[6] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances on Petri Nets '84*, volume 188 of *LNCS*, pages 215–233. Springer Verlag, 1984.

[7] K. Jensen. Coloured Petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[8] J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrand, Princeton, NJ, 1960.

[9] Chuang Lin and Dan C. Marinescu. On stochastic high level Petri nets. In *Proc. Int. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.

is difficult to achieve automatically.

6 Conclusions

In this paper an extension to the SRG-based performance analysis technique has been presented for a class of SWN models called *quasi-lumpable* SWNs.

The idea behind this method came from the observation that a system may contain objects that behave homogeneously from a qualitative point of view but the symmetry disappears when quantitative aspects are taken into account. Using the bounds computation method presented in [4] it is possible to take advantage of the stronger aggregation achievable when only the qualitative behavior is taken into account. In this case bounds on the system performance indices are computed instead of exact results; this precision loss can be accepted whether the state space size reduction transforms a model whose state space is too huge to be analyzed into an analyzable model. Observe also that the fact that bounds are obtained instead of approximate results of uncertain precision makes the method more robust than approximation methods especially those that do not give error bounds.

We have shown two application examples, belonging to the class of SWN models just described. Future developments of this work will be in the direction of (1) devising new techniques to further improve the bounds, possibly by using some information from the high level model (e.g. from the SWN model structural analysis), (2) studying the sensitivity of performance measure bounds spread as a function of the difference $\mathcal{Q}^+ - \mathcal{Q}^-$, and (3) finding new model classes to which the method could apply, as for example those representing systems whose arrival/departure rates are state dependent and are a smooth function of the system population.

References

[1] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets and symmetric modeling. Accepted for publication on IEEE Transactions on Computers, 1992.

[2] P.J. Courtois and P. Semal. Computable bounds on conditional steady-state probabilities in large markov chains and queueing models. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):926–937, Sept. 1986.

[3] C. Dutheillet and S. Haddad. Regular stochastic Petri nets. In *Proc. 10th Intern. Conf. Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[4] G. Franceschinis and R. Muntz. Bounds for quasi-lumpable Markov chains. In *Proc. of Performance 93*, Rome, Italy, September 1993.

[5] S. Haddad. *Une Categorie Regulier de Reseau de Petri de Haut Niveau: Definition, Proprietes et Reductions.* PhD thesis, Lab. MASI, Universite P. et M. Curie (Paris 6), Paris, France, Oct 1987. These de Doctorat, RR87/197 (in French).

The detailed state representation for this model is a list of program counter values, one for each process in the program. This SWN model is quasi-lumpable with respect to the color class of slaves: indeed the qualitative behavior of slaves is identical but the quantitative behavior is not because one slave is farther from the master. We thus aggregate all the states that are equal up to a permutation of slaves identities. It is immediate to verify that the static subclasses $S1$ and $S2$ could be merged. Concerning the symbolic firing instances to be considered for the computation of the lower bound aggregate matrix, we have only to take into account transitions $endRx$ and $endSnd$. From the structure of the net it is possible to know in advance that only instances of kind $[endRx, Z_S^j]$ and $[endSnd, Z_S^j]$ with $|Z_S^j| = 1$ are possible so that only two cases have to be considered: $Z_S^j \in S1$ or $Z_S^j \in S2$ so that $min_rate = com_rate_1$ and $max_rate = com_rate_2$ (where com_rate_i is the inverse of the mean time required for a communication between processing nodes at distance i).

The performance measures we have considered are three: (i) throughput of the system, (THRUPUT); (ii) mean number of slaves waiting to receive a task from the master (MNSWRx); (iii) mean number of slaves waiting to send the result of their computation to the master (MNSWSnd).

We have done three experiments varying the values of rates associated with the communication between master and slaves (transitions $endRx$ and $endSnd$). We have fixed the values of $\theta(endRx, S2)$ and $\theta(endSnd, S2)$ to 0.5 and 3.5 respectively, while the rates for $(endRx, S1)$ and $(endSnd, S1)$ that have been used in the three experiments are the following:
$\theta(endRx, S1) = (1)\ 0.51, (2)\ 0.525, (3)\ 0.55$; $\theta(endSnd, S1) = (1)\ 3.57, (2)\ 3.675, (3)\ 3.85$.

