavior (for customers from queue 1 to queue 3). The queue capacities are respectively 3, 3 and 2 customers.

Figure 3 is an equivalent SAN (CTA) model to this *Open Queueing Network*. Each automaton $\mathcal{A}^{(i)}$ represents a queue. States $x^{(i)}$ of an automaton $\mathcal{A}^{(i)}$ indicate the number of customers in queue i.

The arrival of customers in queue 1, and the departure of customers from queue 2 and queue 3 are represented by local events e_1, e_2 and e_3 respectively. Synchronizing events e_{12} and e_{13} respectively represent the routing of customers from queue 1 to queue 2 and from queue 1 to queue 3.

Local event e_2 has a functional rate[9] defined by f_2, which it is dependent on the number of customers in automaton $\mathcal{A}^{(2)}$, since queue 2 has two servers. C_2 represents the number of servers in queue 2.

[9]Syntax used by PEPS2003 software tool [6].

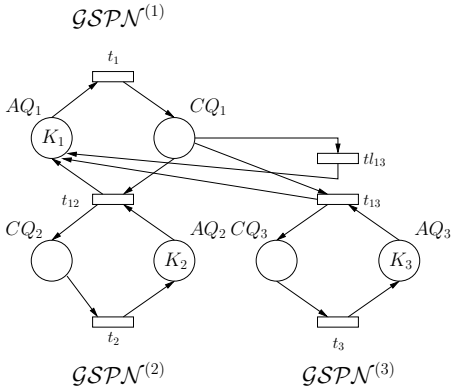


Event	Rate
e_1	λ_1
e_2	f_2
e_3	μ_3
e_{12}	$\mu_1\pi_1$
e_{13}	$\mu_1\pi_2$

Figure 3: OQN - SAN (CTA)

The occurrence of event e_{12} forces the simultaneous change of automata $A^{(1)}$ and $A^{(2)}$. Synchronizing event e_{12} can not occur when automaton $A^{(2)}$ is in local state $3^{(2)}$ (queue 2 is full). Hence, it corresponds to the *blocking* of the departure of customers from queue 1 to queue 2. An extra loop transition in the last state of automaton $A^{(3)}$ (event e_{13}) allows the departure of customers from queue 1 without an arrival in queue 3, *i.e.*, the *loss* of customers in queue 3 (queue 3 is full).

The SAN (CTA) model for this example does not require the use of GTA, since the functional element f_2 depends only on the state of the automaton where it appears. Figure 4 presents an equivalent SGSPN model to SAN (CTA) model of Figure 3.



Transition	Rate
t_1	λ_1
t_2	$\mu_2 * min(tk(CQ_2), C_2)$
t_3	μ_3
t_{12}	$\mu_1\pi_1$
t_{13}	$\mu_1\pi_2$
tl_{13}	$\mu_1\pi_2$

Guards
$gl_{13} = (tk(AQ_3) == 0)$

Figure 4: OQN - SGSPN

Each queue i is equivalent to a pair of places: CQ_i and AQ_i (component $\mathcal{GSPN}^{(i)}$). Place AQ_i represents the available room for customers in queue i (capacity), and place CQ_i represents the number of customers currently in queue i. This is an equivalent SGSPN model to SAN (CTA) model described previously. So K_1, K_2 and K_3 represent the queue capacities.

Transition t_1 represents the arrival in queue 1, as well as transition t_3 represents the departure from queue 3. Alike local event e_2 in SAN (CTA) model, transition t_2 has a funcional rate[10] which it is dependent on number of customers in queue 2, since queue 2 has two servers.

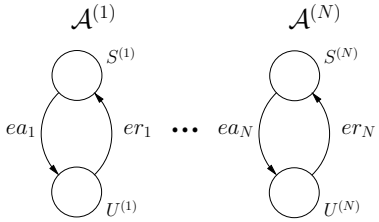
Synchronized transition t_{12} indicates the routing of customers from queue 1 to queue 2. Analogously, t_{13} represents the routing of customers from queue 1 to queue 3. Transition tl_{13} indicates the loss of customers from queue 1 and its rate is also equal to $\mu_1\pi_2$, but this transition

[10]Syntax used by SMART software tool [10].

can only be fired if the third queue is full. Thus, a *guard* with condition $tk(A_3) == 0$ (the queue is full, since there is no tokens in the *available* A_3 place) must be assigned to transition tl_{13}.

5.2 RS - Resource Sharing

Figure 5 shows a SAN (GTA) model of a *Resource Sharing* system. In this model, there are N process sharing R resources. Each process is represent by an automaton $\mathcal{A}^{(i)}$, which has two states: $S^{(i)}$ (*sleeping*) and $U^{(i)}$ (*using*). Function f_{u_i} is defined to control the number of process which use the resources. Each local event ea_i has its rate multiplied by function f_{u_i} so as to control the number of available resources. Thus, local events ea_i only happen when there are available resources ($nb\ [\mathcal{A}^{(1)}..\mathcal{A}^{(N)}]\ U < R$).

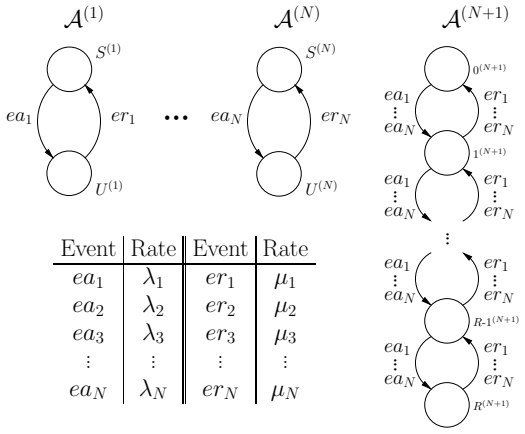


Event	Rate	Event	Rate
ea_1	f_{u_1}	er_1	μ_1
ea_2	f_{u_2}	er_2	μ_2
ea_3	f_{u_3}	er_3	μ_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_N	f_{u_N}	er_N	μ_N

$$f_{u_i} = (nb\ [\mathcal{A}^{(1)}..\mathcal{A}^{(N)}]\ U < R) * \lambda_i$$

Figure 5: RS - SAN (GTA)

As we demonstrated in Section 4.1, any SAN (GTA) model has an equivalent SAN (CTA) model. Figure 6 represents an equivalent SAN (CTA) model to Figure 5. In this model, resource pool is represent by automaton $\mathcal{A}^{(N+1)}$ and it has $R+1$ states, which indicate the number of resources in use. Events ea_i and er_i are synchronizing events among process and resource pool ($\mathcal{A}^{(N+1)}$).



Event	Rate	Event	Rate
ea_1	λ_1	er_1	μ_1
ea_2	λ_2	er_2	μ_2
ea_3	λ_3	er_3	μ_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_N	λ_N	er_N	μ_N

Figure 6: RS - SAN (CTA)

Figure 7 presents an equivalent SGSPN model to Figure 6. Each process ($\mathcal{GSPN}^{(i=1..N)}$) has two places: S_i (*sleeping*) and U_i (*using*). In component $\mathcal{GSPN}^{(N+1)}$, the tokens in RS place represent the number of available resources, whereas they represent the number of using resources in RU place.

Transition ta_i and tr_i are synchronized transitions among each process and the resource pool ($\mathcal{GSPN}^{(N+1)}$). Synchronized transitions ta_i and tr_i have the same rates used by synchronizing events ea_i and er_i of Figure 6.

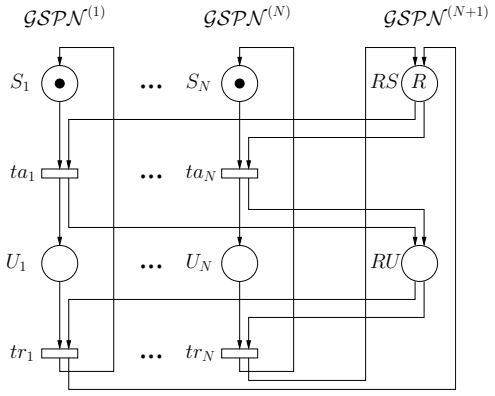


Figure 7: RS - SGSPN

5.3 ASP - Alternate Service Pattern

This model describes a small open network composed of four queues (Q_1, Q_2, Q_3 and Q_4) with finite capacities K_1, K_2, K_3 and K_4 respectively. Figure 8 describes the routing pattern of customers in this model, where the customers arrive in $Q1$ and Q_2 with constant rates λ_1 and λ_2 respectively. Customers may leave from Q_1 to Q_3, if and only if there is room in that queue (blocking behavior), whereas customers may leave from Q_2 to Q_3 whether there is room in that queue or leave the model otherwise (loss behavior). Customers may also leave from Q_3 to Q_4 with blocking behavior.

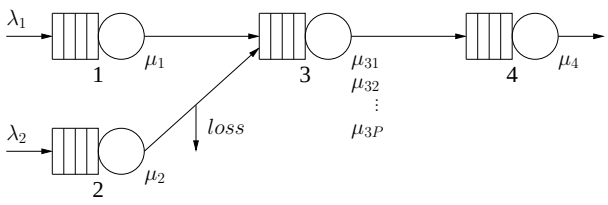
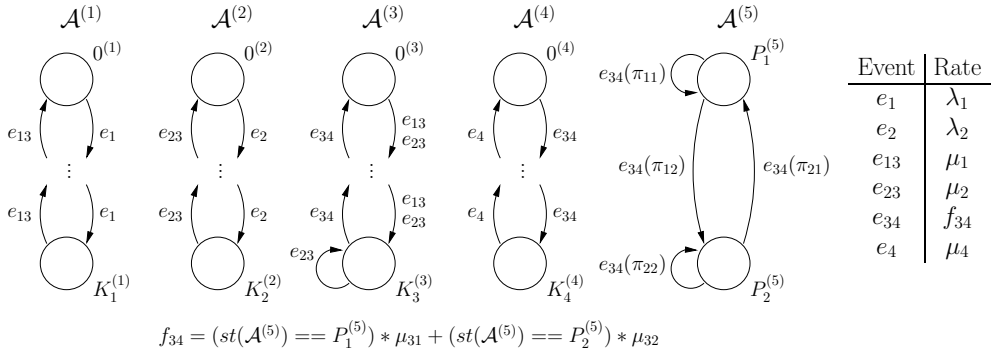


Figure 8: ASP model

While Q_1, Q_2 and Q_4 have standard (single) service behavior, *i.e.*, a same average service rate for all customers (respectively (μ_1, μ_2 and μ_4), queue Q_3 has an *Alternate Service Pattern* behavior. The service rate of this queue varies according to P different service patterns ($\mu_{31}, \ldots, \mu_{3P}$). Q_3 can exchange its service pattern simultaneously to the end of service of a customer. Therefore, when a customer is served by service pattern P_i, Q_3 can remain serving the next customer in the same pattern with a given probability π_{ii}, or it can alternate to other service pattern P_j, with a given probability π_{ij} (for all service patterns P_i: $\sum_{j=1}^{P} \pi_{ij} = 1$).

SAN model for this example is composed of one automaton to each single-service pattern queue (Q_1, Q_2 and Q_4) and two automata for the alternate service pattern queue (Q_3). Let us call $\mathcal{A}^{(i)}$ ($i = 1, \ldots, 4$) the automaton representing the number of customers in Q_i, and $\mathcal{A}^{(5)}$ the automaton representing the current service pattern in Q_3. Local events e_1 and e_2 represent the arrival in queues Q_1 and Q_2 respectively, and local event e_4 represents the departure from Q_4. Synchronizing events e_{13} and e_{34} represent the routings between queues Q_1 to Q_3 and Q_3 to Q_4, respectively, and synchronizing event e_{23} represents both the routing from Q_2 to Q_3 and the departure from Q_2 due to lack of room in Q_3 (loss). All events, besides e_{34}, have constant rates according to the model definition. The functional rate of event e_{34} is defined by the function f_{34} and it depends on the state of automaton $\mathcal{A}^{(5)}$.

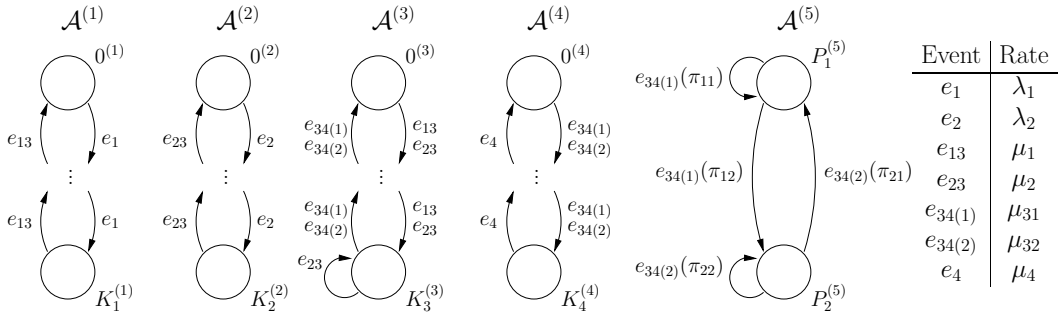
Figure 9 presents the SAN (GTA) model for this example considering $P = 2$, the extension to a higher number of service patterns will correspond to the addition of more local states to automaton $\mathcal{A}^{(5)}$, which will always have P local states. Note that event e_{34} obviously synchronizes automata $\mathcal{A}^{(3)}$ and $\mathcal{A}^{(4)}$, but it also synchronizes automaton $\mathcal{A}^{(5)}$ due to the (possible) exchange of service pattern. The use of probabilities to event e_{34} represents the stochastic distribution of all the possible exchanges between the service patterns.



Event	Rate
e_1	λ_1
e_2	λ_2
e_{13}	μ_1
e_{23}	μ_2
e_{34}	f_{34}
e_4	μ_4

Figure 9: ASP - SAN (GTA)

Figure 10 presents an equivalent SAN (CTA) model to SAN (GTA) model of Figure 9. Note that there are two synchronizing events ($e_{34(1)}$ and $e_{34(2)}$) to model the change between the possible service patterns. Events $e_{34(1)}$ and $e_{34(2)}$ have constant rates equal to μ_{31} and μ_{32} respectively. In fact, a SAN (CTA) model to an example with P service patterns will have P synchronizing events $e_{34(1)}, e_{34(2)}, \ldots, e_{34(P)}$, each of them with constant rate $\mu_{31}, \mu_{32}, \ldots, \mu_{3P}$ respectively.



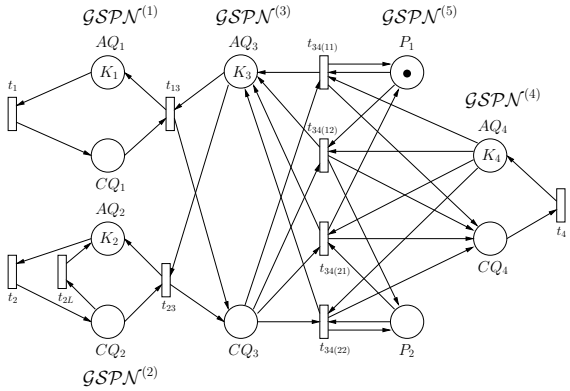
Event	Rate
e_1	λ_1
e_2	λ_2
e_{13}	μ_1
e_{23}	μ_2
$e_{34(1)}$	μ_{31}
$e_{34(2)}$	μ_{32}
e_4	μ_4

Figure 10: ASP - SAN (CTA)

The equivalent SGSPN model is presented in Figure 11 (considering $P = 2$). This model defines a pair of places to represent each queue number of customers, one place (CQ_i) to represent the number of customers currently in the queue Q_i, and another place (AQ_i) to represent the available room in this queue. Therefore, the number of tokens in each pair of CQ_i and AQ_i places must always sum the capacity (K_i) of queue Q_i. Places P_1 and P_2 indicate which service pattern is being used in the Q_3. Component $\mathcal{GSPN}^{(i)}$ ($i = 1, \ldots, 4$) is composed of AQ_i and CQ_i places, which represents the Q_i. Component $\mathcal{GSPN}^{(5)}$ is composed of P_1 and P_2 places (considering $P = 2$), which represents the set of possible service patterns.

The most significative differences between the events of the SAN model and the transitions of the SGSPN model concern the end of service in Q_2 and in Q_3. The end of service in Q_2 may lead to the routing from Q_2 to Q_3 (t_{23}) which occurs only when Q_3 has room for another

customer, or to the departure (loss) of Q_2 (t_{2L}) which occurs when Q_3 is full. The end of service in Q_3 always represents the routing of one customer from Q_3 to Q_4, but it may occur simultaneously to the change of service pattern. Therefore, P^2 transitions are used to represent this routing considering all possible changes and permaneces in the service pattern. Generically, transition $t_{34(ij)}$ represents the end of service in Q_3 with service pattern P_i and changing to service pattern to P_j (when $i \neq j$) or staying in the same service pattern (when $i = j$).



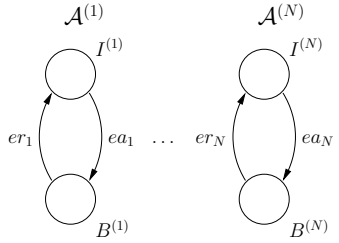
Transition	Rate
t_1	λ_1
t_2	λ_2
t_{13}	μ_1
t_{23}	μ_2
t_{2L}	μ_2
$t_{34(11)}$	$\mu_{31}\pi_{11}$
$t_{34(12)}$	$\mu_{31}\pi_{12}$
$t_{34(21)}$	$\mu_{32}\pi_{21}$
$t_{34(22)}$	$\mu_{32}\pi_{22}$
t_4	μ_4

Guards
$g_{2L} = (tk(CQ_3) == K_3)$

Figure 11: ASP - SGSPN

5.4 FAS - First Available Server

This example considers a single queue with common exponential arrival (with rate λ) and a finite number (N) of distinguishable and ordered servers. As a client arrives, it is served by the *First Available Server* with a rate μ regardless of which server was used. SAN (GTA) model for this example (Figure 12) has N automata ($\mathcal{A}^{(i)}$). The i-th server is represented by two-state automaton with the local states: *Idle* ($I^{(i)}$) and *Busy* ($B^{(i)}$). The arrival in the i-th server is expressed by a local event (called ea_i) with a functional rate equal to λ, if all preceding servers are busy, or zero otherwise. Consequently, at any global state of the model only one server, the first available, has a nonzero arrival rate. Local event er_i (with constant rate μ) represents the end of service in i-th server.

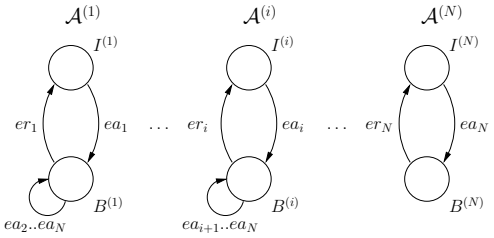


Event	Rate	Event	Rate
ea_1	λ	er_1	μ
ea_2	f_2	er_2	μ
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_N	f_N	er_N	μ

Figure 12: FAS - SAN (GTA)

SAN (CTA) model has also one automaton to each server. However, all events ea_i became synchronizing events. Each synchronizing event ea_i has a constant rate equal to λ and it can only occur when all precedent servers (automata) are busy (state $B^{(i)}$).

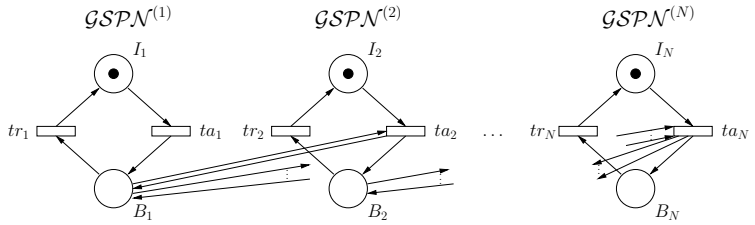
The equivalent SGSPN model is composed of N pairs (components $\mathcal{GSPN}$), each of them representing one of the servers. The i-th server is represented by I_i and B_i places, and only one of those two places may have a token at time.



Event	Rate	Event	Rate
ea_1	λ	er_1	μ
ea_2	λ	er_2	μ
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_N	λ	er_N	μ

Figure 13: FAS - SAN (CTA)

The transition ta_i represents the passage of the token from I_i to B_i place, and it means the i-th server becomes busy. Such transition must only occur if all precedent servers are busy, *i.e.*, all places B_j ($j < i$) have tokens. The transitions tr_i means the end of service in the i-th server and it may occur independently of the state of the other servers. Note that this model may not use the guards to avoid undesirable ta_i transitions because the resulting model will not be a valid SGSPN (it would be a disconnected net).



Transition	Rate	Transition	Rate
ta_1	λ	tr_1	μ
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ta_N	λ	tr_N	μ

Figure 14: FAS - SGSPN

5.5 LCS - Limited Capacity Subnets

This model describes a small open network composed of five queues (Q_1, Q_2, Q_3, Q_4 and Q_5) with finite capacities K_1, K_2, K_3 , K_4 and K_5 respectively. Figure 15 describes the routing pattern of customers in this model.

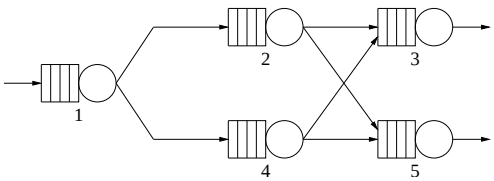
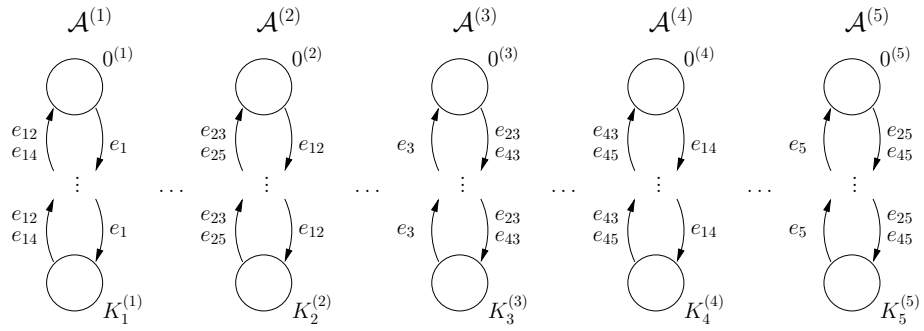


Figure 15: LCS model

All routing patterns assume blocking behavior and all queues have a single service pattern. The particularity of this example is the *Limited Capacity of Subnets* represented by the pairs Q_2-Q_3 and Q_4-Q_5. In fact, we consider that the total number of customers in queues Q_2 and Q_3 must not exceed a threshold Th_{23} and, analogously, for the pair Q_4-Q_5 the number of customers must not exceed Th_{45}.

The SAN (GTA) model (Figure 16) follows the same principle of the previous example using local events to represent arrivals and departures to/from the network and synchronizing events to represent the routing of customers between two queues. The main difference will appear in the rates of the events corresponding to the enter of a customer in a subnet. Such rates must be functions of the state of the pair of queues to represent the subnet.



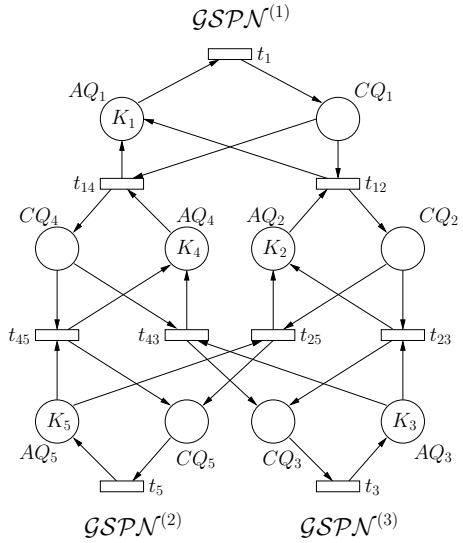
$f_1 = (st(\mathcal{A}^{(2)}) + st(\mathcal{A}^{(3)})) < Th_{23}$

$f_2 = (st(\mathcal{A}^{(4)}) + st(\mathcal{A}^{(5)})) < Th_{45}$

Event	Rate	Event	Rate	Event	Rate
e_1	λ_1	e_{23}	$\mu_2\pi_3$	e_{43}	$\mu_4\pi_5 f_1$
e_{12}	$\mu_1\pi_1 f_1$	e_{25}	$\mu_2\pi_4 f_2$	e_{45}	$\mu_4\pi_6$
e_{14}	$\mu_1\pi_2 f_2$	e_3	μ_3	e_5	μ_5

Figure 16: LCS - SAN (GTA)

Analogously, the equivalent SGSPN model (Figure 17) follows the same principle as before (a pair of places for each queue), but it uses guards to avoid the undesirable transitions, *i.e.*, transitions that lead to states where the thresholds are not respected. In this example, there are only three components $\mathcal{GSPN}$. Each subnet is represented by a component $\mathcal{GSPN}$, *i.e.*, component $\mathcal{GSPN}^{(1)}$ is composed of queue 1, component $\mathcal{GSPN}^{(2)}$ is composed of queues 2 and 3, whereas component $\mathcal{GSPN}^{(3)}$ is composed of queues 4 and 5.



Transition	Rate
t_1	λ_1
t_{12}	$\mu_1\pi_1$
t_{14}	$\mu_1\pi_2$
t_{23}	$\mu_2\pi_3$
t_{25}	$\mu_2\pi_4$
t_3	μ_3
t_{43}	$\mu_4\pi_5$
t_{45}	$\mu_4\pi_6$
t_5	μ_5

Guards
$g_{12} = (tk(CQ_2) + tk(CQ_3)) < Th_{23}$
$g_{14} = (tk(CQ_4) + tk(CQ_5)) < Th_{45}$
$g_{25} = (tk(CQ_4) + tk(CQ_5)) < Th_{45}$
$g_{43} = (tk(CQ_2) + tk(CQ_3)) < Th_{23}$

Figure 17: LCS - SGSPN

6 Theoretical Comparison

In this section, we show the required computational cost to achieve a transient or stationary solution of a SAN and a SGSPN model expressed in a tensor format. SAN and SGSPN are formalisms based on Kronecker representations. Both formalisms have a tensor representation of theirs infinitesimal generators and they are represented by a *Markovian Descriptor*.

Markovian Descriptor is an algebraic formula which allows to store in a compact form the infinitesimal generator of the Markov chain equivalent to a SAN model using a mathematical formula [14, 19, 4]:

$$Q = \bigoplus_{i=1}^{N}{}_g \, Q_l^{(i)} + \sum_{e \in \mathcal{E}_s} \left(\bigotimes_{i=1}^{N}{}_g \, Q_{e^+}^{(i)} + \bigotimes_{i=1}^{N}{}_g \, Q_{e^-}^{(i)} \right) \tag{1}$$

Fernandes, Plateau and Stewart [14] present the computational cost (number of multiplications) to evaluate the product of a probability vector by a *tensor product* or *tensor sum* of N matrices[11]:

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i, \tag{2}$$

where n_i is the order of the i^{th} matrix. This equation is valid if all matrices are full. However, *Markovian Descriptor* may have sparse matrices and so the computational cost is given by:

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}, \tag{3}$$

where nz_i represents the number of nonzero elements of the i^{th} matrix.

A SAN model has N automata $\mathcal{A}^{(i)}$, where $i \in [1..N]$. In the equation (1), there is one tensor sum (local events) $\bigoplus_{i=1}^{N} Q_l^{(i)}$ and 2 $|\mathcal{E}_s|$ tensor products (synchronizing events) to each automaton $\mathcal{A}^{(i)}$ ($i \in [1..N]$) of the model.

We now utilize equation (3) on *Markovian Descriptor* of a SAN model - equation (1) - to evaluate its computational cost, since it uses tensors on its solution method.

So the computational cost of the *Markovian Descriptor* - equation (1) - of a SAN model is given by:

$$\sum_{k=1}^{1+2|\mathcal{E}_s|} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i^{(k)}}{n_i} \right), \tag{4}$$

where $nz_i^{(k)}$ represents the number of nonzero elements of the k^{th} tensor as follows:

$$nz_i^{(k)} = \begin{cases} Q_l^{(i)} & \text{if } k = 1 \\ Q_{e^+_{(k-1)}}^{(i)} & \text{if } 1 < k \leq (1+|\mathcal{E}_s|) \\ Q_{e^-_{(k-(1+|\mathcal{E}_s|))}}^{(i)} & \text{if } k > (1+|\mathcal{E}_s|) \end{cases}$$

The order of matrix n_i is equal for all k terms. Then it is possible to factor the equation (4):

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \left(\frac{\sum_{k=1}^{1+2|\mathcal{E}_s|} nz_i^{(k)}}{n_i} \right) \tag{5}$$

[11] The vector-descriptor multiplication is the basic operation for most of the iterative solutions, *e.g.*, Power method, Uniformization [21].

Thus equation (5) represents the computational cost to evaluate the *Markovian Descriptor* of a SAN model.

Markovian Descriptor of a SGSPN model uses tensors obtained from *Stochastic Automata*. Components $\mathcal{GSPN}^{(i)}$ interact through synchronized transitions, as well as automata $\mathcal{A}^{(i)}$ interact through synchronizing events.

So all component $\mathcal{GSPN}^{(i)}$ has a $\mathcal{TRG}^{(i)}(M_0^{(i)})$ which is viewed as an automaton $\mathcal{A}^{(i)}$, as well as the synchronized transitions are equivalent to synchronizing events. Hence the computational cost of a SGSPN model with N components $\mathcal{GSPN}^{(i)}$ and $|\mathcal{T}_s|$ synchronized transitions is:

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \left(\frac{\sum_{k=1}^{1+2|\mathcal{T}_s|} nz_i^{(k)}}{n_i} \right) \tag{6}$$

where n_i is the number of tangible markings in the component $\mathcal{GSPN}^{(i)}$ and $nz_i^{(k)}$ is the number of nonzero elements of the i^{th} matrix of the k^{th} term.

Although the equations to computational cost of both formalisms are quite similar, note that the *number of synchronizing events* in a SAN model is not necessarily equal to the *number of synchronized transitions* in an equivalent SGSPN model.

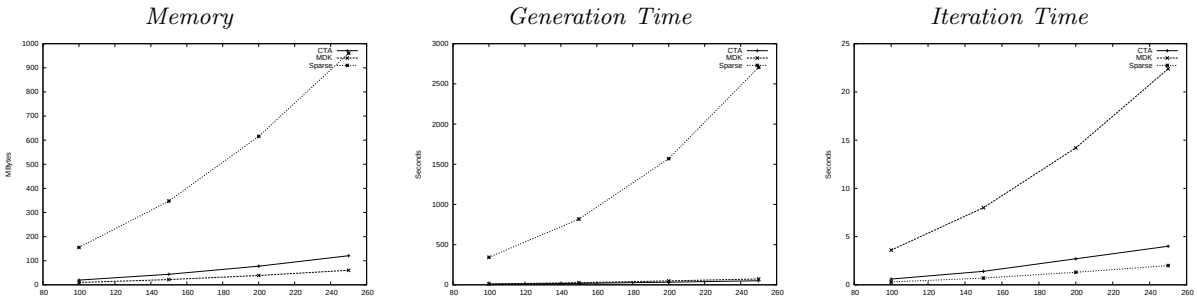
7 Practical Comparison

We present, in this section, the results achieved in the comparisons between the memory needs and CPU time to the generation, and the solution of the models presented in Section 5.

The numerical results were solved using *Power* method for all examples. These results were obtained on a 2.8 GHz Pentium IV Xeon under Linux operating system with 2 GBytes of memory. To each example, we solve the SAN (GTA) and SAN (CTA) models using PEPS software tool (version 1.1) [6]. The SGSPN model was solved with SMART software tool (version 2003) [10] using MxD technique with Kronecker storage (**MDK**) and generation of a sparse matrix (**Sparse**). The table columns indicate the **size** of infinitesimal generator (in MBytes), CPU time to the generation of the infinitesimal generator (**gen.**), and the CPU time to perform one single power iteration (**iter.**) in seconds. Note that the actual solution of a model will demand a different number of iterations according to the required precision and the chosen parameters. Finally, in the bottom of each table, a row indicates the reachable state space (rss). Although the product state space may be different to models, the reachable state space is the same regardless the modeling formalism (SAN or SGSPN) or representation technique (GTA, CTA, MDK, or Sparse).

7.1 OQN - Open Queueing Network

Figure 18 presents the results of OQN models with Queue 1 capacity (K_1) equal to 250, and Queue 2 and 3 capacities (K_2 and K_3) equals to 100, 150, 200, and 250. It is important to notice that SAN model for this example does not require the use of GTA. In all those examples, SAN (CTA) models were faster (generation and solution time) than SGSPN-MDK models, which also is described only with CTA operators. However, SGSPN-MDK model obtains a quite efficient Kronecker structure. SGSPN-Sparse solution has a smaller iteration time, but both generation time and memory needs are considerably larger.

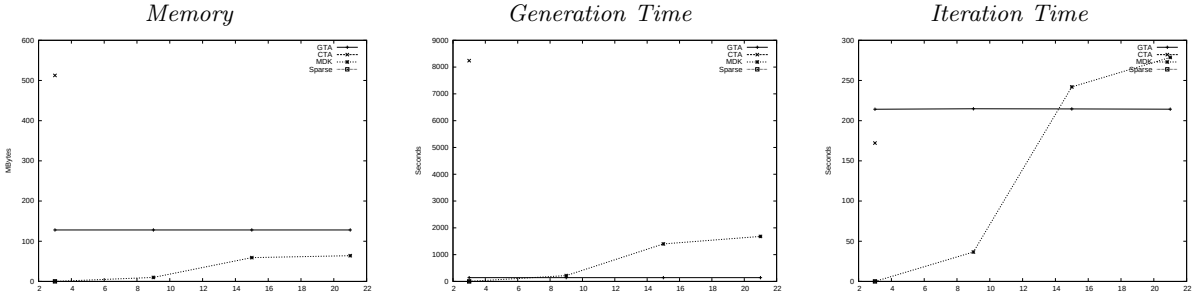


OQN - Open Queueing Network													
K_1 - K_2 - K_3		250 - 100 - 100			250 - 150 - 150			250 - 200 - 200			250 - 250 - 250		
		size	gen.	iter.	size	gen.	iter.	size	gen.	iter.	size	gen.	iter.
SAN	CTA	19.56MB	8.2s	0.6s	43.69MB	18.5s	1.4s	77.40MB	32.7s	2.7s	120.68MB	51.1s	4.0s
SGSPN	MDK	9.80MB	10.8s	3.6s	21.86MB	27.0s	8.0s	38.72MB	49.0s	14.2s	60.37MB	73.0s	22.4s
	Sparse	155.06MB	342.2s	0.3s	347.26MB	817.8s	0.7s	615.82MB	1569.6s	1.3s	960.84MB	2706.9s	2.0s
rss		2,560,451 states			5,723,051 states			10,140,651 states			15,813,251 states		

Figure 18: Results for Open Queueing Network (OQN) models

7.2 RS - Resource Sharing

RS models were solved with 24 processes (N) and 3, 9, 15 and 21 resources (R). As may be observed in Figure 19, SAN (CTA) and SGSPN-Sparse models could only be solved for 3 resources. The solution of models using SGSPN approach is really fast considering a relative small number of reachable states. But, as the number of resources grows, *i.e.*, with fewer unreachable states, the generation and solution times grow considerably, whereas they remain the same for SAN (GTA) model.

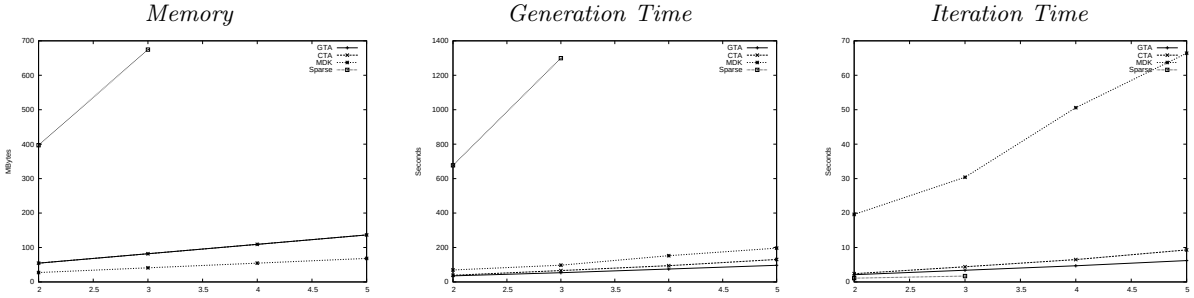


RS - Resource Sharing ($N = 24$)													
R		3			9			15			21		
		size	gen.	iter.	size	gen.	iter.	size	gen.	iter.	size	gen.	iter.
SAN	GTA	128.02MB	140.7s	214.2s	128.02MB	140.3s	214.8s	128.02MB	140.5s	214.6s	128.02MB	140.6s	214.3s
	CTA	512.55MB	8238.8s	172.2s	-	-	-	-	-	-	-	-	-
SGSPN	MDK	0.05MB	0.2s	$\approx 0.0s$	9.92MB	217.4s	36.6s	59.23MB	1400.0s	242.0s	64.05MB	1679.5s	278.7s
	Sparse	0.16MB	0.5s	$\approx 0.0s$	-	-	-	-	-	-	-	-	-
rss		2,325 states			2,579,130 states			15,505,590 states			16,776,915 states		

Figure 19: Results for Resource Sharing (RS) models

7.3 ASP - Alternate Service Pattern

Figure 20 shows the results of ASP models with queue capacities equals to 30, 30, 60 and 60 (K_1, K_2, K_3, K_4), and 2, 3, 4 and 5 service patterns (P). SGSPN-Sparse models have better results for the solution times, but this approach can not be used for larger models ($P = 4$ and $P = 5$) due to the huge amount of memory needed to handle the generator matrix. The memory used in the SGSPN-MDK models is quite smaller than the memory used in the SAN (GTA and CTA) models. However, since SAN (GTA) allows to use a small number of synchronizing primitives (synchronizing events in SAN and synchronized transitions in SGSPN), it can obtain smaller generation and solution times. It becomes evident to models with a large number of service patterns (P). Moreover, for models that converge in a small number of iterations (*e.g.* 500 iterations) the overall SAN (GTA) solution (generation plus iterations) could be faster than SGSPN sparse solution. It is also remarkable the specific gains of the use of GTA when comparing the SAN (GTA and CTA) solutions.



ASP - Alternate Service Pattern ($K_1 = 30$; $K_2 = 30$; $K_3 = 60$; $K_4 = 60$)													
P		2			3			4			5		
		size	gen.	iter.	size	gen.	iter.	size	gen.	iter.	size	gen.	iter.
SAN	GTA	54.54MB	35.5s	2.1s	81.86MB	53.7s	3.4s	109.14MB	74.9s	4.7s	136.42MB	96.7s	6.2s
	CTA	54.58MB	38.8s	2.4s	81.86MB	66.1s	4.4s	109.14MB	94.5s	6.5s	136.42MB	130.5s	9.3s
SGSPN	MDK	27.30MB	69.4s	19.6s	40.95MB	97.4s	30.4s	54.60MB	152.4s	50.6s	68.24MB	196.4s	66.4s
	Sparse	396.88MB	677.2s	1.1s	674.50MB	1,299.5s	1.7s	-	-	-	-	-	-
rss		7,151,762 states			10,727,643 states			14,303,524 states			17,879,405 states		

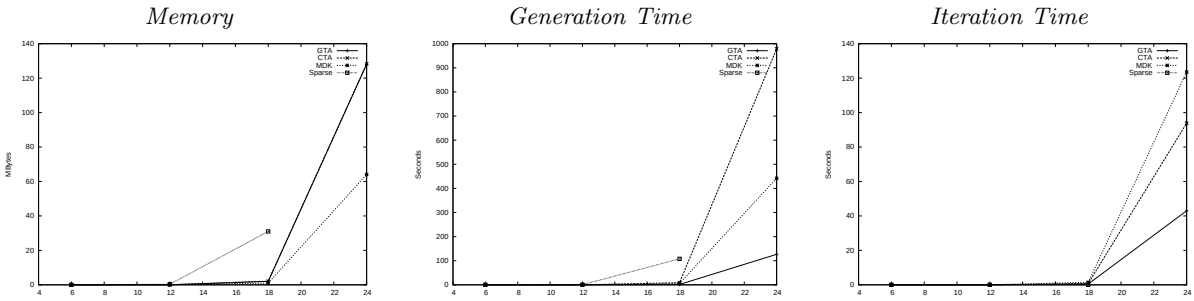
Figure 20: Results for Alternate Service Pattern (ASP) models

7.4 FAS - First Available Server

Figure 21 presents the results of FAS models with 6, 12, 18, and 24 servers (N). Alike ASP models, SGSPN-Sparse models also have better results for the solution times, however sparse approach can not be useful to large models due to the huge amount of memory, *e.g.*, $N = 24$. For 6 and 12 servers, the generation and solution times are insignificant to all approaches. But for 18 and 24 servers, SAN (GTA) models have irrefutable advantages due to the use of functional rates.