	Perf. Ind.	Lower bnd	Upper bnd	Spread
1	THRUPUT	0.012396	0.014595	18%
	MNSWSnd	0.079165	0.137257	73%
	MNSWRx	2.547911	2.726692	7%
2	THRUPUT	0.011936	0.016718	40%
	MNSWSnd	0.056813	0.180464	218%
	MNSWRx	2.436226	2.802309	15%
3	THRUPUT	0.011333	0.019553	73%
	MNSWSnd	0.039295	0.257986	557%
	MNSWRx	2.249367	2.861513	27%

The results shown in the table above are obtained from the direct method. We have also applied the bounds improvement method: it resulted in tightened bounds in the first experiment but it did not give any substantial improvement in experiments 2 and 3.

Clearly there is a strict correlation between the difference $\mathcal{Q}^+ - \mathcal{Q}^-$ and the spread in the bounds. It might be worthwhile to assess a priori the sensitivity of the system on variations of the transition probabilities that are positive in $\mathcal{Q}^+ - \mathcal{Q}^-$.

In this example we have also observed that significant bounds refinement could be obtained by the consideration of the high level model (see [4]). Of course, the kind of property to be proved about the high level model in order to obtain improved estimates of $\mathcal{Q}^-$ and/or $\mathcal{Q}^+$ and the technique used to prove it depend heavily on the kind of high level formalism adopted and

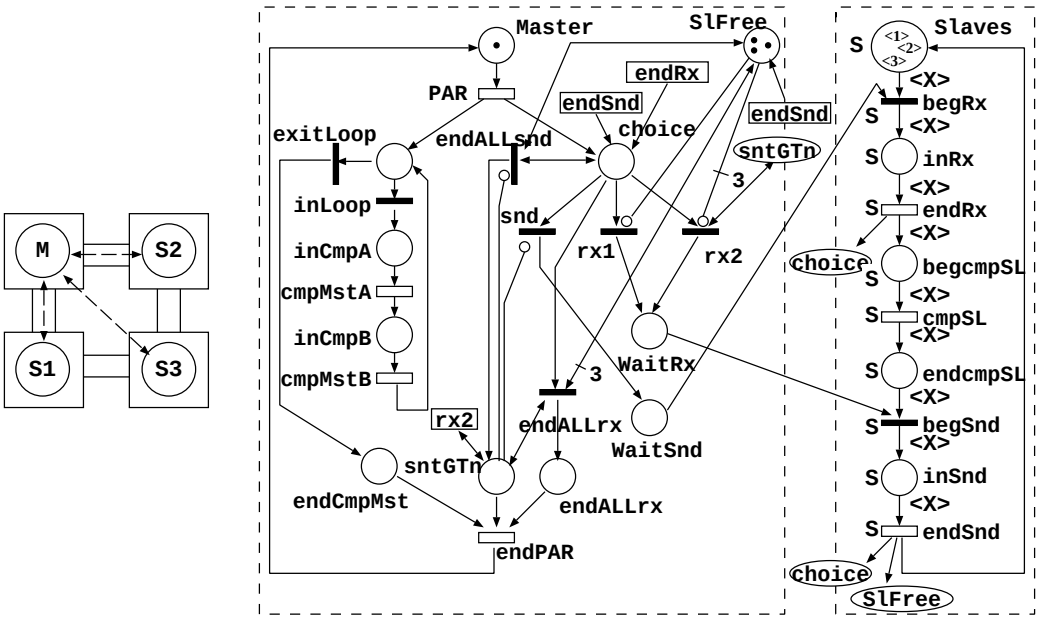


Figure 6: The SWN model of the Master-Slave Program.

by squares, while the processes are represented by circles. In the figure both the physical channels (connecting the processors) and the logical channels (connecting the processes) are depicted.