7.5 LCS - Limited Capacity Subnets

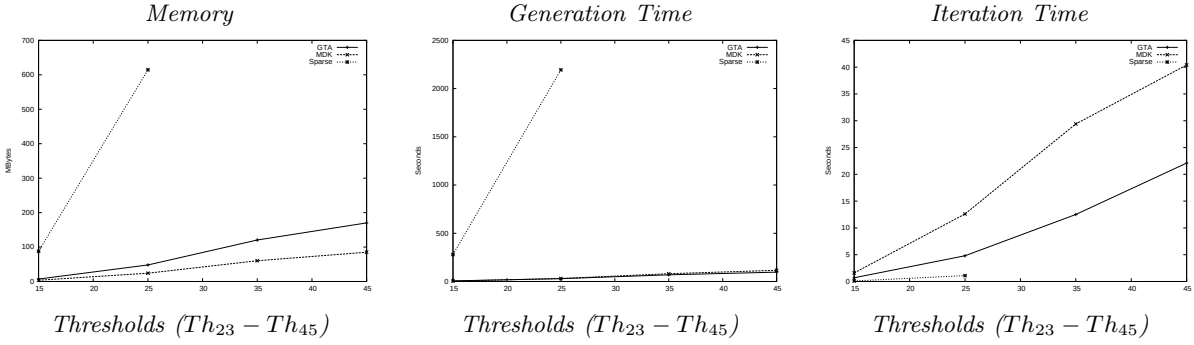
Figure 21 presents the results of LCS models with queue capacities equals to 50, 25, 25 and 25 (K_1, K_2, K_3, K_4) respectively, and thresholds equals to 15, 25, 35 and 45 ($Th_{23} - Th_{45}$). In this example, the description of SAN (CTA) were impracticable to the enormous amount of distinct synchronizing events needed. In a similar way to other models, SGSPN-Sparse models



FAS - First Available Server													
N		6			12			18			24		
		size	gen.	iter.	size	gen.	iter.	size	gen.	iter.	size	gen.	iter.
SAN	GTA	$\approx$ 0.00MB	0.01s	$\approx$ 0.00s	0.04MB	0.03s	$\approx$ 0.00s	2.01MB	1.58s	0.50s	128.02MB	126.50s	43.00s
	CTA	$\approx$ 0.00MB	$\approx$ 0.00s	$\approx$ 0.00s	0.07MB	0.08s	$\approx$ 0.00s	2.11MB	9.03s	0.80s	128.26MB	978.55s	93.80s
SGSPN	MDK	$\approx$ 0.00MB	$\approx$ 0.00s	$\approx$ 0.00s	0.03MB	0.06s	0.02s	1.02MB	5.55s	1.50s	64.04MB	441.86s	123.54s
	Sparse	$\approx$ 0.00MB	$\approx$ 0.00s	$\approx$ 0.00s	0.34MB	0.71s	$\approx$ 0.00s	31.00MB	107.71s	$\approx$ 0.00s	-	-	-
rss		64 states			4,096 states			262,144 states			16,777,216 states		

Figure 21: Results for First Available Server (FAS) models

(with few reachable states) have better results for the solution times. But, for models that converge fast (*e.g.* 450 iterations) the overall SAN (GTA) solution could be faster regarding SGSPN-Sparse solution.



LCS - Limited Capacity Subnets ($K_1 = 50$; $K_2 = 25$; $K_3 = 25$; $K_4 = 25$; $K_5 = 25$)													
$Th_{23} - Th_{45}$		$15 - 15$			$25 - 25$			$35 - 35$			$45 - 45$		
		size	gen.	iter.	size	gen.	iter.	size	gen.	iter.	size	gen.	iter.
SAN	GTA	7.22MB	6.4s	0.7s	48.00MB	29.2s	4.8s	120.38MB	69.0s	12.5s	170.12MB	96.7s	22.1s
SGSPN	MDK	3.63MB	4.1s	1.6s	24.04MB	31.0s	12.6s	60.25MB	80.4s	29.4s	85.13MB	116.2s	40.4s
	Sparse	87.77MB	280.2s	0.1s	614.63MB	2193.8s	1.1s	-	-	-	-	-	-
rss		943,296 states			6,283,251 states			15,765,936 states			22,282,971 states		

Figure 22: Results for Limited Capacity Subnets (LCS) models

References

[1] M. Ajmone-Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. An Introduction to Generalized Stochastic Petri Nets. *Microelectronics Reliability*, 31(4):699–725, 1991.

[2] M. Ajmone-Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[3] V. Amoia, G. De Micheli, and M. Santomauro. Computer-Oriented Formulation of Transition-Rate Matrices via Kronecker Algebra. *IEEE Transactions on Reliability*, R-30(2):123–132, 1981.

[4] K. Atif and B. Plateau. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[5] R. Bellman. *Introduction to Matrix Analysis*. McGraw-Hill, New York, 1960.

[6] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In *Performance TOOLS 2003*, pages 98–115, Urbana, Illinois, USA, 2003. Springer-Verlag.

[7] J. W. Brewer. Kronecker Products and Matrix Calculus in System Theory. *IEEE Transactions on Circuits and Systems*, CAS-25(9):772–780, 1978.

[8] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 13(3):203–222, 2000.

[9] P. Buchholz and T. Dayar. Block SOR for Kronecker structured representations. In *Proceedings of the 2003 International Conference on the Numerical Solution of Markov Chains*, pages 121–143, Urbana and Monticello, Illinois, USA, 2003. Springer-Verlag.

[10] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. In *Performance TOOLS 2003*, pages 78–97, Urbana-Champaign, Illinois, USA, September 2003. Springer-Verlag.

[11] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, 1981.

[12] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[13] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proceedings of the* 15^{th} *International Conference on Applications and Theory of Petri Nets*, pages 258–277, Springer-Verlag, Berlin Heidelberg, 1994. R. Valette.

[14] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor - Vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[15] S. Gilmore, J. Hillston, and M. Ribaudo. An Efficient Algorithm for Aggregating PEPA Models. *IEEE Transactions on Software Engineering*, 27(5):449–464, 2001.

[16] L. Kronecker. ber einige Interpolationsformeln fr ganze Funktionen mehrerer Variabeln. In *L. Kroneckers Werke*, volume I, pages 133–141, Lectures at the academy of sciences, December 21 1865. Chelsea Publishing Company.

[17] A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *Proceedings of the* 20^{th} *International Conference on Applications and Theory of Petri Nets*, pages 6–25, Williamsburg, VA, USA, June 1999. Springer-Verlag.

[18] A. S. Miner, G. Ciardo, and S. Donatelli. Using the exact state space of a Markov model to compute approximate stationary measures. In *Proceedings of the 2000 ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 207–216, Santa Clara, California, United States, June 2000. ACM Press.

[19] B. Plateau. *De l'Evaluation du Paralllisme et de la Synchronisation.* PhD thesis, Paris-Sud, Orsay, 1984.

[20] William H. Sanders and John F. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[21] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

A Examples - PEPS

A.1 OQN (CTA) - $K_1 = 250$ - $K_2 = 100$ - $K_3 = 100$

```
identifiers
    K1 = [0..250];
    K2 = [0..100];
    K3 = [0..100];

    C2 = 2;

    Lambda01 = 2.0;

    Mu01 = 0.4;
    Mu02 = 0.5;
    Mu03 = 0.2;

    pi12 = 0.7;
    pi13 = 0.3;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01*pi12;
    rate_q1toq3    = Mu01*pi13;
    rate_q2toOUT   = f2;
    rate_q3toOUT   = Mu03;

    f2 = min(C2, st Queue2)*Mu02;

events

    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e13 (rate_q1toq3);
    loc e2  (rate_q2toOUT);
    loc e3  (rate_q3toOUT);

reachability = 1;

network OQN (continuous)
  aut Queue1
    stt n[K1]
      to(++) e1
      to(--) e12 e13

  aut Queue2
    stt n[K2]
      to(++) e12
      to(--) e2

  aut Queue3
    stt n[K3]
      to(++) e13
      to(--) e3
      from n[100] to(n[100]) e13

results
  queue1 = st Queue1;
  queue2 = st Queue2;
  queue3 = st Queue3;

```

A.2 OQN (CTA) - $K_1 = 250$ - $K_2 = 150$ - $K_3 = 150$

```
identifiers
    K1 = [0..250];
    K2 = [0..150];
    K3 = [0..150];

    C2 = 2;

```

```
    Lambda01 = 2.0;

    Mu01 = 0.4;
    Mu02 = 0.5;
    Mu03 = 0.2;

    pi12 = 0.7;
    pi13 = 0.3;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01*pi12;
    rate_q1toq3    = Mu01*pi13;
    rate_q2toOUT   = f2;
    rate_q3toOUT   = Mu03;

    f2 = min(C2, st Queue2)*Mu02;

events

    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e13 (rate_q1toq3);
    loc e2  (rate_q2toOUT);
    loc e3  (rate_q3toOUT);

reachability = 1;

network OQN (continuous)
  aut Queue1
    stt n[K1]
      to(++) e1
      to(--) e12 e13

  aut Queue2
    stt n[K2]
      to(++) e12
      to(--) e2

  aut Queue3
    stt n[K3]
      to(++) e13
      to(--) e3
      from n[150] to(n[150]) e13

results
  queue1 = st Queue1;
  queue2 = st Queue2;
  queue3 = st Queue3;
```

A.3 OQN (CTA) - $K_1 = 250$ - $K_2 = 200$ - $K_3 = 200$

```
identifiers
    K1 = [0..250];
    K2 = [0..200];
    K3 = [0..200];

    C2 = 2;

    Lambda01 = 2.0;

    Mu01 = 0.4;
    Mu02 = 0.5;
    Mu03 = 0.2;

    pi12 = 0.7;
    pi13 = 0.3;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01*pi12;
```

```
    rate_q1toq3    = Mu01*pi13;
    rate_q2toOUT   = f2;
    rate_q3toOUT   = Mu03;

    f2 = min(C2, st Queue2)*Mu02;

events

    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e13 (rate_q1toq3);
    loc e2  (rate_q2toOUT);
    loc e3  (rate_q3toOUT);

reachability = 1;

network OQN (continuous)
  aut Queue1
    stt n[K1]
      to(++) e1
      to(--) e12 e13

  aut Queue2
    stt n[K2]
      to(++) e12
      to(--) e2

  aut Queue3
    stt n[K3]
      to(++) e13
      to(--) e3
      from n[200] to(n[200]) e13

results
  queue1 = st Queue1;
  queue2 = st Queue2;
  queue3 = st Queue3;
```

A.4 OQN (CTA) - $K_1 = 250$ - $K_2 = 250$ - $K_3 = 250$

```
identifiers
    K1 = [0..250];
    K2 = [0..250];
    K3 = [0..250];

    C2 = 2;

    Lambda01 = 2.0;

    Mu01 = 0.4;
    Mu02 = 0.5;
    Mu03 = 0.2;

    pi12 = 0.7;
    pi13 = 0.3;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01*pi12;
    rate_q1toq3    = Mu01*pi13;
    rate_q2toOUT   = f2;
    rate_q3toOUT   = Mu03;

    f2 = min(C2, st Queue2)*Mu02;

events

    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e13 (rate_q1toq3);
```

```
30      loc e2  (rate_q2toOUT);
31      loc e3  (rate_q3toOUT);
32
33  reachability = 1;
34
35  network OQN (continuous)
36    aut Queue1
37      stt n[K1]
38        to(++) e1
39        to(--) e12 e13
40
41    aut Queue2
42      stt n[K2]
43        to(++) e12
44        to(--) e2
45
46    aut Queue3
47      stt n[K3]
48        to(++) e13
49        to(--) e3
50        from n[250] to(n[250]) e13
51
52  results
53    queue1 = st Queue1;
54    queue2 = st Queue2;
55    queue3 = st Queue3;
56
```

A.5 RS (GTA) - $N = 24$ - $R = 3$

```
1   identifiers
2     lambda = 5;
3     mu = 10;
4     N = [0..23];
5     P = 3;
6     f = (nb [a1[0]..a1[23]] on < P) * mu;
7
8   events
9     loc take f;
10    loc leave lambda;
11
12  reachability = nb [a1[0]..a1[23]] on <= P;
13
14  network mutex (continuous)
15    aut a1[N]
16      stt off
17        to(on)
18          take
19      stt on
20        to(off)
21          leave
22
23  results
24    used = st a1[0] == on; // probability that the first
25                           // process uses the resource.
```

A.6 RS (GTA) - $N = 24$ - $R = 9$

```
1   identifiers
2     lambda = 5;
3     mu = 10;
4     N = [0..23];
5     P = 9;
6     f = (nb [a1[0]..a1[23]] on < P) * mu;
7
8   events
9     loc take f;
10    loc leave lambda;
11
12  reachability = nb [a1[0]..a1[23]] on <= P;
```

```
13
14  network mutex (continuous)
15    aut a1[N]
16      stt off
17        to(on)
18          take
19      stt on
20        to(off)
21          leave
22
23  results
24    used = st a1[0] == on; // probability that the first
25                           // process uses the resource.
```

A.7 RS (GTA) - $N = 24$ - $R = 15$

```
1   identifiers
2     lambda = 5;
3     mu = 10;
4     N = [0..23];
5     P = 15;
6     f = (nb [a1[0]..a1[23]] on < P) * mu;
7
8   events
9     loc take f;
10    loc leave lambda;
11
12  reachability = nb [a1[0]..a1[23]] on <= P;
13
14  network mutex (continuous)
15    aut a1[N]
16      stt off
17        to(on)
18          take
19      stt on
20        to(off)
21          leave
22
23  results
24    used = st a1[0] == on; // probability that the first
25                           // process uses the resource.
```

A.8 RS (GTA) - $N = 24$ - $R = 21$

```
1   identifiers
2     lambda = 5;
3     mu = 10;
4     N = [0..23];
5     P = 21;
6     f = (nb [a1[0]..a1[23]] on < P) * mu;
7
8   events
9     loc take f;
10    loc leave lambda;
11
12  reachability = nb [a1[0]..a1[23]] on <= P;
13
14  network mutex (continuous)
15    aut a1[N]
16      stt off
17        to(on)
18          take
19      stt on
20        to(off)
21          leave
22
23  results
24    used = st a1[0] == on; // probability that the first
25                           // process uses the resource.
```

A.9 RS (CTA) - $N = 24$ - $R = 3$

```
identifiers
  lambda = 0.3;
  mu = 0.1;
  N = [0..23];
  P = 3;
  Pd = [0..3];

events
  syn take[N] mu ;
  syn leave[N] lambda;

reachability = (nb [a0..a23] on <= P) && (nb [a0..a23] on == st rec);

network mutex (continuous)
  aut a0
    stt off to(on)  take[0]
    stt on  to(off) leave[0]

  aut a1
    stt off to(on)  take[1]
    stt on  to(off) leave[1]

  aut a2
    stt off to(on)  take[2]
    stt on  to(off) leave[2]

  aut a3
    stt off to(on)  take[3]
    stt on  to(off) leave[3]

  aut a4
    stt off to(on)  take[4]
    stt on  to(off) leave[4]

  aut a5
    stt off to(on)  take[5]
    stt on  to(off) leave[5]

  aut a6
    stt off to(on)  take[6]
    stt on  to(off) leave[6]

  aut a7
    stt off to(on)  take[7]
    stt on  to(off) leave[7]

  aut a8
    stt off to(on)  take[8]
    stt on  to(off) leave[8]

  aut a9
    stt off to(on)  take[9]
    stt on  to(off) leave[9]

  aut a10
    stt off to(on)  take[10]
    stt on  to(off) leave[10]

  aut a11
    stt off to(on)  take[11]
    stt on  to(off) leave[11]

  aut a12
    stt off to(on)  take[12]
    stt on  to(off) leave[12]

  aut a13
    stt off to(on)  take[13]
    stt on  to(off) leave[13]
```

```
70
71    aut a14
72      stt off to(on)  take[14]
73      stt on  to(off) leave[14]
74
75    aut a15
76      stt off to(on)  take[15]
77      stt on  to(off) leave[15]
78
79    aut a16
80      stt off to(on)  take[16]
81      stt on  to(off) leave[16]
82
83    aut a17
84      stt off to(on)  take[17]
85      stt on  to(off) leave[17]
86
87    aut a18
88      stt off to(on)  take[18]
89      stt on  to(off) leave[18]
90
91    aut a19
92      stt off to(on)  take[19]
93      stt on  to(off) leave[19]
94
95    aut a20
96      stt off to(on)  take[20]
97      stt on  to(off) leave[20]
98
99    aut a21
100     stt off to(on)  take[21]
101     stt on  to(off) leave[21]
102
103   aut a22
104     stt off to(on)  take[22]
105     stt on  to(off) leave[22]
106
107   aut a23
108     stt off to(on)  take[23]
109     stt on  to(off) leave[23]
110
111   aut rec
112     stt using[Pd]
113       to(++)
114         take[0] take[1] take[2] take[3] take[4] take[5]
115         take[6] take[7] take[8] take[9] take[10] take[11]
116         take[12] take[13] take[14] take[15] take[16] take[17]
117         take[18] take[19] take[20] take[21] take[22] take[23]
118       to(--)
119         leave[0] leave[1] leave[2] leave[3] leave[4] leave[5]
120         leave[6] leave[7] leave[8] leave[9] leave[10] leave[11]
121         leave[12] leave[13] leave[14] leave[15] leave[16] leave[17]
122         leave[18] leave[19] leave[20] leave[21] leave[22] leave[23]
123 results
124   used = st rec;
```

A.10 RS (CTA) - $N = 24$ - $R = 9$

```
1   identifiers
2     lambda = 0.3;
3     mu = 0.1;
4     N = [0..23];
5     P = 9;
6     Pd = [0..9];
7
8   events
9     syn take[N] mu ;
10    syn leave[N] lambda;
11
12  reachability = (nb [a0..a23] on <= P) && (nb [a0..a23] on == st rec);
```

```
13
14  network mutex (continuous)
15    aut a0
16      stt off to(on)  take[0]
17      stt on  to(off) leave[0]
18
19    aut a1
20      stt off to(on)  take[1]
21      stt on  to(off) leave[1]
22
23    aut a2
24      stt off to(on)  take[2]
25      stt on  to(off) leave[2]
26
27    aut a3
28      stt off to(on)  take[3]
29      stt on  to(off) leave[3]
30
31    aut a4
32      stt off to(on)  take[4]
33      stt on  to(off) leave[4]
34
35    aut a5
36      stt off to(on)  take[5]
37      stt on  to(off) leave[5]
38
39    aut a6
40      stt off to(on)  take[6]
41      stt on  to(off) leave[6]
42
43    aut a7
44      stt off to(on)  take[7]
45      stt on  to(off) leave[7]
46
47    aut a8
48      stt off to(on)  take[8]
49      stt on  to(off) leave[8]
50
51    aut a9
52      stt off to(on)  take[9]
53      stt on  to(off) leave[9]
54
55    aut a10
56      stt off to(on)  take[10]
57      stt on  to(off) leave[10]
58
59    aut a11
60      stt off to(on)  take[11]
61      stt on  to(off) leave[11]
62
63    aut a12
64      stt off to(on)  take[12]
65      stt on  to(off) leave[12]
66
67    aut a13
68      stt off to(on)  take[13]
69      stt on  to(off) leave[13]
70
71    aut a14
72      stt off to(on)  take[14]
73      stt on  to(off) leave[14]
74
75    aut a15
76      stt off to(on)  take[15]
77      stt on  to(off) leave[15]
78
79    aut a16
80      stt off to(on)  take[16]
81      stt on  to(off) leave[16]
82
83    aut a17
```

```
    stt off to(on)  take[17]
    stt on  to(off) leave[17]

  aut a18
    stt off to(on)  take[18]
    stt on  to(off) leave[18]

  aut a19
    stt off to(on)  take[19]
    stt on  to(off) leave[19]

  aut a20
    stt off to(on)  take[20]
    stt on  to(off) leave[20]

  aut a21
    stt off to(on)  take[21]
    stt on  to(off) leave[21]

  aut a22
    stt off to(on)  take[22]
    stt on  to(off) leave[22]

  aut a23
    stt off to(on)  take[23]
    stt on  to(off) leave[23]

  aut rec
    stt using[Pd]
      to(++)
        take[0] take[1] take[2] take[3] take[4] take[5]
        take[6] take[7] take[8] take[9] take[10] take[11]
        take[12] take[13] take[14] take[15] take[16] take[17]
        take[18] take[19] take[20] take[21] take[22] take[23]
      to(--)
        leave[0] leave[1] leave[2] leave[3] leave[4] leave[5]
        leave[6] leave[7] leave[8] leave[9] leave[10] leave[11]
        leave[12] leave[13] leave[14] leave[15] leave[16] leave[17]
        leave[18] leave[19] leave[20] leave[21] leave[22] leave[23]
results
  used = st rec;
```

A.11 RS (CTA) - $N = 24$ - $R = 15$

```
identifiers
  lambda = 0.3;
  mu = 0.1;
  N = [0..23];
  P = 15;
  Pd = [0..15];

events
  syn take[N] mu ;
  syn leave[N] lambda;

reachability = (nb [a0..a23] on <= P) && (nb [a0..a23] on == st rec);

network mutex (continuous)
  aut a0
    stt off to(on)  take[0]
    stt on  to(off) leave[0]

  aut a1
    stt off to(on)  take[1]
    stt on  to(off) leave[1]

  aut a2
    stt off to(on)  take[2]
    stt on  to(off) leave[2]

```

```
27    aut a3
28      stt off to(on)  take[3]
29      stt on  to(off) leave[3]
30
31    aut a4
32      stt off to(on)  take[4]
33      stt on  to(off) leave[4]
34
35    aut a5
36      stt off to(on)  take[5]
37      stt on  to(off) leave[5]
38
39    aut a6
40      stt off to(on)  take[6]
41      stt on  to(off) leave[6]
42
43    aut a7
44      stt off to(on)  take[7]
45      stt on  to(off) leave[7]
46
47    aut a8
48      stt off to(on)  take[8]
49      stt on  to(off) leave[8]
50
51    aut a9
52      stt off to(on)  take[9]
53      stt on  to(off) leave[9]
54
55    aut a10
56      stt off to(on)  take[10]
57      stt on  to(off) leave[10]
58
59    aut a11
60      stt off to(on)  take[11]
61      stt on  to(off) leave[11]
62
63    aut a12
64      stt off to(on)  take[12]
65      stt on  to(off) leave[12]
66
67    aut a13
68      stt off to(on)  take[13]
69      stt on  to(off) leave[13]
70
71    aut a14
72      stt off to(on)  take[14]
73      stt on  to(off) leave[14]
74
75    aut a15
76      stt off to(on)  take[15]
77      stt on  to(off) leave[15]
78
79    aut a16
80      stt off to(on)  take[16]
81      stt on  to(off) leave[16]
82
83    aut a17
84      stt off to(on)  take[17]
85      stt on  to(off) leave[17]
86
87    aut a18
88      stt off to(on)  take[18]
89      stt on  to(off) leave[18]
90
91    aut a19
92      stt off to(on)  take[19]
93      stt on  to(off) leave[19]
94
95    aut a20
96      stt off to(on)  take[20]
97      stt on  to(off) leave[20]
```

```
98
99    aut a21
100     stt off to(on)  take[21]
101     stt on  to(off) leave[21]
102
103   aut a22
104     stt off to(on)  take[22]
105     stt on  to(off) leave[22]
106
107   aut a23
108     stt off to(on)  take[23]
109     stt on  to(off) leave[23]
110
111   aut rec
112     stt using[Pd]
113       to(++)
114         take[0] take[1] take[2] take[3] take[4] take[5]
115         take[6] take[7] take[8] take[9] take[10] take[11]
116         take[12] take[13] take[14] take[15] take[16] take[17]
117         take[18] take[19] take[20] take[21] take[22] take[23]
118       to(--)
119         leave[0] leave[1] leave[2] leave[3] leave[4] leave[5]
120         leave[6] leave[7] leave[8] leave[9] leave[10] leave[11]
121         leave[12] leave[13] leave[14] leave[15] leave[16] leave[17]
122         leave[18] leave[19] leave[20] leave[21] leave[22] leave[23]
123 results
124   used = st rec;
```

A.12 RS (CTA) - $N = 24$ - $R = 21$

```
1   identifiers
2     lambda = 0.3;
3     mu = 0.1;
4     N = [0..23];
5     P = 21;
6     Pd = [0..21];
7
8   events
9     syn take[N] mu ;
10    syn leave[N] lambda;
11
12  reachability = (nb [a0..a23] on <= P) && (nb [a0..a23] on == st rec);
13
14  network mutex (continuous)
15    aut a0
16      stt off to(on)  take[0]
17      stt on  to(off) leave[0]
18
19    aut a1
20      stt off to(on)  take[1]
21      stt on  to(off) leave[1]
22
23    aut a2
24      stt off to(on)  take[2]
25      stt on  to(off) leave[2]
26
27    aut a3
28      stt off to(on)  take[3]
29      stt on  to(off) leave[3]
30
31    aut a4
32      stt off to(on)  take[4]
33      stt on  to(off) leave[4]
34
35    aut a5
36      stt off to(on)  take[5]
37      stt on  to(off) leave[5]
38
39    aut a6
40      stt off to(on)  take[6]
```

```
41        stt on  to(off) leave[6]
42
43      aut a7
44        stt off to(on)  take[7]
45        stt on  to(off) leave[7]
46
47      aut a8
48        stt off to(on)  take[8]
49        stt on  to(off) leave[8]
50
51      aut a9
52        stt off to(on)  take[9]
53        stt on  to(off) leave[9]
54
55      aut a10
56        stt off to(on)  take[10]
57        stt on  to(off) leave[10]
58
59      aut a11
60        stt off to(on)  take[11]
61        stt on  to(off) leave[11]
62
63      aut a12
64        stt off to(on)  take[12]
65        stt on  to(off) leave[12]
66
67      aut a13
68        stt off to(on)  take[13]
69        stt on  to(off) leave[13]
70
71      aut a14
72        stt off to(on)  take[14]
73        stt on  to(off) leave[14]
74
75      aut a15
76        stt off to(on)  take[15]
77        stt on  to(off) leave[15]
78
79      aut a16
80        stt off to(on)  take[16]
81        stt on  to(off) leave[16]
82
83      aut a17
84        stt off to(on)  take[17]
85        stt on  to(off) leave[17]
86
87      aut a18
88        stt off to(on)  take[18]
89        stt on  to(off) leave[18]
90
91      aut a19
92        stt off to(on)  take[19]
93        stt on  to(off) leave[19]
94
95      aut a20
96        stt off to(on)  take[20]
97        stt on  to(off) leave[20]
98
99      aut a21
100       stt off to(on)  take[21]
101       stt on  to(off) leave[21]
102
103     aut a22
104       stt off to(on)  take[22]
105       stt on  to(off) leave[22]
106
107     aut a23
108       stt off to(on)  take[23]
109       stt on  to(off) leave[23]
110
111     aut rec
```

```
    stt using[Pd]
      to(++)
        take[0] take[1] take[2] take[3] take[4] take[5]
        take[6] take[7] take[8] take[9] take[10] take[11]
        take[12] take[13] take[14] take[15] take[16] take[17]
        take[18] take[19] take[20] take[21] take[22] take[23]
      to(--)
        leave[0] leave[1] leave[2] leave[3] leave[4] leave[5]
        leave[6] leave[7] leave[8] leave[9] leave[10] leave[11]
        leave[12] leave[13] leave[14] leave[15] leave[16] leave[17]
        leave[18] leave[19] leave[20] leave[21] leave[22] leave[23]
results
  used = st rec;
```

A.13 ASP (GTA) - $P = 2$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1  = 4;
  t_arrive2  = 1;
  t_service1 = 3;
  t_service2 = 2;
  t_service3 = ((st service == p1) * 6) + ((st service == p2) * 3);
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.6;

events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_4   (t_service3);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_4
    from n[10] to (n[10]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_4
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
    stt p2   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.14 ASP (GTA) - $P = 3$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1  = 4;
  t_arrive2  = 1;
  t_service1 = 3;
  t_service2 = 2;
  t_service3 = ((st service == p1) * 6) + ((st service == p2) * 3) + ((st service == p3) * 2);
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.3;
  pi3        = 0.3;


events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_4   (t_service3);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_4
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_4
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
    stt p2   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
    stt p3   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.15 ASP (GTA) - $P = 4$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1  = 4;
```

```
  t_arrive2  = 1;
  t_service1 = 3;
  t_service2 = 2;
  t_service3 = ((st service == p1) * 6) + ((st service == p2) * 3) + ((st service == p3) * 2) +
               ((st service == p4) * 1);
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.3;
  pi3        = 0.15;
  pi4        = 0.15;


events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_4   (t_service3);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_4
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_4
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
    stt p2   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
    stt p3   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
    stt p4   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.16 ASP (GTA) - $P = 5$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
```

```
  K4         = [0..60];
  t_arrive1  = 4;
  t_arrive2  = 1;
  t_service1 = 3;
  t_service2 = 2;
  t_service3 = ((st service == p1) * 6) + ((st service == p2) * 3) + ((st service == p3) * 2) +
               ((st service == p4) * 1) + ((st service == p5) * 4);
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.3;
  pi3        = 0.15;
  pi4        = 0.10;
  pi5        = 0.05;


events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_4   (t_service3);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_4
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_4
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
             to (p5) rot_3_4(pi5)
    stt p2   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
             to (p5) rot_3_4(pi5)
    stt p3   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
             to (p5) rot_3_4(pi5)
    stt p4   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
             to (p5) rot_3_4(pi5)
    stt p5   to (p1) rot_3_4(pi1)
             to (p2) rot_3_4(pi2)
             to (p3) rot_3_4(pi3)
             to (p4) rot_3_4(pi4)
             to (p5) rot_3_4(pi5)

```

```
results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.17 ASP (CTA) - $P = 2$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1   = 4;
  t_arrive2   = 1;
  t_service1  = 3;
  t_service2  = 2;
  t_service31 = 6;
  t_service32 = 3;
  t_service4  = 5;
  pi1         = 0.4;
  pi2         = 0.6;

events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_41  (t_service31);
  syn rot_3_42  (t_service32);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_41 rot_3_42
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_41 rot_3_42
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_41(pi1)
             to (p2) rot_3_41(pi2)
    stt p2   to (p1) rot_3_42(pi1)
             to (p2) rot_3_42(pi2)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.18 ASP (CTA) - $P = 3$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
```

```
  t_arrive1   = 4;
  t_arrive2   = 1;
  t_service1  = 3;
  t_service2  = 2;
  t_service31 = 6;
  t_service32 = 3;
  t_service33 = 2;
  t_service4  = 5;
  pi1         = 0.4;
  pi2         = 0.3;
  pi3         = 0.3;


events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_41  (t_service31);
  syn rot_3_42  (t_service32);
  syn rot_3_43  (t_service33);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_41 rot_3_42 rot_3_43
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_41 rot_3_42 rot_3_43
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_41(pi1)
             to (p2) rot_3_41(pi2)
             to (p3) rot_3_41(pi3)
    stt p2   to (p1) rot_3_42(pi1)
             to (p2) rot_3_42(pi2)
             to (p3) rot_3_42(pi3)
    stt p3   to (p1) rot_3_43(pi1)
             to (p2) rot_3_43(pi2)
             to (p3) rot_3_43(pi3)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.19 ASP (CTA) - $P = 4$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1   = 4;
  t_arrive2   = 1;
  t_service1  = 3;
```

```
  t_service2  = 2;
  t_service31 = 6;
  t_service32 = 3;
  t_service33 = 2;
  t_service34 = 1;
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.3;
  pi3        = 0.15;
  pi4        = 0.15;


events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_41  (t_service31);
  syn rot_3_42  (t_service32);
  syn rot_3_43  (t_service33);
  syn rot_3_44  (t_service34);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_41 rot_3_42 rot_3_43 rot_3_44
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_41 rot_3_42 rot_3_43 rot_3_44
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_41(pi1)
             to (p2) rot_3_41(pi2)
             to (p3) rot_3_41(pi3)
             to (p4) rot_3_41(pi4)
    stt p2   to (p1) rot_3_42(pi1)
             to (p2) rot_3_42(pi2)
             to (p3) rot_3_42(pi3)
             to (p4) rot_3_42(pi4)
    stt p3   to (p1) rot_3_43(pi1)
             to (p2) rot_3_43(pi2)
             to (p3) rot_3_43(pi3)
             to (p4) rot_3_43(pi4)
    stt p4   to (p1) rot_3_44(pi1)
             to (p2) rot_3_44(pi2)
             to (p3) rot_3_44(pi3)
             to (p4) rot_3_44(pi4)

results
  used    = st q1;
  p_serv1 = st service == p1;
  p_serv2 = st service == p2;
```

A.20 ASP (CTA) - $P = 5$

```
identifiers
  K1         = [0..30];
  K2         = [0..30];
  K3         = [0..60];
  K4         = [0..60];
  t_arrive1   = 4;
  t_arrive2   = 1;
  t_service1  = 3;
  t_service2  = 2;
  t_service31 = 6;
  t_service32 = 3;
  t_service33 = 2;
  t_service34 = 1;
  t_service35 = 4;
  t_service4 = 5;
  pi1        = 0.4;
  pi2        = 0.3;
  pi3        = 0.15;
  pi4        = 0.10;
  pi5        = 0.05;

events
  loc l1        (t_arrive1);
  loc l2        (t_arrive2);
  syn rot_1_3   (t_service1);
  syn rot_2_3   (t_service2);
  syn rot_3_41  (t_service31);
  syn rot_3_42  (t_service32);
  syn rot_3_43  (t_service33);
  syn rot_3_44  (t_service34);
  syn rot_3_45  (t_service35);
  loc rot_4_out (t_service4);


reachability = 1;

network aspq (continuous)
  aut q1
    stt n[K1]  to (++) l1
               to (--) rot_1_3

  aut q2
    stt n[K2]  to (++) l2
               to (--) rot_2_3

  aut q3
    stt n[K3]  to (++) rot_1_3 rot_2_3
               to (--) rot_3_41 rot_3_42 rot_3_43 rot_3_44 rot_3_45
    from n[60] to (n[60]) rot_2_3

  aut q4
    stt n[K4]  to (++) rot_3_41 rot_3_42 rot_3_43 rot_3_44 rot_3_45
               to (--) rot_4_out

  aut service
    stt p1   to (p1) rot_3_41(pi1)
             to (p2) rot_3_41(pi2)
             to (p3) rot_3_41(pi3)
             to (p4) rot_3_41(pi4)
             to (p5) rot_3_41(pi5)
    stt p2   to (p1) rot_3_42(pi1)
             to (p2) rot_3_42(pi2)
             to (p3) rot_3_42(pi3)
             to (p4) rot_3_42(pi4)
             to (p5) rot_3_42(pi5)
    stt p3   to (p1) rot_3_43(pi1)
             to (p2) rot_3_43(pi2)
             to (p3) rot_3_43(pi3)
             to (p4) rot_3_43(pi4)
```

```
70                to (p5) rot_3_43(pi5)
71      stt p4    to (p1) rot_3_44(pi1)
72                to (p2) rot_3_44(pi2)
73                to (p3) rot_3_44(pi3)
74                to (p4) rot_3_44(pi4)
75                to (p5) rot_3_44(pi5)
76      stt p5    to (p1) rot_3_45(pi1)
77                to (p2) rot_3_45(pi2)
78                to (p3) rot_3_45(pi3)
79                to (p4) rot_3_45(pi4)
80                to (p5) rot_3_45(pi5)
81
82  results
83    used    = st q1;
84    p_serv1 = st service == p1;
85    p_serv2 = st service == p2;
```

A.21 FAS (GTA) - $N = 6$

```
1   identifiers
2       Lambda = 1;
3
4       Mu = 0.25;
5
6       f1  = Lambda;
7       f2  = (st C1==busy) * Lambda;
8       f3  = (nb[C1..C2] busy == 2) * Lambda;
9       f4  = (nb[C1..C3] busy == 3) * Lambda;
10      f5  = (nb[C1..C4] busy == 4) * Lambda;
11      f6  = (nb[C1..C5] busy == 5) * Lambda;
12
13  events
14      loc A1 f1;
15      loc R1 Mu;
16
17      loc A2 f2;
18      loc R2 Mu;
19
20      loc A3 f3;
21      loc R3 Mu;
22
23      loc A4 f4;
24      loc R4 Mu;
25
26      loc A5 f5;
27      loc R5 Mu;
28
29      loc A6 f6;
30      loc R6 Mu;
31
32  reachability = 1;
33
34  network FAS (continuous)
35    aut C1  stt idle to(busy) A1
36            stt busy to(idle) R1
37    aut C2  stt idle to(busy) A2
38            stt busy to(idle) R2
39    aut C3  stt idle to(busy) A3
40            stt busy to(idle) R3
41    aut C4  stt idle to(busy) A4
42            stt busy to(idle) R4
43    aut C5  stt idle to(busy) A5
44            stt busy to(idle) R5
45    aut C6  stt idle to(busy) A6
46            stt busy to(idle) R6
47
48  results
49    Used1 = st C1 == busy;
50    Used2 = st C2 == busy;
51    Used3 = st C3 == busy;
```

A.22 FAS (GTA) - $N = 12$

```
identifiers
    Lambda = 1;

    Mu = 0.25;

    f1   = Lambda;
    f2   = (st C1==busy) * Lambda;
    f3   = (nb[C1..C2] busy == 2) * Lambda;
    f4   = (nb[C1..C3] busy == 3) * Lambda;
    f5   = (nb[C1..C4] busy == 4) * Lambda;
    f6   = (nb[C1..C5] busy == 5) * Lambda;
    f7   = (nb[C1..C6] busy == 6) * Lambda;
    f8   = (nb[C1..C7] busy == 7) * Lambda;
    f9   = (nb[C1..C8] busy == 8) * Lambda;
    f10  = (nb[C1..C9] busy == 9) * Lambda;
    f11  = (nb[C1..C10] busy == 10) * Lambda;
    f12  = (nb[C1..C11] busy == 11) * Lambda;

events
    loc A1 f1;
    loc R1 Mu;

    loc A2 f2;
    loc R2 Mu;

    loc A3 f3;
    loc R3 Mu;

    loc A4 f4;
    loc R4 Mu;

    loc A5 f5;
    loc R5 Mu;

    loc A6 f6;
    loc R6 Mu;

    loc A7 f7;
    loc R7 Mu;

    loc A8 f8;
    loc R8 Mu;

    loc A9 f9;
    loc R9 Mu;

    loc A10 f10;
    loc R10 Mu;

    loc A11 f11;
    loc R11 Mu;

    loc A12 f12;
    loc R12 Mu;

reachability = 1;

network FAS (continuous)
  aut C1  stt idle to(busy) A1
          stt busy to(idle) R1
  aut C2  stt idle to(busy) A2
          stt busy to(idle) R2
  aut C3  stt idle to(busy) A3
          stt busy to(idle) R3
  aut C4  stt idle to(busy) A4
          stt busy to(idle) R4
  aut C5  stt idle to(busy) A5
          stt busy to(idle) R5
  aut C6  stt idle to(busy) A6
```