The master divides the problem to be solved into several subproblems, and as long as there are free slaves it delivers the tasks to be executed to them. When no more free slaves are available, the master waits for some slave to complete its current task. The master also performs some computation in parallel with communicating, to process the results received to prepare new data to be distributed. A SWN model of the master-slaves system is depicted in Figure 6. The model of the program resembles a flow-chart so that its semantics is rather intuitive. The transitions stand for statements in a process code. Places represent pointers to instructions in a given process code.

The model is composed of two subnets, the first (leftmost) one represents the master's behavior while the second (rightmost) one represent the common behavior of the slaves. The distinction among the three slaves is obtained by using *distinguishable* tokens, i.e., tokens in the slave behavior subnet are labeled with a slave identifier. The behavior model has been simplified as much as possible to make the picture readable. The master is composed of two parts that work in parallel: the computation part, consisting of a certain number of iterations of two procedures (transitions *inLoop*, *cmpMstA*, *cmpMstB*), and the synchronous communication part consisting of a certain number of iterations of send/receive operations (transitions *snd*, *rx1*, *rx2* plus the transitions representing actual communication, shared with the slaves net: *begRx* , *endRx*, *begSnd*, *endSnd*). The slaves behavior can be described as repeated iteration of three operations: receive (transitions *begRx* and *endRx*), compute (transition *compSL*), send (transitions *begSnd* and *endSnd*). There is only one color class needed: the slaves class S. It has cardinality three and is divided into two static subclasses $S1$ of cardinality 2 and $S2$ of cardinality 1. Subclass $S1$ represents the slaves that are closer to the master while subclass $S2$ represents the farther slave. As a consequence the transitions representing communication between the master and the slaves have a rate that depends on the slave identity.

$Make_choice$ and $Serv1$. We describe the transition rate bounds computation for transition $Serv1$ only, a similar argument applies to $Make_choice$. Let's consider the generic symbolic instance $[Serv1, Z^j_{Lines}]$. We have to consider four cases: $|Z^j_{Lines}| = k,\ k = 1, \ldots, 4$.
i) $|Z^j_{Lines}| = 1$: There are only two possibilities to take into account: $Z^j_{Lines} \in Lines1$ and $Z^j_{Lines} \in Lines2$. Hence $min_rate([Serv1, Z^j_{Lines}]) = \lambda_1$ and $max_rate([Serv1, Z^j_{Lines}]) = \lambda_2$.
ii) $|Z^j_{Lines}| = 2$: There are three possibilities to consider: (1) both objects in Z^j_{Lines} belong to $Lines1$, (2) both objects in Z^j_{Lines} belong to $Lines2$ or (3) one object belongs to $Lines1$, while the other one belongs to $Lines2$. Thus we have $min_rate([Serv1, Z^j_{Lines}]) = min(2\lambda_1, 2\lambda_2, \lambda_1 + \lambda_2) = 2\lambda_1$ while $max_rate([Serv1, Z^j_{Lines}]) = 2\lambda_2$.
iii) $|Z^j_{Lines}| = 3$: There are two possibilities to consider: two out of three objects belong to $Lines1$ and the remaining object belongs to $Lines2$ or vice versa. Hence $min_rate([Serv1, Z^j_{Lines}]) = min(\lambda_1 + 2\lambda_2, \lambda_2 + 2\lambda_1) = \lambda_2 + 2\lambda_1$ while $max_rate([Serv1, Z^j_{Lines}]) = \lambda_1 + 2\lambda_2$.
iv) $|Z^j_{Lines}| = 4$: There is only one possible assignment of objects to static subclasses in this cases: two objects belong to $Lines1$ while the other two belong to $Line2$ so that $min_rate([Serv1, Z^j_{Lines}]) = max_rate([Serv1, Z^j_{Lines}]) = 2\lambda_1 + 2\lambda_2$.

The bounds are obtained solving 17 aggregated CTMCs each comprising 18 states. The following table contains mean queue length (MQL), and throughput (THRU) bounds as well as the corresponding exact values.