```
            stt busy to(idle) R6
    aut C7  stt idle to(busy) A7
            stt busy to(idle) R7
    aut C8  stt idle to(busy) A8
            stt busy to(idle) R8
    aut C9  stt idle to(busy) A9
            stt busy to(idle) R9
    aut C10 stt idle to(busy) A10
            stt busy to(idle) R10
    aut C11 stt idle to(busy) A11
            stt busy to(idle) R11
    aut C12 stt idle to(busy) A12
            stt busy to(idle) R12

results
  Used1 = st C1 == busy;
  Used2 = st C2 == busy;
  Used3 = st C3 == busy;
```

A.23 FAS (GTA) - $N = 18$

```
identifiers
    Lambda = 1;

    Mu = 0.25;

    f1   = Lambda;
    f2   = (st C1==busy) * Lambda;
    f3   = (nb[C1..C2] busy == 2) * Lambda;
    f4   = (nb[C1..C3] busy == 3) * Lambda;
    f5   = (nb[C1..C4] busy == 4) * Lambda;
    f6   = (nb[C1..C5] busy == 5) * Lambda;
    f7   = (nb[C1..C6] busy == 6) * Lambda;
    f8   = (nb[C1..C7] busy == 7) * Lambda;
    f9   = (nb[C1..C8] busy == 8) * Lambda;
    f10  = (nb[C1..C9] busy == 9) * Lambda;
    f11  = (nb[C1..C10] busy == 10) * Lambda;
    f12  = (nb[C1..C11] busy == 11) * Lambda;
    f13  = (nb[C1..C12] busy == 12) * Lambda;
    f14  = (nb[C1..C13] busy == 13) * Lambda;
    f15  = (nb[C1..C14] busy == 14) * Lambda;
    f16  = (nb[C1..C15] busy == 15) * Lambda;
    f17  = (nb[C1..C16] busy == 16) * Lambda;
    f18  = (nb[C1..C17] busy == 17) * Lambda;

events
    loc A1 f1;
    loc R1 Mu;

    loc A2 f2;
    loc R2 Mu;

    loc A3 f3;
    loc R3 Mu;

    loc A4 f4;
    loc R4 Mu;

    loc A5 f5;
    loc R5 Mu;

    loc A6 f6;
    loc R6 Mu;

    loc A7 f7;
    loc R7 Mu;

    loc A8 f8;
    loc R8 Mu;

```

```
50      loc A9 f9;
51      loc R9 Mu;
52
53      loc A10 f10;
54      loc R10 Mu;
55
56      loc A11 f11;
57      loc R11 Mu;
58
59      loc A12 f12;
60      loc R12 Mu;
61
62      loc A13 f13;
63      loc R13 Mu;
64
65      loc A14 f14;
66      loc R14 Mu;
67
68      loc A15 f15;
69      loc R15 Mu;
70
71      loc A16 f16;
72      loc R16 Mu;
73
74      loc A17 f17;
75      loc R17 Mu;
76
77      loc A18 f18;
78      loc R18 Mu;
79
80  reachability = 1;
81
82  network FAS (continuous)
83    aut C1  stt idle to(busy) A1
84            stt busy to(idle) R1
85    aut C2  stt idle to(busy) A2
86            stt busy to(idle) R2
87    aut C3  stt idle to(busy) A3
88            stt busy to(idle) R3
89    aut C4  stt idle to(busy) A4
90            stt busy to(idle) R4
91    aut C5  stt idle to(busy) A5
92            stt busy to(idle) R5
93    aut C6  stt idle to(busy) A6
94            stt busy to(idle) R6
95    aut C7  stt idle to(busy) A7
96            stt busy to(idle) R7
97    aut C8  stt idle to(busy) A8
98            stt busy to(idle) R8
99    aut C9  stt idle to(busy) A9
100           stt busy to(idle) R9
101   aut C10 stt idle to(busy) A10
102           stt busy to(idle) R10
103   aut C11 stt idle to(busy) A11
104           stt busy to(idle) R11
105   aut C12 stt idle to(busy) A12
106           stt busy to(idle) R12
107   aut C13 stt idle to(busy) A13
108           stt busy to(idle) R13
109   aut C14 stt idle to(busy) A14
110           stt busy to(idle) R14
111   aut C15 stt idle to(busy) A15
112           stt busy to(idle) R15
113   aut C16 stt idle to(busy) A16
114           stt busy to(idle) R16
115   aut C17 stt idle to(busy) A17
116           stt busy to(idle) R17
117   aut C18 stt idle to(busy) A18
118           stt busy to(idle) R18
119
120 results
```

```
121     Used1 = st C1 == busy;
122     Used2 = st C2 == busy;
123     Used3 = st C3 == busy;
```

A.24 FAS (GTA) - $N = 24$

```
1   identifiers
2       Lambda = 1;
3
4       Mu = 0.25;
5
6       f1   = Lambda;
7       f2   = (st C1==busy) * Lambda;
8       f3   = (nb[C1..C2] busy == 2) * Lambda;
9       f4   = (nb[C1..C3] busy == 3) * Lambda;
10      f5   = (nb[C1..C4] busy == 4) * Lambda;
11      f6   = (nb[C1..C5] busy == 5) * Lambda;
12      f7   = (nb[C1..C6] busy == 6) * Lambda;
13      f8   = (nb[C1..C7] busy == 7) * Lambda;
14      f9   = (nb[C1..C8] busy == 8) * Lambda;
15      f10  = (nb[C1..C9] busy == 9) * Lambda;
16      f11  = (nb[C1..C10] busy == 10) * Lambda;
17      f12  = (nb[C1..C11] busy == 11) * Lambda;
18      f13  = (nb[C1..C12] busy == 12) * Lambda;
19      f14  = (nb[C1..C13] busy == 13) * Lambda;
20      f15  = (nb[C1..C14] busy == 14) * Lambda;
21      f16  = (nb[C1..C15] busy == 15) * Lambda;
22      f17  = (nb[C1..C16] busy == 16) * Lambda;
23      f18  = (nb[C1..C17] busy == 17) * Lambda;
24      f19  = (nb[C1..C18] busy == 18) * Lambda;
25      f20  = (nb[C1..C19] busy == 19) * Lambda;
26      f21  = (nb[C1..C20] busy == 20) * Lambda;
27      f22  = (nb[C1..C21] busy == 21) * Lambda;
28      f23  = (nb[C1..C22] busy == 22) * Lambda;
29      f24  = (nb[C1..C23] busy == 23) * Lambda;
30
31  events
32      loc A1 f1;
33      loc R1 Mu;
34
35      loc A2 f2;
36      loc R2 Mu;
37
38      loc A3 f3;
39      loc R3 Mu;
40
41      loc A4 f4;
42      loc R4 Mu;
43
44      loc A5 f5;
45      loc R5 Mu;
46
47      loc A6 f6;
48      loc R6 Mu;
49
50      loc A7 f7;
51      loc R7 Mu;
52
53      loc A8 f8;
54      loc R8 Mu;
55
56      loc A9 f9;
57      loc R9 Mu;
58
59      loc A10 f10;
60      loc R10 Mu;
61
62      loc A11 f11;
63      loc R11 Mu;
64
```

```
65      loc A12 f12;
66      loc R12 Mu;
67
68      loc A13 f13;
69      loc R13 Mu;
70
71      loc A14 f14;
72      loc R14 Mu;
73
74      loc A15 f15;
75      loc R15 Mu;
76
77      loc A16 f16;
78      loc R16 Mu;
79
80      loc A17 f17;
81      loc R17 Mu;
82
83      loc A18 f18;
84      loc R18 Mu;
85
86      loc A19 f19;
87      loc R19 Mu;
88
89      loc A20 f20;
90      loc R20 Mu;
91
92      loc A21 f21;
93      loc R21 Mu;
94
95      loc A22 f22;
96      loc R22 Mu;
97
98      loc A23 f23;
99      loc R23 Mu;
100
101     loc A24 f24;
102     loc R24 Mu;
103
104 reachability = 1;
105
106 network FAS (continuous)
107   aut C1  stt idle to(busy) A1
108           stt busy to(idle) R1
109   aut C2  stt idle to(busy) A2
110           stt busy to(idle) R2
111   aut C3  stt idle to(busy) A3
112           stt busy to(idle) R3
113   aut C4  stt idle to(busy) A4
114           stt busy to(idle) R4
115   aut C5  stt idle to(busy) A5
116           stt busy to(idle) R5
117   aut C6  stt idle to(busy) A6
118           stt busy to(idle) R6
119   aut C7  stt idle to(busy) A7
120           stt busy to(idle) R7
121   aut C8  stt idle to(busy) A8
122           stt busy to(idle) R8
123   aut C9  stt idle to(busy) A9
124           stt busy to(idle) R9
125   aut C10 stt idle to(busy) A10
126           stt busy to(idle) R10
127   aut C11 stt idle to(busy) A11
128           stt busy to(idle) R11
129   aut C12 stt idle to(busy) A12
130           stt busy to(idle) R12
131   aut C13 stt idle to(busy) A13
132           stt busy to(idle) R13
133   aut C14 stt idle to(busy) A14
134           stt busy to(idle) R14
135   aut C15 stt idle to(busy) A15
```

```
            stt busy to(idle) R15
    aut C16 stt idle to(busy) A16
            stt busy to(idle) R16
    aut C17 stt idle to(busy) A17
            stt busy to(idle) R17
    aut C18 stt idle to(busy) A18
            stt busy to(idle) R18
    aut C19 stt idle to(busy) A19
            stt busy to(idle) R19
    aut C20 stt idle to(busy) A20
            stt busy to(idle) R20
    aut C21 stt idle to(busy) A21
            stt busy to(idle) R21
    aut C22 stt idle to(busy) A22
            stt busy to(idle) R22
    aut C23 stt idle to(busy) A23
            stt busy to(idle) R23
    aut C24 stt idle to(busy) A24
            stt busy to(idle) R24
  results
    Used1 = st C1 == busy;
    Used2 = st C2 == busy;
    Used3 = st C3 == busy;
```

A.25 FAS (CTA) - $N = 6$

```
identifiers
 lambda = 1;
 alpha  = 0.25;
 N      = [2..6];

events
 loc D    (lambda);
 loc L1   (alpha);
 syn L[N] (alpha);

reachability = 1;

network FAS (continuous)
 aut C1  stt idle to(busy) L1
         stt busy to(idle) D L[2] L[3] L[4] L[5] L[6]
 aut C2  stt idle to(busy) L[2]
         stt busy to(idle) D L[3] L[4] L[5] L[6]
 aut C3  stt idle to(busy) L[3]
         stt busy to(idle) D L[4] L[5] L[6]
 aut C4  stt idle to(busy) L[4]
         stt busy to(idle) D L[5] L[6]
 aut C5  stt idle to(busy) L[5]
         stt busy to(idle) D L[6]
 aut C6  stt idle to(busy) L[6]
         stt busy to(idle) D


results
 full  = nb busy;
 use1  = st C1 == busy;
 use2  = st C2 == busy;
 use3  = st C3 == busy;
 average = nb busy;
```

A.26 FAS (CTA) - $N = 12$

```
identifiers
 lambda = 1;
 alpha  = 0.25;
 N      = [2..12];

events
 loc D    (lambda);
 loc L1   (alpha);
```

```
 syn L[N] (alpha);

reachability = 1;

network FAS (continuous)
 aut C1  stt idle to(busy) L1
         stt busy to(idle) D L[2] L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12]
 aut C2  stt idle to(busy) L[2]
         stt busy to(idle) D L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12]
 aut C3  stt idle to(busy) L[3]
         stt busy to(idle) D L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12]
 aut C4  stt idle to(busy) L[4]
         stt busy to(idle) D L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12]
 aut C5  stt idle to(busy) L[5]
         stt busy to(idle) D L[6] L[7] L[8] L[9] L[10] L[11] L[12]
 aut C6  stt idle to(busy) L[6]
         stt busy to(idle) D L[7] L[8] L[9] L[10] L[11] L[12]
 aut C7  stt idle to(busy) L[7]
         stt busy to(idle) D L[8] L[9] L[10] L[11] L[12]
 aut C8  stt idle to(busy) L[8]
         stt busy to(idle) D L[9] L[10] L[11] L[12]
 aut C9  stt idle to(busy) L[9]
         stt busy to(idle) D L[10] L[11] L[12]
 aut C10 stt idle to(busy) L[10]
         stt busy to(idle) D L[11] L[12]
 aut C11 stt idle to(busy) L[11]
         stt busy to(idle) D L[12]
 aut C12 stt idle to(busy) L[12]
         stt busy to(idle) D


results
 full  = nb busy;
 use1  = st C1 == busy;
 use2  = st C2 == busy;
 use3  = st C3 == busy;
 average = nb busy;
```

A.27 FAS (CTA) - $N = 18$

```
identifiers
 lambda = 1;
 alpha  = 0.25;
 N      = [2..18];

events
 loc D    (lambda);
 loc L1   (alpha);
 syn L[N] (alpha);

reachability = 1;

network FAS (continuous)
 aut C1  stt idle to(busy) L1
         stt busy to(idle) D L[2] L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10]
                           L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C2  stt idle to(busy) L[2]
         stt busy to(idle) D L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11]
                           L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C3  stt idle to(busy) L[3]
         stt busy to(idle) D L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12]
                           L[13] L[14] L[15] L[16] L[17] L[18]
 aut C4  stt idle to(busy) L[4]
         stt busy to(idle) D L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13]
                           L[14] L[15] L[16] L[17] L[18]
 aut C5  stt idle to(busy) L[5]
         stt busy to(idle) D L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14]
                           L[15] L[16] L[17] L[18]
 aut C6  stt idle to(busy) L[6]
         stt busy to(idle) D L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15]
```

```
                              L[16] L[17] L[18]
 aut C7  stt idle to(busy) L[7]
         stt busy to(idle) D L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15]
                              L[16] L[17] L[18]
 aut C8  stt idle to(busy) L[8]
         stt busy to(idle) D L[9] L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C9  stt idle to(busy) L[9]
         stt busy to(idle) D L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C10 stt idle to(busy) L[10]
         stt busy to(idle) D L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C11 stt idle to(busy) L[11]
         stt busy to(idle) D L[12] L[13] L[14] L[15] L[16] L[17] L[18]
 aut C12 stt idle to(busy) L[12]
         stt busy to(idle) D L[13] L[14] L[15] L[16] L[17] L[18]
 aut C13 stt idle to(busy) L[13]
         stt busy to(idle) D L[14] L[15] L[16] L[17] L[18]
 aut C14 stt idle to(busy) L[14]
         stt busy to(idle) D L[15] L[16] L[17] L[18]
 aut C15 stt idle to(busy) L[15]
         stt busy to(idle) D L[16] L[17] L[18]
 aut C16 stt idle to(busy) L[16]
         stt busy to(idle) D L[17] L[18]
 aut C17 stt idle to(busy) L[17]
         stt busy to(idle) D L[18]
 aut C18 stt idle to(busy) L[18]
         stt busy to(idle) D

results
 full  = nb busy;
 use1  = st C1 == busy;
 use2  = st C2 == busy;
 use3  = st C3 == busy;
 average = nb busy;
```

A.28 FAS (CTA) - $N = 24$

```
identifiers
 lambda = 1;
 alpha  = 0.25;
 N      = [2..24];

events
 loc D    (lambda);
 loc L1   (alpha);
 syn L[N] (alpha);

reachability = 1;

network FAS (continuous)
 aut C1  stt idle to(busy) L1
         stt busy to(idle) D L[2] L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13]
                              L[14] L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C2  stt idle to(busy) L[2]
         stt busy to(idle) D L[3] L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13]
                              L[14] L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C3  stt idle to(busy) L[3]
         stt busy to(idle) D L[4] L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14]
                              L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C4  stt idle to(busy) L[4]
         stt busy to(idle) D L[5] L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15]
                              L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C5  stt idle to(busy) L[5]
         stt busy to(idle) D L[6] L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15] L[16]
                              L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C6  stt idle to(busy) L[6]
         stt busy to(idle) D L[7] L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17]
                              L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C7  stt idle to(busy) L[7]
         stt busy to(idle) D L[8] L[9] L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18]
                              L[19] L[20] L[21] L[22] L[23] L[24]
```

```
 aut C8  stt idle to(busy) L[8]
         stt busy to(idle) D L[9] L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18] L[19]
                           L[20] L[21] L[22] L[23] L[24]
 aut C9  stt idle to(busy) L[9]
         stt busy to(idle) D L[10] L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18] L[19]
                           L[20] L[21] L[22] L[23] L[24]
 aut C10 stt idle to(busy) L[10]
         stt busy to(idle) D L[11] L[12] L[13] L[14] L[15] L[16] L[17] L[18] L[19] L[20]
                           L[21] L[22] L[23] L[24]
 aut C11 stt idle to(busy) L[11]
         stt busy to(idle) D L[12] L[13] L[14] L[15] L[16] L[17] L[18] L[19] L[20] L[21]
                           L[22] L[23] L[24]
 aut C12 stt idle to(busy) L[12]
         stt busy to(idle) D L[13] L[14] L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22]
                           L[23] L[24]
 aut C13 stt idle to(busy) L[13]
         stt busy to(idle) D L[14] L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C14 stt idle to(busy) L[14]
         stt busy to(idle) D L[15] L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C15 stt idle to(busy) L[15]
         stt busy to(idle) D L[16] L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C16 stt idle to(busy) L[16]
         stt busy to(idle) D L[17] L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C17 stt idle to(busy) L[17]
         stt busy to(idle) D L[18] L[19] L[20] L[21] L[22] L[23] L[24]
 aut C18 stt idle to(busy) L[18]
         stt busy to(idle) D L[19] L[20] L[21] L[22] L[23] L[24]
 aut C19 stt idle to(busy) L[19]
         stt busy to(idle) D L[20] L[21] L[22] L[23] L[24]
 aut C20 stt idle to(busy) L[20]
         stt busy to(idle) D L[21] L[22] L[23] L[24]
 aut C21 stt idle to(busy) L[21]
         stt busy to(idle) D L[22] L[23] L[24]
 aut C22 stt idle to(busy) L[22]
         stt busy to(idle) D L[23] L[24]
 aut C23 stt idle to(busy) L[23]
         stt busy to(idle) D L[24]
 aut C24 stt idle to(busy) L[24]
         stt busy to(idle) D

results
 full  = nb busy;
 use1  = st C1 == busy;
 use2  = st C2 == busy;
 use3  = st C3 == busy;
 average = nb busy;
```

A.29 LCS (GTA) - $Th_{23} = 15$ - $Th_{45} = 15$

```
identifiers
    K1 = [0..50];
    K2 = [0..25];
    K3 = [0..25];
    K4 = [0..25];
    K5 = [0..25];

    TH1 = 15;
    TH2 = 15;

    Lambda01 = 0.2;

    Mu01 = 1.0;
    Mu02 = 1.0;
    Mu03 = 2.0;
    Mu04 = 5.0;
    Mu05 = 1.0;

    pi12 = 0.6;
    pi14 = 0.4;
    pi23 = 0.7;
```

```
    pi25 = 0.3;
    pi43 = 0.5;
    pi45 = 0.5;

    f1 = (st Queue2 + st Queue3) < TH1;
    f2 = (st Queue4 + st Queue5) < TH2;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01 * pi12 * f1;
    rate_q1toq4    = Mu01 * pi14 * f2;
    rate_q2toq3    = Mu02 * pi23;
    rate_q2toq5    = Mu02 * pi25 * f2;
    rate_q3toOUT   = Mu03;
    rate_q4toq3    = Mu04 * pi43 * f1;
    rate_q4toq5    = Mu04 * pi45;
    rate_q5toOUT   = Mu05;

events
    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e14 (rate_q1toq4);
    syn e23 (rate_q2toq3);
    syn e25 (rate_q2toq5);
    loc e3  (rate_q3toOUT);
    syn e43 (rate_q4toq3);
    syn e45 (rate_q4toq5);
    loc e5  (rate_q5toOUT);

reachability = ((st Queue2 + st Queue3) <= TH1) && ((st Queue4 + st Queue5) <= TH2);

network LCS (continuous)
  aut Queue1
    stt n[K1]  to(++) e1
               to(--) e12 e14

  aut Queue2
    stt n[K2]  to(++) e12
               to(--) e23 e25

  aut Queue3
    stt n[K3]  to(++) e23 e43
               to(--) e3

  aut Queue4
    stt n[K4]  to(++) e14
               to(--) e43 e45

  aut Queue5
    stt n[K5]  to(++) e25 e45
               to(--) e5

results
  queue1 = st Queue1;
  queue2 = st Queue2;
  queue3 = st Queue3;
  queue4 = st Queue4;
  queue5 = st Queue5;
```

A.30 LCS (GTA) - $Th_{23} = 25$ - $Th_{45} = 25$

```
identifiers
    K1 = [0..50];
    K2 = [0..25];
    K3 = [0..25];
    K4 = [0..25];
    K5 = [0..25];

    TH1 = 25;
    TH2 = 25;
```

```
10
11      Lambda01 = 0.2;
12
13      Mu01 = 1.0;
14      Mu02 = 1.0;
15      Mu03 = 2.0;
16      Mu04 = 5.0;
17      Mu05 = 1.0;
18
19      pi12 = 0.6;
20      pi14 = 0.4;
21      pi23 = 0.7;
22      pi25 = 0.3;
23      pi43 = 0.5;
24      pi45 = 0.5;
25
26      f1 = (st Queue2 + st Queue3) < TH1;
27      f2 = (st Queue4 + st Queue5) < TH2;
28
29      rate_arrival01 = Lambda01;
30      rate_q1toq2    = Mu01 * pi12 * f1;
31      rate_q1toq4    = Mu01 * pi14 * f2;
32      rate_q2toq3    = Mu02 * pi23;
33      rate_q2toq5    = Mu02 * pi25 * f2;
34      rate_q3toOUT   = Mu03;
35      rate_q4toq3    = Mu04 * pi43 * f1;
36      rate_q4toq5    = Mu04 * pi45;
37      rate_q5toOUT   = Mu05;
38
39  events
40      loc e1  (rate_arrival01);
41      syn e12 (rate_q1toq2);
42      syn e14 (rate_q1toq4);
43      syn e23 (rate_q2toq3);
44      syn e25 (rate_q2toq5);
45      loc e3  (rate_q3toOUT);
46      syn e43 (rate_q4toq3);
47      syn e45 (rate_q4toq5);
48      loc e5  (rate_q5toOUT);
49
50  reachability = ((st Queue2 + st Queue3) <= TH1) && ((st Queue4 + st Queue5) <= TH2);
51
52  network LCS (continuous)
53    aut Queue1
54      stt n[K1]  to(++) e1
55                 to(--) e12 e14
56
57    aut Queue2
58      stt n[K2]  to(++) e12
59                 to(--) e23 e25
60
61    aut Queue3
62      stt n[K3]  to(++) e23 e43
63                 to(--) e3
64
65    aut Queue4
66      stt n[K4]  to(++) e14
67                 to(--) e43 e45
68
69    aut Queue5
70      stt n[K5]  to(++) e25 e45
71                 to(--) e5
72
73  results
74    queue1 = st Queue1;
75    queue2 = st Queue2;
76    queue3 = st Queue3;
77    queue4 = st Queue4;
78    queue5 = st Queue5;
79
```

A.31 LCS (GTA) - $Th_{23} = 35$ - $Th_{45} = 35$

```
identifiers
    K1 = [0..50];
    K2 = [0..25];
    K3 = [0..25];
    K4 = [0..25];
    K5 = [0..25];

    TH1 = 35;
    TH2 = 35;

    Lambda01 = 0.2;

    Mu01 = 1.0;
    Mu02 = 1.0;
    Mu03 = 2.0;
    Mu04 = 5.0;
    Mu05 = 1.0;

    pi12 = 0.6;
    pi14 = 0.4;
    pi23 = 0.7;
    pi25 = 0.3;
    pi43 = 0.5;
    pi45 = 0.5;

    f1 = (st Queue2 + st Queue3) < TH1;
    f2 = (st Queue4 + st Queue5) < TH2;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01 * pi12 * f1;
    rate_q1toq4    = Mu01 * pi14 * f2;
    rate_q2toq3    = Mu02 * pi23;
    rate_q2toq5    = Mu02 * pi25 * f2;
    rate_q3toOUT   = Mu03;
    rate_q4toq3    = Mu04 * pi43 * f1;
    rate_q4toq5    = Mu04 * pi45;
    rate_q5toOUT   = Mu05;

events
    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e14 (rate_q1toq4);
    syn e23 (rate_q2toq3);
    syn e25 (rate_q2toq5);
    loc e3  (rate_q3toOUT);
    syn e43 (rate_q4toq3);
    syn e45 (rate_q4toq5);
    loc e5  (rate_q5toOUT);

reachability = ((st Queue2 + st Queue3) <= TH1) && ((st Queue4 + st Queue5) <= TH2);

network LCS (continuous)
  aut Queue1
    stt n[K1]  to(++) e1
               to(--) e12 e14

  aut Queue2
    stt n[K2]  to(++) e12
               to(--) e23 e25

  aut Queue3
    stt n[K3]  to(++) e23 e43
               to(--) e3

  aut Queue4
    stt n[K4]  to(++) e14
               to(--) e43 e45

  aut Queue5
```

```
    stt n[K5]  to(++) e25 e45
               to(--) e5

results
  queue1 = st Queue1;
  queue2 = st Queue2;
  queue3 = st Queue3;
  queue4 = st Queue4;
  queue5 = st Queue5;

```

A.32 LCS (GTA) - $Th_{23} = 45$ - $Th_{45} = 45$

```
identifiers
    K1 = [0..50];
    K2 = [0..25];
    K3 = [0..25];
    K4 = [0..25];
    K5 = [0..25];

    TH1 = 45;
    TH2 = 45;

    Lambda01 = 0.2;

    Mu01 = 1.0;
    Mu02 = 1.0;
    Mu03 = 2.0;
    Mu04 = 5.0;
    Mu05 = 1.0;

    pi12 = 0.6;
    pi14 = 0.4;
    pi23 = 0.7;
    pi25 = 0.3;
    pi43 = 0.5;
    pi45 = 0.5;

    f1 = (st Queue2 + st Queue3) < TH1;
    f2 = (st Queue4 + st Queue5) < TH2;

    rate_arrival01 = Lambda01;
    rate_q1toq2    = Mu01 * pi12 * f1;
    rate_q1toq4    = Mu01 * pi14 * f2;
    rate_q2toq3    = Mu02 * pi23;
    rate_q2toq5    = Mu02 * pi25 * f2;
    rate_q3toOUT   = Mu03;
    rate_q4toq3    = Mu04 * pi43 * f1;
    rate_q4toq5    = Mu04 * pi45;
    rate_q5toOUT   = Mu05;

events
    loc e1  (rate_arrival01);
    syn e12 (rate_q1toq2);
    syn e14 (rate_q1toq4);
    syn e23 (rate_q2toq3);
    syn e25 (rate_q2toq5);
    loc e3  (rate_q3toOUT);
    syn e43 (rate_q4toq3);
    syn e45 (rate_q4toq5);
    loc e5  (rate_q5toOUT);

reachability = ((st Queue2 + st Queue3) <= TH1) && ((st Queue4 + st Queue5) <= TH2);

network LCS (continuous)
  aut Queue1
    stt n[K1]  to(++) e1
               to(--) e12 e14

  aut Queue2
```

```
58      stt n[K2]  to(++) e12
59                 to(--) e23 e25
60
61    aut Queue3
62      stt n[K3]  to(++) e23 e43
63                 to(--) e3
64
65    aut Queue4
66      stt n[K4]  to(++) e14
67                 to(--) e43 e45
68
69    aut Queue5
70      stt n[K5]  to(++) e25 e45
71                 to(--) e5
72
73  results
74    queue1 = st Queue1;
75    queue2 = st Queue2;
76    queue3 = st Queue3;
77    queue4 = st Queue4;
78    queue5 = st Queue5;
79
```

B Examples - SMART

B.1 OQN - $K_1 = 250$ - $K_2 = 100$ - $K_3 = 100$

```
1   print("*** Open Queueing Network with Loss and Blocking (K1=250, K2=100, K3=100) ***\n\n");
2
3   spn OQN := {
4
5     int K1 := 250;
6     int K2 := 100;
7     int K3 := 100;
8
9     int C2 := 2;
10
11    real rt1    := 2.0;
12    real rt1_23 := 0.4;
13    real rt2    := 0.5;
14    real rt3    := 0.2;
15
16    real pi1 := 0.7;
17    real pi2 := 0.3;
18
19    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3;
20
21    trans t1, t12, t13, t13L, t2, t3;
22
23    guard(t13L:tk(CQ3)==K3);
24
25    firing(t1:expo(rt1), t12:expo(rt1_23*pi1), t13:expo(rt1_23*pi2), t13L:expo(rt1_23*pi2),
26           t2:expo(min(tk(CQ2),C2)*rt2), t3:expo(rt3));
27
28    arcs(AQ1:t1,  t1:CQ1,
29         CQ1:t12, t12:AQ1, AQ2:t12, t12:CQ2,
30         CQ1:t13, t13:AQ1, CQ1:t13L, t13L:AQ1, AQ3:t13, t13:CQ3,
31         CQ2:t2,  t2:AQ2,
32         CQ3:t3,  t3:AQ3);
33
34    init(AQ1:K1, AQ2:K2, AQ3:K3);
35
36    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3);
37
38    real Q1 := avg_ss(tk(CQ1));
39    real Q2 := avg_ss(tk(CQ2));
40    real Q3 := avg_ss(tk(CQ3));
41
42  };
```

```
43
44  print("Average number of customers in Q1: ", OQN.Q1, "\n");
45  print("Average number of customers in Q2: ", OQN.Q2, "\n");
46  print("Average number of customers in Q3: ", OQN.Q3, "\n\n");
```

B.2 OQN - $K_1 = 250$ - $K_2 = 150$ - $K_3 = 150$

```
1   print("*** Open Queueing Network with Loss and Blocking (K1=250, K2=150, K3=150) ***\n\n");
2
3   spn OQN := {
4
5     int K1 := 250;
6     int K2 := 150;
7     int K3 := 150;
8
9     int C2 := 2;
10
11    real rt1    := 2.0;
12    real rt1_23 := 0.4;
13    real rt2    := 0.5;
14    real rt3    := 0.2;
15
16    real pi1 := 0.7;
17    real pi2 := 0.3;
18
19    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3;
20
21    trans t1, t12, t13, t13L, t2, t3;
22
23    guard(t13L:tk(CQ3)==K3);
24
25    firing(t1:expo(rt1), t12:expo(rt1_23*pi1), t13:expo(rt1_23*pi2), t13L:expo(rt1_23*pi2),
26           t2:expo(min(tk(CQ2),C2)*rt2), t3:expo(rt3));
27
28    arcs(AQ1:t1,  t1:CQ1,
29         CQ1:t12, t12:AQ1, AQ2:t12, t12:CQ2,
30         CQ1:t13, t13:AQ1, CQ1:t13L, t13L:AQ1, AQ3:t13, t13:CQ3,
31         CQ2:t2,  t2:AQ2,
32         CQ3:t3,  t3:AQ3);
33
34    init(AQ1:K1, AQ2:K2, AQ3:K3);
35
36    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3);
37
38    real Q1 := avg_ss(tk(CQ1));
39    real Q2 := avg_ss(tk(CQ2));
40    real Q3 := avg_ss(tk(CQ3));
41
42  };
43
44  print("Average number of customers in Q1: ", OQN.Q1, "\n");
45  print("Average number of customers in Q2: ", OQN.Q2, "\n");
46  print("Average number of customers in Q3: ", OQN.Q3, "\n\n");
```

B.3 OQN - $K_1 = 250$ - $K_2 = 200$ - $K_3 = 200$

```
1   print("*** Open Queueing Network with Loss and Blocking (K1=250, K2=200, K3=200) ***\n\n");
2
3   spn OQN := {
4
5     int K1 := 250;
6     int K2 := 200;
7     int K3 := 200;
8
9     int C2 := 2;
10
11    real rt1    := 2.0;
12    real rt1_23 := 0.4;
13    real rt2    := 0.5;
14    real rt3    := 0.2;
```

```

  real pi1 := 0.7;
  real pi2 := 0.3;

  place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3;

  trans t1, t12, t13, t13L, t2, t3;

  guard(t13L:tk(CQ3)==K3);

  firing(t1:expo(rt1), t12:expo(rt1_23*pi1), t13:expo(rt1_23*pi2), t13L:expo(rt1_23*pi2),
         t2:expo(min(tk(CQ2),C2)*rt2), t3:expo(rt3));

  arcs(AQ1:t1,  t1:CQ1,
       CQ1:t12, t12:AQ1, AQ2:t12, t12:CQ2,
       CQ1:t13, t13:AQ1, CQ1:t13L, t13L:AQ1, AQ3:t13, t13:CQ3,
       CQ2:t2,  t2:AQ2,
       CQ3:t3,  t3:AQ3);

  init(AQ1:K1, AQ2:K2, AQ3:K3);

  partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3);

  real Q1 := avg_ss(tk(CQ1));
  real Q2 := avg_ss(tk(CQ2));
  real Q3 := avg_ss(tk(CQ3));

};

print("Average number of customers in Q1: ", OQN.Q1, "\n");
print("Average number of customers in Q2: ", OQN.Q2, "\n");
print("Average number of customers in Q3: ", OQN.Q3, "\n\n");
```

B.4 OQN - $K_1 = 250$ - $K_2 = 250$ - $K_3 = 250$

```
print("*** Open Queueing Network with Loss and Blocking (K1=250, K2=250, K3=250) ***\n\n");

spn OQN := {

  int K1 := 250;
  int K2 := 250;
  int K3 := 250;

  int C2 := 2;

  real rt1    := 2.0;
  real rt1_23 := 0.4;
  real rt2    := 0.5;
  real rt3    := 0.2;

  real pi1 := 0.7;
  real pi2 := 0.3;

  place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3;

  trans t1, t12, t13, t13L, t2, t3;

  guard(t13L:tk(CQ3)==K3);

  firing(t1:expo(rt1), t12:expo(rt1_23*pi1), t13:expo(rt1_23*pi2), t13L:expo(rt1_23*pi2),
         t2:expo(min(tk(CQ2),C2)*rt2), t3:expo(rt3));

  arcs(AQ1:t1,  t1:CQ1,
       CQ1:t12, t12:AQ1, AQ2:t12, t12:CQ2,
       CQ1:t13, t13:AQ1, CQ1:t13L, t13L:AQ1, AQ3:t13, t13:CQ3,
       CQ2:t2,  t2:AQ2,
       CQ3:t3,  t3:AQ3);

  init(AQ1:K1, AQ2:K2, AQ3:K3);

```

```
36    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3);
37
38    real Q1 := avg_ss(tk(CQ1));
39    real Q2 := avg_ss(tk(CQ2));
40    real Q3 := avg_ss(tk(CQ3));
41
42  };
43
44  print("Average number of customers in Q1: ", OQN.Q1, "\n");
45  print("Average number of customers in Q2: ", OQN.Q2, "\n");
46  print("Average number of customers in Q3: ", OQN.Q3, "\n\n");
```

B.5 RS - $N = 24$ - $R = 3$

```
1   print("*** Resource Sharing (24 processes and 3 resources) ***\n");
2
3   spn RS(int p, int r) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..p-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15    place RS, RU;
16    partition(RS:RU);
17    init(RS:r);
18
19    for (int i in {0..p-1}) {
20        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i],
21             RS:ta[i], ta[i]:RU, RU:tr[i], tr[i]:RS);
22    }
23
24    real avg_U1 := avg_ss(tk(U[0]));
25    real avg_U2 := avg_ss(tk(U[1]));
26    real avg_U3 := avg_ss(tk(U[2]));
27
28    real avg_R0 := prob_ss(tk(RU)==0);
29    real avg_R1 := prob_ss(tk(RU)==1);
30    real avg_R2 := prob_ss(tk(RU)==2);
31    real avg_RU := avg_ss(tk(RU));
32  };
33
34  // Number of process
35  int P := 24;
36  // Number of resources
37  int R := 3;
38
39  print("Resource Sharing model with process P=", P, " and resources R=", R,"\n");
40
41  print("Average number of tokens in U1: ", RS(P,R).avg_U1, "\n");
42  print("Average number of tokens in U2: ", RS(P,R).avg_U2, "\n");
43  print("Average number of tokens in U3: ", RS(P,R).avg_U3, "\n");
44  print("Average number of tokens in R0: ", RS(P,R).avg_R0, "\n");
45  print("Average number of tokens in R1: ", RS(P,R).avg_R1, "\n");
46  print("Average number of tokens in R2: ", RS(P,R).avg_R2, "\n");
47  print("Average number of tokens in RU: ", RS(P,R).avg_RU, "\n");
```

B.6 RS - $N = 24$ - $R = 9$

```
1   print("*** Resource Sharing (24 processes and 9 resources) ***\n");
2
3   spn RS(int p, int r) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
```

```
7
8     for (int i in {0..p-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15    place RS, RU;
16    partition(RS:RU);
17    init(RS:r);
18
19    for (int i in {0..p-1}) {
20        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i],
21             RS:ta[i], ta[i]:RU, RU:tr[i], tr[i]:RS);
22    }
23
24    real avg_U1 := avg_ss(tk(U[0]));
25    real avg_U2 := avg_ss(tk(U[1]));
26    real avg_U3 := avg_ss(tk(U[2]));
27
28    real avg_R0 := prob_ss(tk(RU)==0);
29    real avg_R1 := prob_ss(tk(RU)==1);
30    real avg_R2 := prob_ss(tk(RU)==2);
31    real avg_RU := avg_ss(tk(RU));
32  };
33
34  // Number of process
35  int P := 24;
36  // Number of resources
37  int R := 9;
38
39  print("Resource Sharing model with process P=", P, " and resources R=", R,"\n");
40
41  print("Average number of tokens in U1: ", RS(P,R).avg_U1, "\n");
42  print("Average number of tokens in U2: ", RS(P,R).avg_U2, "\n");
43  print("Average number of tokens in U3: ", RS(P,R).avg_U3, "\n");
44  print("Average number of tokens in R0: ", RS(P,R).avg_R0, "\n");
45  print("Average number of tokens in R1: ", RS(P,R).avg_R1, "\n");
46  print("Average number of tokens in R2: ", RS(P,R).avg_R2, "\n");
47  print("Average number of tokens in RU: ", RS(P,R).avg_RU, "\n");
```

B.7 RS - $N = 24$ - $R = 15$

```
1   print("*** Resource Sharing (24 processes and 15 resources) ***\n");
2
3   spn RS(int p, int r) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..p-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15    place RS, RU;
16    partition(RS:RU);
17    init(RS:r);
18
19    for (int i in {0..p-1}) {
20        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i],
21             RS:ta[i], ta[i]:RU, RU:tr[i], tr[i]:RS);
22    }
23
24    real avg_U1 := avg_ss(tk(U[0]));
25    real avg_U2 := avg_ss(tk(U[1]));
26    real avg_U3 := avg_ss(tk(U[2]));
```

```
27
28    real avg_R0 := prob_ss(tk(RU)==0);
29    real avg_R1 := prob_ss(tk(RU)==1);
30    real avg_R2 := prob_ss(tk(RU)==2);
31    real avg_RU := avg_ss(tk(RU));
32  };
33
34  // Number of process
35  int P := 24;
36  // Number of resources
37  int R := 15;
38
39  print("Resource Sharing model with process P=", P, " and resources R=", R,"\n");
40
41  print("Average number of tokens in U1: ", RS(P,R).avg_U1, "\n");
42  print("Average number of tokens in U2: ", RS(P,R).avg_U2, "\n");
43  print("Average number of tokens in U3: ", RS(P,R).avg_U3, "\n");
44  print("Average number of tokens in R0: ", RS(P,R).avg_R0, "\n");
45  print("Average number of tokens in R1: ", RS(P,R).avg_R1, "\n");
46  print("Average number of tokens in R2: ", RS(P,R).avg_R2, "\n");
47  print("Average number of tokens in RU: ", RS(P,R).avg_RU, "\n");
```

B.8 RS - $N = 24$ - $R = 21$

```
1   print("*** Resource Sharing (24 processes and 21 resources) ***\n");
2
3   spn RS(int p, int r) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..p-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15    place RS, RU;
16    partition(RS:RU);
17    init(RS:r);
18
19    for (int i in {0..p-1}) {
20        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i],
21             RS:ta[i], ta[i]:RU, RU:tr[i], tr[i]:RS);
22    }
23
24    real avg_U1 := avg_ss(tk(U[0]));
25    real avg_U2 := avg_ss(tk(U[1]));
26    real avg_U3 := avg_ss(tk(U[2]));
27
28    real avg_R0 := prob_ss(tk(RU)==0);
29    real avg_R1 := prob_ss(tk(RU)==1);
30    real avg_R2 := prob_ss(tk(RU)==2);
31    real avg_RU := avg_ss(tk(RU));
32  };
33
34  // Number of process
35  int P := 24;
36  // Number of resources
37  int R := 21;
38
39  print("Resource Sharing model with process P=", P, " and resources R=", R,"\n");
40
41  print("Average number of tokens in U1: ", RS(P,R).avg_U1, "\n");
42  print("Average number of tokens in U2: ", RS(P,R).avg_U2, "\n");
43  print("Average number of tokens in U3: ", RS(P,R).avg_U3, "\n");
44  print("Average number of tokens in R0: ", RS(P,R).avg_R0, "\n");
45  print("Average number of tokens in R1: ", RS(P,R).avg_R1, "\n");
46  print("Average number of tokens in R2: ", RS(P,R).avg_R2, "\n");
```

```
47  print("Average number of tokens in RU: ", RS(P,R).avg_RU, "\n");
```

B.9 ASP - $P = 2$

```
1   print(" *** Alternate Service Pattern (2 patterns) *** \n");
2
3   spn asp := {
4
5     int K1 := 3;
6     int K2 := 3;
7     int K3 := 10;
8     int K4 := 10;
9
10    real rt1    := 4.0;
11    real rt2    := 1.0;
12    real rt13   := 3.0;
13    real rt23   := 2.0;
14    real rt34_1 := 6.0;
15    real rt34_2 := 3.0;
16    real rt4    := 5.0;
17
18    real pi1 := 0.4;
19    real pi2 := 0.6;
20
21    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3, AQ4, CQ4, P1, P2;
22
23    trans t1, t2, t2L, t13, t23, t4, t34_11, t34_12, t34_21, t34_22;
24
25    guard(t2L:tk(CQ3)==K3);
26
27    firing(t1:expo(rt1),
28           t2:expo(rt2),
29           t13:expo(rt13), t23:expo(rt23), t2L:expo(rt23),
30           t4:expo(rt4),
31           t34_11:expo(rt34_1*pi1), t34_12:expo(rt34_1*pi2),
32           t34_21:expo(rt34_2*pi1), t34_22:expo(rt34_2*pi2));
33
34    arcs(AQ1:t1, t1:CQ1,
35         AQ2:t2, t2:CQ2,
36         CQ1:t13, t13:AQ1, CQ2:t23, t23:AQ2, CQ2:t2L, t2L:AQ2,
37         AQ3:t13, t13:CQ3, AQ3:t23, t23:CQ3,
38         CQ4:t4, t4:AQ4,
39         CQ3:t34_11, CQ3:t34_12,
40         CQ3:t34_21, CQ3:t34_22,
41         t34_11:AQ3, t34_12:AQ3,
42         t34_21:AQ3, t34_22:AQ3,
43         P1:t34_11, t34_11:P1,
44         P1:t34_12, t34_12:P2,
45         P2:t34_21, t34_21:P1,
46         P2:t34_22, t34_22:P2,
47         AQ4:t34_11, AQ4:t34_12, AQ4:t34_21, AQ4:t34_22,
48         t34_11:CQ4, t34_12:CQ4, t34_21:CQ4, t34_22:CQ4);
49
50    init(AQ1:K1, AQ2:K2, AQ3:K3, AQ4:K4, P1:1);
51
52    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3, AQ4:CQ4, P1:P2);
53
54    real avg_CQ1 := avg_ss(tk(CQ1));
55    real avg_CQ2 := avg_ss(tk(CQ2));
56    real avg_CQ3 := avg_ss(tk(CQ3));
57    real avg_CQ4 := avg_ss(tk(CQ4));
58    real avg_P1 := avg_ss(tk(P1));
59    real avg_P2 := avg_ss(tk(P2));
60  };
61
62  print("Average number of tokens in Queue 1: ", asp.avg_CQ1, "\n");
63  print("Average number of tokens in Queue 2: ", asp.avg_CQ2, "\n");
64  print("Average number of tokens in Queue 3: ", asp.avg_CQ3, "\n");
65  print("Average number of tokens in Queue 4: ", asp.avg_CQ4, "\n");
66  print("Average number of tokens in P1: ", asp.avg_P1, "\n");
```

```
67  print("Average number of tokens in P2: ", asp.avg_P2, "\n");
```

B.10 ASP - $P = 3$

```
1   print(" *** Alternate Service Pattern (3 patterns) *** \n");
2
3   spn asp := {
4
5     int K1 := 3;
6     int K2 := 3;
7     int K3 := 10;
8     int K4 := 10;
9
10    real rt1    := 4.0;
11    real rt2    := 1.0;
12    real rt13   := 3.0;
13    real rt23   := 2.0;
14    real rt34_1 := 6.0;
15    real rt34_2 := 3.0;
16    real rt34_3 := 2.0;
17    real rt4    := 5.0;
18
19    real pi1 := 0.4;
20    real pi2 := 0.3;
21    real pi3 := 0.3;
22
23    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3, AQ4, CQ4, P1, P2, P3;
24
25    trans t1, t2, t2L, t13, t23, t4,
26          t34_11, t34_12, t34_13,
27          t34_21, t34_22, t34_23,
28          t34_31, t34_32, t34_33;
29
30    guard(t2L:tk(CQ3)==K3);
31
32    firing(t1:expo(rt1),
33           t2:expo(rt2),
34           t13:expo(rt13), t23:expo(rt23), t2L:expo(rt23),
35           t4:expo(rt4),
36           t34_11:expo(rt34_1*pi1), t34_12:expo(rt34_1*pi2), t34_13:expo(rt34_1*pi3),
37           t34_21:expo(rt34_2*pi1), t34_22:expo(rt34_2*pi2), t34_23:expo(rt34_2*pi3),
38           t34_31:expo(rt34_3*pi1), t34_32:expo(rt34_3*pi2), t34_33:expo(rt34_3*pi3));
39
40    arcs(AQ1:t1, t1:CQ1,
41         AQ2:t2, t2:CQ2,
42         CQ1:t13, t13:AQ1, CQ2:t23, t23:AQ2, CQ2:t2L, t2L:AQ2,
43         AQ3:t13, t13:CQ3, AQ3:t23, t23:CQ3,
44         CQ4:t4, t4:AQ4,
45         CQ3:t34_11, CQ3:t34_12, CQ3:t34_13,
46         CQ3:t34_21, CQ3:t34_22, CQ3:t34_23,
47         CQ3:t34_31, CQ3:t34_32, CQ3:t34_33,
48         t34_11:AQ3, t34_12:AQ3, t34_13:AQ3,
49         t34_21:AQ3, t34_22:AQ3, t34_23:AQ3,
50         t34_31:AQ3, t34_32:AQ3, t34_33:AQ3,
51         P1:t34_11, t34_11:P1,
52         P1:t34_12, t34_12:P2,
53         P1:t34_13, t34_13:P3,
54         P2:t34_21, t34_21:P1,
55         P2:t34_22, t34_22:P2,
56         P2:t34_23, t34_23:P3,
57         P3:t34_31, t34_31:P1,
58         P3:t34_32, t34_32:P2,
59         P3:t34_33, t34_33:P3,
60         AQ4:t34_11, AQ4:t34_12, AQ4:t34_13,
61         AQ4:t34_21, AQ4:t34_22, AQ4:t34_23,
62         AQ4:t34_31, AQ4:t34_32, AQ4:t34_33,
63         t34_11:CQ4, t34_12:CQ4, t34_13:CQ4,
64         t34_21:CQ4, t34_22:CQ4, t34_23:CQ4,
65         t34_31:CQ4, t34_32:CQ4, t34_33:CQ4);
66
```

```
67    init(AQ1:K1, AQ2:K2, AQ3:K3, AQ4:K4, P1:1);
68
69    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3, AQ4:CQ4, P1:P2:P3);
70
71    real avg_CQ1 := avg_ss(tk(CQ1));
72    real avg_CQ2 := avg_ss(tk(CQ2));
73    real avg_CQ3 := avg_ss(tk(CQ3));
74    real avg_CQ4 := avg_ss(tk(CQ4));
75    real avg_P1 := avg_ss(tk(P1));
76    real avg_P2 := avg_ss(tk(P2));
77    real avg_P3 := avg_ss(tk(P3));
78  };
79
80  print("Average number of tokens in Queue 1: ", asp.avg_CQ1, "\n");
81  print("Average number of tokens in Queue 2: ", asp.avg_CQ2, "\n");
82  print("Average number of tokens in Queue 3: ", asp.avg_CQ3, "\n");
83  print("Average number of tokens in Queue 4: ", asp.avg_CQ4, "\n");
84  print("Average number of tokens in P1: ", asp.avg_P1, "\n");
85  print("Average number of tokens in P2: ", asp.avg_P2, "\n");
86  print("Average number of tokens in P3: ", asp.avg_P3, "\n");
```

B.11 ASP - $P = 4$

```
1   print(" *** Alternate Service Pattern (4 patterns) *** \n");
2
3   spn asp := {
4
5     int K1 := 3;
6     int K2 := 3;
7     int K3 := 10;
8     int K4 := 10;
9
10    real rt1    := 4.0;
11    real rt2    := 1.0;
12    real rt13   := 3.0;
13    real rt23   := 2.0;
14    real rt34_1 := 6.0;
15    real rt34_2 := 3.0;
16    real rt34_3 := 2.0;
17    real rt34_4 := 1.0;
18    real rt4    := 5.0;
19
20    real pi1 := 0.4;
21    real pi2 := 0.3;
22    real pi3 := 0.15;
23    real pi4 := 0.15;
24
25    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3, AQ4, CQ4, P1, P2, P3, P4;
26
27    trans t1, t2, t2L, t13, t23, t4,
28          t34_11, t34_12, t34_13, t34_14,
29          t34_21, t34_22, t34_23, t34_24,
30          t34_31, t34_32, t34_33, t34_34,
31          t34_41, t34_42, t34_43, t34_44;
32
33    guard(t2L:tk(CQ3)==K3);
34
35    firing(t1:expo(rt1),
36           t2:expo(rt2),
37           t13:expo(rt13), t23:expo(rt23), t2L:expo(rt23),
38           t4:expo(rt4),
39           t34_11:expo(rt34_1*pi1), t34_12:expo(rt34_1*pi2),
40           t34_13:expo(rt34_1*pi3), t34_14:expo(rt34_1*pi4),
41           t34_21:expo(rt34_2*pi1), t34_22:expo(rt34_2*pi2),
42           t34_23:expo(rt34_2*pi3), t34_24:expo(rt34_2*pi4),
43           t34_31:expo(rt34_3*pi1), t34_32:expo(rt34_3*pi2),
44           t34_33:expo(rt34_3*pi3), t34_34:expo(rt34_3*pi4),
45           t34_41:expo(rt34_4*pi1), t34_42:expo(rt34_4*pi2),
46           t34_43:expo(rt34_4*pi3), t34_44:expo(rt34_4*pi4));
47
```

```
48    arcs(AQ1:t1, t1:CQ1,
49         AQ2:t2, t2:CQ2,
50         CQ1:t13, t13:AQ1, CQ2:t23, t23:AQ2, CQ2:t2L, t2L:AQ2,
51         AQ3:t13, t13:CQ3, AQ3:t23, t23:CQ3,
52         CQ4:t4, t4:AQ4,
53         CQ3:t34_11, CQ3:t34_12, CQ3:t34_13, CQ3:t34_14,
54         CQ3:t34_21, CQ3:t34_22, CQ3:t34_23, CQ3:t34_24,
55         CQ3:t34_31, CQ3:t34_32, CQ3:t34_33, CQ3:t34_34,
56         CQ3:t34_41, CQ3:t34_42, CQ3:t34_43, CQ3:t34_44,
57         t34_11:AQ3, t34_12:AQ3, t34_13:AQ3, t34_14:AQ3,
58         t34_21:AQ3, t34_22:AQ3, t34_23:AQ3, t34_24:AQ3,
59         t34_31:AQ3, t34_32:AQ3, t34_33:AQ3, t34_34:AQ3,
60         t34_41:AQ3, t34_42:AQ3, t34_43:AQ3, t34_44:AQ3,
61         P1:t34_11, t34_11:P1,
62         P1:t34_12, t34_12:P2,
63         P1:t34_13, t34_13:P3,
64         P1:t34_14, t34_14:P4,
65         P2:t34_21, t34_21:P1,
66         P2:t34_22, t34_22:P2,
67         P2:t34_23, t34_23:P3,
68         P2:t34_24, t34_24:P4,
69         P3:t34_31, t34_31:P1,
70         P3:t34_32, t34_32:P2,
71         P3:t34_33, t34_33:P3,
72         P3:t34_34, t34_34:P4,
73         P4:t34_41, t34_41:P1,
74         P4:t34_42, t34_42:P2,
75         P4:t34_43, t34_43:P3,
76         P4:t34_44, t34_44:P4,
77         AQ4:t34_11, AQ4:t34_12, AQ4:t34_13, AQ4:t34_14,
78         AQ4:t34_21, AQ4:t34_22, AQ4:t34_23, AQ4:t34_24,
79         AQ4:t34_31, AQ4:t34_32, AQ4:t34_33, AQ4:t34_34,
80         AQ4:t34_41, AQ4:t34_42, AQ4:t34_43, AQ4:t34_44,
81         t34_11:CQ4, t34_12:CQ4, t34_13:CQ4, t34_14:CQ4,
82         t34_21:CQ4, t34_22:CQ4, t34_23:CQ4, t34_24:CQ4,
83         t34_31:CQ4, t34_32:CQ4, t34_33:CQ4, t34_34:CQ4,
84         t34_41:CQ4, t34_42:CQ4, t34_43:CQ4, t34_44:CQ4);
85
86    init(AQ1:K1, AQ2:K2, AQ3:K3, AQ4:K4, P1:1);
87
88    partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3, AQ4:CQ4, P1:P2:P3:P4);
89
90    real avg_CQ1 := avg_ss(tk(CQ1));
91    real avg_CQ2 := avg_ss(tk(CQ2));
92    real avg_CQ3 := avg_ss(tk(CQ3));
93    real avg_CQ4 := avg_ss(tk(CQ4));
94    real avg_P1 := avg_ss(tk(P1));
95    real avg_P2 := avg_ss(tk(P2));
96    real avg_P3 := avg_ss(tk(P3));
97    real avg_P4 := avg_ss(tk(P4));
98  };
99
100 print("Average number of tokens in Queue 1: ", asp.avg_CQ1, "\n");
101 print("Average number of tokens in Queue 2: ", asp.avg_CQ2, "\n");
102 print("Average number of tokens in Queue 3: ", asp.avg_CQ3, "\n");
103 print("Average number of tokens in Queue 4: ", asp.avg_CQ4, "\n");
104 print("Average number of tokens in P1: ", asp.avg_P1, "\n");
105 print("Average number of tokens in P2: ", asp.avg_P2, "\n");
106 print("Average number of tokens in P3: ", asp.avg_P3, "\n");
107 print("Average number of tokens in P4: ", asp.avg_P4, "\n");
```

B.12 ASP - $P = 5$

```
1  print(" *** Alternate Service Pattern (5 patterns) *** \n");
2
3  spn asp := {
4
5    int K1 := 3;
6    int K2 := 3;
7    int K3 := 10;
```

```
8     int K4 := 10;
9
10    real rt1    := 4.0;
11    real rt2    := 1.0;
12    real rt13   := 3.0;
13    real rt23   := 2.0;
14    real rt34_1 := 6.0;
15    real rt34_2 := 3.0;
16    real rt34_3 := 2.0;
17    real rt34_4 := 1.0;
18    real rt34_5 := 4.0;
19    real rt4    := 5.0;
20
21    real pi1 := 0.4;
22    real pi2 := 0.3;
23    real pi3 := 0.15;
24    real pi4 := 0.1;
25    real pi5 := 0.05;
26
27    place AQ1, CQ1, AQ2, CQ2, AQ3, CQ3, AQ4, CQ4, P1, P2, P3, P4, P5;
28
29    trans t1, t2, t2L, t13, t23, t4,
30          t34_11, t34_12, t34_13, t34_14, t34_15,
31          t34_21, t34_22, t34_23, t34_24, t34_25,
32          t34_31, t34_32, t34_33, t34_34, t34_35,
33          t34_41, t34_42, t34_43, t34_44, t34_45,
34          t34_51, t34_52, t34_53, t34_54, t34_55;
35
36    guard(t2L:tk(CQ3)==K3);
37
38    firing(t1:expo(rt1),
39           t2:expo(rt2),
40           t13:expo(rt13), t23:expo(rt23), t2L:expo(rt23),
41           t4:expo(rt4),
42           t34_11:expo(rt34_1*pi1), t34_12:expo(rt34_1*pi2), t34_13:expo(rt34_1*pi3),
43           t34_14:expo(rt34_1*pi4), t34_15:expo(rt34_1*pi5),
44           t34_21:expo(rt34_2*pi1), t34_22:expo(rt34_2*pi2), t34_23:expo(rt34_2*pi3),
45           t34_24:expo(rt34_2*pi4), t34_25:expo(rt34_2*pi5),
46           t34_31:expo(rt34_3*pi1), t34_32:expo(rt34_3*pi2), t34_33:expo(rt34_3*pi3),
47           t34_34:expo(rt34_3*pi4), t34_35:expo(rt34_3*pi5),
48           t34_41:expo(rt34_4*pi1), t34_42:expo(rt34_4*pi2), t34_43:expo(rt34_4*pi3),
49           t34_44:expo(rt34_4*0.1), t34_45:expo(rt34_4*pi5),
50           t34_51:expo(rt34_5*pi1), t34_52:expo(rt34_5*pi2), t34_53:expo(rt34_5*pi3),
51           t34_54:expo(rt34_5*pi4), t34_55:expo(rt34_5*pi5));
52
53    arcs(AQ1:t1, t1:CQ1,
54         AQ2:t2, t2:CQ2,
55         CQ1:t13, t13:AQ1, CQ2:t23, t23:AQ2, CQ2:t2L, t2L:AQ2,
56         AQ3:t13, t13:CQ3, AQ3:t23, t23:CQ3,
57         CQ4:t4, t4:AQ4,
58         CQ3:t34_11, CQ3:t34_12, CQ3:t34_13, CQ3:t34_14, CQ3:t34_15,
59         CQ3:t34_21, CQ3:t34_22, CQ3:t34_23, CQ3:t34_24, CQ3:t34_25,
60         CQ3:t34_31, CQ3:t34_32, CQ3:t34_33, CQ3:t34_34, CQ3:t34_35,
61         CQ3:t34_41, CQ3:t34_42, CQ3:t34_43, CQ3:t34_44, CQ3:t34_45,
62         CQ3:t34_51, CQ3:t34_52, CQ3:t34_53, CQ3:t34_54, CQ3:t34_55,
63         t34_11:AQ3, t34_12:AQ3, t34_13:AQ3, t34_14:AQ3, t34_15:AQ3,
64         t34_21:AQ3, t34_22:AQ3, t34_23:AQ3, t34_24:AQ3, t34_25:AQ3,
65         t34_31:AQ3, t34_32:AQ3, t34_33:AQ3, t34_34:AQ3, t34_35:AQ3,
66         t34_41:AQ3, t34_42:AQ3, t34_43:AQ3, t34_44:AQ3, t34_45:AQ3,
67         t34_51:AQ3, t34_52:AQ3, t34_53:AQ3, t34_54:AQ3, t34_55:AQ3,
68         P1:t34_11, t34_11:P1,
69         P1:t34_12, t34_12:P2,
70         P1:t34_13, t34_13:P3,
71         P1:t34_14, t34_14:P4,
72         P1:t34_15, t34_15:P5,
73         P2:t34_21, t34_21:P1,
74         P2:t34_22, t34_22:P2,
75         P2:t34_23, t34_23:P3,
76         P2:t34_24, t34_24:P4,
77         P2:t34_25, t34_25:P5,
78         P3:t34_31, t34_31:P1,
```

```
        P3:t34_32, t34_32:P2,
        P3:t34_33, t34_33:P3,
        P3:t34_34, t34_34:P4,
        P3:t34_35, t34_35:P5,
        P4:t34_41, t34_41:P1,
        P4:t34_42, t34_42:P2,
        P4:t34_43, t34_43:P3,
        P4:t34_44, t34_44:P4,
        P4:t34_45, t34_45:P5,
        P5:t34_51, t34_51:P1,
        P5:t34_52, t34_52:P2,
        P5:t34_53, t34_53:P3,
        P5:t34_54, t34_54:P4,
        P5:t34_55, t34_55:P5,
        AQ4:t34_11, AQ4:t34_12, AQ4:t34_13, AQ4:t34_14, AQ4:t34_15,
        AQ4:t34_21, AQ4:t34_22, AQ4:t34_23, AQ4:t34_24, AQ4:t34_25,
        AQ4:t34_31, AQ4:t34_32, AQ4:t34_33, AQ4:t34_34, AQ4:t34_35,
        AQ4:t34_41, AQ4:t34_42, AQ4:t34_43, AQ4:t34_44, AQ4:t34_45,
        AQ4:t34_51, AQ4:t34_52, AQ4:t34_53, AQ4:t34_54, AQ4:t34_55,
        t34_11:CQ4, t34_12:CQ4, t34_13:CQ4, t34_14:CQ4, t34_15:CQ4,
        t34_21:CQ4, t34_22:CQ4, t34_23:CQ4, t34_24:CQ4, t34_25:CQ4,
        t34_31:CQ4, t34_32:CQ4, t34_33:CQ4, t34_34:CQ4, t34_35:CQ4,
        t34_41:CQ4, t34_42:CQ4, t34_43:CQ4, t34_44:CQ4, t34_45:CQ4,
        t34_51:CQ4, t34_52:CQ4, t34_53:CQ4, t34_54:CQ4, t34_55:CQ4);

  init(AQ1:K1, AQ2:K2, AQ3:K3, AQ4:K4, P1:1);

  partition(AQ1:CQ1, AQ2:CQ2, AQ3:CQ3, AQ4:CQ4, P1:P2:P3:P4:P5);

  real avg_CQ1 := avg_ss(tk(CQ1));
  real avg_CQ2 := avg_ss(tk(CQ2));
  real avg_CQ3 := avg_ss(tk(CQ3));
  real avg_CQ4 := avg_ss(tk(CQ4));
  real avg_P1 := avg_ss(tk(P1));
  real avg_P2 := avg_ss(tk(P2));
  real avg_P3 := avg_ss(tk(P3));
  real avg_P4 := avg_ss(tk(P4));
  real avg_P5 := avg_ss(tk(P5));
};

print("Average number of tokens in Queue 1: ", asp.avg_CQ1, "\n");
print("Average number of tokens in Queue 2: ", asp.avg_CQ2, "\n");
print("Average number of tokens in Queue 3: ", asp.avg_CQ3, "\n");
print("Average number of tokens in Queue 4: ", asp.avg_CQ4, "\n");
print("Average number of tokens in P1: ", asp.avg_P1, "\n");
print("Average number of tokens in P2: ", asp.avg_P2, "\n");
print("Average number of tokens in P3: ", asp.avg_P3, "\n");
print("Average number of tokens in P4: ", asp.avg_P4, "\n");
print("Average number of tokens in P5: ", asp.avg_P5, "\n");
```

B.13 FAS - $N = 6$

```
print(" *** First Available Service (6 servers) *** \n");

spn fas(int N) := {

  real rta := 1.0;
  real rtr := 0.25;

  for (int i in {0..N-1}) {
    place S[i], U[i];
    partition(S[i]:U[i]);
    trans tr[i], ta[i];
    firing(tr[i]:expo(rtr), ta[i]:expo(rta));
    init(S[i]:1);
  }

  for (int i in {0..N-1}) {
      arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i]);
  }
```

```
19    for (int i in {0..N-1}) {
20      for (int j in {i+1..N-1}) {
21        arcs(U[i]:ta[j], ta[j]:U[i]);
22      }
23    }
24
25    real avg_U1 := avg_ss(tk(U[0]));
26    real avg_U2 := avg_ss(tk(U[1]));
27    real avg_U3 := avg_ss(tk(U[2]));
28  };
29
30  // Number of servers
31  int N := 6;
32
33  print("Average number of tokens in U1: ", fas(N).avg_U1, "\n");
34  print("Average number of tokens in U2: ", fas(N).avg_U2, "\n");
35  print("Average number of tokens in U3: ", fas(N).avg_U3, "\n");
```

B.14 FAS - $N = 12$

```
1   print(" *** First Available Service (12 servers) *** \n");
2
3   spn fas(int N) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..N-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15
16    for (int i in {0..N-1}) {
17        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i]);
18    }
19    for (int i in {0..N-1}) {
20      for (int j in {i+1..N-1}) {
21        arcs(U[i]:ta[j], ta[j]:U[i]);
22      }
23    }
24
25    real avg_U1 := avg_ss(tk(U[0]));
26    real avg_U2 := avg_ss(tk(U[1]));
27    real avg_U3 := avg_ss(tk(U[2]));
28  };
29
30  // Number of servers
31  int N := 12;
32
33  print("Average number of tokens in U1: ", fas(N).avg_U1, "\n");
34  print("Average number of tokens in U2: ", fas(N).avg_U2, "\n");
35  print("Average number of tokens in U3: ", fas(N).avg_U3, "\n");
```

B.15 FAS - $N = 18$

```
1   print(" *** First Available Service (18 servers) *** \n");
2
3   spn fas(int N) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..N-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
```

```
13      init(S[i]:1);
14    }
15
16    for (int i in {0..N-1}) {
17        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i]);
18    }
19    for (int i in {0..N-1}) {
20      for (int j in {i+1..N-1}) {
21        arcs(U[i]:ta[j], ta[j]:U[i]);
22      }
23    }
24
25    real avg_U1 := avg_ss(tk(U[0]));
26    real avg_U2 := avg_ss(tk(U[1]));
27    real avg_U3 := avg_ss(tk(U[2]));
28  };
29
30  // Number of servers
31  int N := 18;
32
33  print("Average number of tokens in U1: ", fas(N).avg_U1, "\n");
34  print("Average number of tokens in U2: ", fas(N).avg_U2, "\n");
35  print("Average number of tokens in U3: ", fas(N).avg_U3, "\n");
```

B.16 FAS - $N = 24$

```
1   print(" *** First Available Service (24 servers) *** \n");
2
3   spn fas(int N) := {
4
5     real rta := 1.0;
6     real rtr := 0.25;
7
8     for (int i in {0..N-1}) {
9       place S[i], U[i];
10      partition(S[i]:U[i]);
11      trans tr[i], ta[i];
12      firing(tr[i]:expo(rtr), ta[i]:expo(rta));
13      init(S[i]:1);
14    }
15
16    for (int i in {0..N-1}) {
17        arcs(S[i]:ta[i], ta[i]:U[i], U[i]:tr[i], tr[i]:S[i]);
18    }
19    for (int i in {0..N-1}) {
20      for (int j in {i+1..N-1}) {
21        arcs(U[i]:ta[j], ta[j]:U[i]);
22      }
23    }
24
25    real avg_U1 := avg_ss(tk(U[0]));
26    real avg_U2 := avg_ss(tk(U[1]));
27    real avg_U3 := avg_ss(tk(U[2]));
28  };
29
30  // Number of servers
31  int N := 24;
32
33  print("Average number of tokens in U1: ", fas(N).avg_U1, "\n");
34  print("Average number of tokens in U2: ", fas(N).avg_U2, "\n");
35  print("Average number of tokens in U3: ", fas(N).avg_U3, "\n");
```

B.17 LCS - $Th_{23} = 15$ - $Th_{45} = 15$

```
1   print(" *** Open Queueing Networks with Limited Capacity Subnets (Thresholds: 15) *** \n");
2
3   spn lcs := {
4
5     int TH1 := 15;
6     int TH2 := 15;
```

```
7
8     int K1 := 50;
9     int K2 := 25;
10    int K3 := 25;
11    int K4 := 25;
12    int K5 := 25;
13
14    real rt1    := 0.2;
15    real rt1_24 := 1.0;
16    real rt2_35 := 1.0;
17    real rt3    := 2.0;
18    real rt4_35 := 5.0;
19    real rt5    := 1.0;
20
21    real pi12 := 0.6;
22    real pi14 := 0.4;
23    real pi23 := 0.7;
24    real pi25 := 0.3;
25    real pi43 := 0.5;
26    real pi45 := 0.5;
27
28    place A1, Q1, A2, Q2, A3, Q3, A4, Q4, A5, Q5;
29
30    trans t1, t12, t14, t23, t25, t3, t43, t45, t5;
31
32    guard(t12:(tk(Q2)+tk(Q3))<TH1,
33          t14:(tk(Q4)+tk(Q5))<TH2,
34          t25:(tk(Q4)+tk(Q5))<TH2,
35          t43:(tk(Q2)+tk(Q3))<TH1);
36
37    firing(t1:expo(rt1), t12:expo(rt1_24*pi12), t14:expo(rt1_24*pi14),
38           t23:expo(rt2_35*pi23), t25:expo(rt2_35*pi25), t3:expo(rt3),
39           t43:expo(rt4_35*pi43), t45:expo(rt4_35*pi45), t5:expo(rt5));
40
41    arcs(A1:t1, t1:Q1,
42         Q1:t12, Q1:t14, t12: A1, t14:A1,
43         A2:t12, t12:Q2, A4:t14, t14:Q4,
44         Q2:t23, t23:A2, Q2:t25, t25:A2,
45         Q4:t43, t43:A4, Q4:t45, t45:A4,
46         A3:t23, t23:Q3, A3:t43, t43:Q3,
47         A5:t25, t25:Q5, A5:t45, t45:Q5,
48         Q3:t3, t3:A3, Q5:t5, t5:A5);
49
50    init(A1:K1, A2:K2, A3:K3, A4:K4, A5:K5);
51
52    partition(A1:Q1, A2:Q2:A3:Q3, A4:Q4:A5:Q5);
53
54    real avg_Q1 := avg_ss(tk(Q1));
55    real avg_Q2 := avg_ss(tk(Q2));
56    real avg_Q3 := avg_ss(tk(Q3));
57    real avg_Q4 := avg_ss(tk(Q4));
58    real avg_Q5 := avg_ss(tk(Q5));
59  };
60
61  print("Average number of tokens in Queue 1: ", lcs.avg_Q1, "\n");
62  print("Average number of tokens in Queue 2: ", lcs.avg_Q2, "\n");
63  print("Average number of tokens in Queue 3: ", lcs.avg_Q3, "\n");
64  print("Average number of tokens in Queue 4: ", lcs.avg_Q4, "\n");
65  print("Average number of tokens in Queue 5: ", lcs.avg_Q5, "\n");
```

B.18 LCS - $Th_{23} = 25$ - $Th_{45} = 25$

```
1   print(" *** Open Queueing Networks with Limited Capacity Subnets (Thresholds: 25) *** \n");
2
3   spn lcs := {
4
5     int TH1 := 25;
6     int TH2 := 25;
7
8     int K1 := 50;
```

```
  int K2 := 25;
  int K3 := 25;
  int K4 := 25;
  int K5 := 25;

  real rt1    := 0.2;
  real rt1_24 := 1.0;
  real rt2_35 := 1.0;
  real rt3    := 2.0;
  real rt4_35 := 5.0;
  real rt5    := 1.0;

  real pi12 := 0.6;
  real pi14 := 0.4;
  real pi23 := 0.7;
  real pi25 := 0.3;
  real pi43 := 0.5;
  real pi45 := 0.5;

  place A1, Q1, A2, Q2, A3, Q3, A4, Q4, A5, Q5;

  trans t1, t12, t14, t23, t25, t3, t43, t45, t5;

  guard(t12:(tk(Q2)+tk(Q3))<TH1,
        t14:(tk(Q4)+tk(Q5))<TH2,
        t25:(tk(Q4)+tk(Q5))<TH2,
        t43:(tk(Q2)+tk(Q3))<TH1);

  firing(t1:expo(rt1), t12:expo(rt1_24*pi12), t14:expo(rt1_24*pi14),
         t23:expo(rt2_35*pi23), t25:expo(rt2_35*pi25), t3:expo(rt3),
         t43:expo(rt4_35*pi43), t45:expo(rt4_35*pi45), t5:expo(rt5));

  arcs(A1:t1, t1:Q1,
       Q1:t12, Q1:t14, t12: A1, t14:A1,
       A2:t12, t12:Q2, A4:t14, t14:Q4,
       Q2:t23, t23:A2, Q2:t25, t25:A2,
       Q4:t43, t43:A4, Q4:t45, t45:A4,
       A3:t23, t23:Q3, A3:t43, t43:Q3,
       A5:t25, t25:Q5, A5:t45, t45:Q5,
       Q3:t3, t3:A3, Q5:t5, t5:A5);

  init(A1:K1, A2:K2, A3:K3, A4:K4, A5:K5);

  partition(A1:Q1, A2:Q2:A3:Q3, A4:Q4:A5:Q5);

  real avg_Q1 := avg_ss(tk(Q1));
  real avg_Q2 := avg_ss(tk(Q2));
  real avg_Q3 := avg_ss(tk(Q3));
  real avg_Q4 := avg_ss(tk(Q4));
  real avg_Q5 := avg_ss(tk(Q5));
};

print("Average number of tokens in Queue 1: ", lcs.avg_Q1, "\n");
print("Average number of tokens in Queue 2: ", lcs.avg_Q2, "\n");
print("Average number of tokens in Queue 3: ", lcs.avg_Q3, "\n");
print("Average number of tokens in Queue 4: ", lcs.avg_Q4, "\n");
print("Average number of tokens in Queue 5: ", lcs.avg_Q5, "\n");
```

B.19 LCS - $Th_{23} = 35$ - $Th_{45} = 35$

```
print(" *** Open Queueing Networks with Limited Capacity Subnets (Thresholds: 35) *** \n");

spn lcs := {

  int TH1 := 35;
  int TH2 := 35;

  int K1 := 50;
  int K2 := 25;
  int K3 := 25;
```

```
11    int K4 := 25;
12    int K5 := 25;
13
14    real rt1    := 0.2;
15    real rt1_24 := 1.0;
16    real rt2_35 := 1.0;
17    real rt3    := 2.0;
18    real rt4_35 := 5.0;
19    real rt5    := 1.0;
20
21    real pi12 := 0.6;
22    real pi14 := 0.4;
23    real pi23 := 0.7;
24    real pi25 := 0.3;
25    real pi43 := 0.5;
26    real pi45 := 0.5;
27
28    place A1, Q1, A2, Q2, A3, Q3, A4, Q4, A5, Q5;
29
30    trans t1, t12, t14, t23, t25, t3, t43, t45, t5;
31
32    guard(t12:(tk(Q2)+tk(Q3))<TH1,
33          t14:(tk(Q4)+tk(Q5))<TH2,
34          t25:(tk(Q4)+tk(Q5))<TH2,
35          t43:(tk(Q2)+tk(Q3))<TH1);
36
37    firing(t1:expo(rt1), t12:expo(rt1_24*pi12), t14:expo(rt1_24*pi14),
38           t23:expo(rt2_35*pi23), t25:expo(rt2_35*pi25), t3:expo(rt3),
39           t43:expo(rt4_35*pi43), t45:expo(rt4_35*pi45), t5:expo(rt5));
40
41    arcs(A1:t1, t1:Q1,
42         Q1:t12, Q1:t14, t12: A1, t14:A1,
43         A2:t12, t12:Q2, A4:t14, t14:Q4,
44         Q2:t23, t23:A2, Q2:t25, t25:A2,
45         Q4:t43, t43:A4, Q4:t45, t45:A4,
46         A3:t23, t23:Q3, A3:t43, t43:Q3,
47         A5:t25, t25:Q5, A5:t45, t45:Q5,
48         Q3:t3, t3:A3, Q5:t5, t5:A5);
49
50    init(A1:K1, A2:K2, A3:K3, A4:K4, A5:K5);
51
52    partition(A1:Q1, A2:Q2:A3:Q3, A4:Q4:A5:Q5);
53
54    real avg_Q1 := avg_ss(tk(Q1));
55    real avg_Q2 := avg_ss(tk(Q2));
56    real avg_Q3 := avg_ss(tk(Q3));
57    real avg_Q4 := avg_ss(tk(Q4));
58    real avg_Q5 := avg_ss(tk(Q5));
59  };
60
61  print("Average number of tokens in Queue 1: ", lcs.avg_Q1, "\n");
62  print("Average number of tokens in Queue 2: ", lcs.avg_Q2, "\n");
63  print("Average number of tokens in Queue 3: ", lcs.avg_Q3, "\n");
64  print("Average number of tokens in Queue 4: ", lcs.avg_Q4, "\n");
65  print("Average number of tokens in Queue 5: ", lcs.avg_Q5, "\n");
```

B.20 LCS - $Th_{23} = 45$ - $Th_{45} = 45$

```
1   print(" *** Open Queueing Networks with Limited Capacity Subnets (Thresholds: 45) *** \n");
2
3   spn lcs := {
4
5     int TH1 := 45;
6     int TH2 := 45;
7
8     int K1 := 50;
9     int K2 := 25;
10    int K3 := 25;
11    int K4 := 25;
12    int K5 := 25;
```

```
13
14    real rt1    := 0.2;
15    real rt1_24 := 1.0;
16    real rt2_35 := 1.0;
17    real rt3    := 2.0;
18    real rt4_35 := 5.0;
19    real rt5    := 1.0;
20
21    real pi12 := 0.6;
22    real pi14 := 0.4;
23    real pi23 := 0.7;
24    real pi25 := 0.3;
25    real pi43 := 0.5;
26    real pi45 := 0.5;
27
28    place A1, Q1, A2, Q2, A3, Q3, A4, Q4, A5, Q5;
29
30    trans t1, t12, t14, t23, t25, t3, t43, t45, t5;
31
32    guard(t12:(tk(Q2)+tk(Q3))<TH1,
33          t14:(tk(Q4)+tk(Q5))<TH2,
34          t25:(tk(Q4)+tk(Q5))<TH2,
35          t43:(tk(Q2)+tk(Q3))<TH1);
36
37    firing(t1:expo(rt1), t12:expo(rt1_24*pi12), t14:expo(rt1_24*pi14),
38           t23:expo(rt2_35*pi23), t25:expo(rt2_35*pi25), t3:expo(rt3),
39           t43:expo(rt4_35*pi43), t45:expo(rt4_35*pi45), t5:expo(rt5));
40
41    arcs(A1:t1, t1:Q1,
42         Q1:t12, Q1:t14, t12: A1, t14:A1,
43         A2:t12, t12:Q2, A4:t14, t14:Q4,
44         Q2:t23, t23:A2, Q2:t25, t25:A2,
45         Q4:t43, t43:A4, Q4:t45, t45:A4,
46         A3:t23, t23:Q3, A3:t43, t43:Q3,
47         A5:t25, t25:Q5, A5:t45, t45:Q5,
48         Q3:t3, t3:A3, Q5:t5, t5:A5);
49
50    init(A1:K1, A2:K2, A3:K3, A4:K4, A5:K5);
51
52    partition(A1:Q1, A2:Q2:A3:Q3, A4:Q4:A5:Q5);
53
54    real avg_Q1 := avg_ss(tk(Q1));
55    real avg_Q2 := avg_ss(tk(Q2));
56    real avg_Q3 := avg_ss(tk(Q3));
57    real avg_Q4 := avg_ss(tk(Q4));
58    real avg_Q5 := avg_ss(tk(Q5));
59  };
60
61  print("Average number of tokens in Queue 1: ", lcs.avg_Q1, "\n");
62  print("Average number of tokens in Queue 2: ", lcs.avg_Q2, "\n");
63  print("Average number of tokens in Queue 3: ", lcs.avg_Q3, "\n");
64  print("Average number of tokens in Queue 4: ", lcs.avg_Q4, "\n");
65  print("Average number of tokens in Queue 5: ", lcs.avg_Q5, "\n");
```