	Lower bound	Exact value	Upper bound	Spread %
MQL	1.250705	1.273430	1.385868	11.0
THRU	0.967612	0.98284	0.983502	1.6

For this model it has been possible to get better bounds by applying the bounds improvement method cited in Section 3:

	Lower bound	Exact value	Upper bound	Spread %
MQL	1.250705	1.273430	1.295075	3.5
THRU	0.980303	0.98284	0.983502	0.3

The simplicity of this example is such that an intuitive argument could be found that would easily allow to compute mean queue length and throughput bounds for this model. In next section a more complex example is presented in which the presence of synchronization among the system components makes it difficult to find any intuitive argument for bounds computation.

5.2 A simple concurrent program

In this section we present a more complex system consisting of a parallel program organized according to a master-slave computation paradigm. The program is mapped on a parallel architecture: we assume there is a processor for each process, that the processors are homogeneous, and that due to the type of the interconnection network the communication time between the master and the slaves is not homogeneous. This is the case for the example of a master-slave program mapped onto a mesh architecture depicted in Figure 6. The processors are represented

• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle] + [Compute, \langle Z_P^{1a}, Z_J^{1b}\rangle]$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_s$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; $rate = 2 \cdot 0.3 + 2 \cdot 0.15 = 0.9$
• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle] + [Compute, \langle Z_P^{1a}, Z_J^{1b}\rangle]$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; $rate = 2 \cdot 1 + 2 \cdot 0.5 = 3$
• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle]$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_s$, $|Z_J^{1a}| = 2$ and $Z_J^{1a} \in J_s$; $rate = 4 \cdot 0.3 = 1.2$
• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle]$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 2$ and $Z_J^{1a} \in J_s$; $rate = 4 \cdot 1 = 4$
• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle] + [Compute, \langle Z_P^{1a}, Z_J^{1b}\rangle] + [Compute, \langle Z_P^{1b}, Z_J^{1a}\rangle] + [Compute, \langle Z_P^{1b}, Z_J^{1b}\rangle]$ with $|Z_P^{1a}| = 1$ and $Z_P^{1a} \in P_s$, $|Z_P^{1b}| = 1$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; $rate = 0.3 + 0.15 + 1 + 0.5 = 1.95$
• $[Compute, \langle Z_P^{1a}, Z_J^{1a}\rangle] + [Compute, \langle Z_P^{1b}, Z_J^{1a}\rangle]$ with $|Z_P^{1a}| = 1$ and $Z_P^{1a} \in P_s$, $|Z_P^{1b}| = 1$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 2$ and $Z_J^{1b} \in J_s$; $rate = 2 \cdot 0.3 + 2 \cdot 1 = 2.6$

There is a computationally less expensive but not always accurate method for computing lower/upper bounds for the elements of $\mathcal{Q}^-$ and $\mathcal{Q}^+$. Given a transition t and a tuple d of aggregate static subclasses, compute the Cartesian product D of the sets of original subclasses in the component aggregate subclasses. Associate with d the $min(max)_{d' \in D}\theta_t(d')$. In this way the computed $\mathcal{Q}^-$ and $\mathcal{Q}^+$ are bounds for $\mathcal{Q}$, but they can be very inaccurate in some cases. In the previous example this simplified method would have predicted correctly the maximum rate ($= 4$), while the minimum ($= 0.6$) would have been less than the the correct one (0.9).

Observe that even if the computation of the aggregate transition rates has a certain cost, it could be performed once and for all in a way that the result is easily reusable for models differing only in the actual parameter values. Moreover the computation has to be done only for the transition instances involving merged static subclasses: if the model has many transitions that do not contain the merged color class in their color domain definition there are good chances that the complex rates computation method has to be applied only to a minority of transition instances and that the corresponding overhead be compensated by the computation saving due to the stronger state aggregation.

5 Two examples

In this section the proposed method is applied to two example models: the first model represents the two servers in tandem system introduces in Section 2, while the second model represents a simple concurrent program mapped onto a parallel architecture.

5.1 Two servers in tandem

The model of Figure 2 is a good candidate for the application of the method. In fact this is a typical case in which the partition of the servers in two subclasses with equal speed within each subclass leads to a quasi-lumpable MC structure. We assume that the rates λ_1 and λ_2 associated with the two classes of servers are $\lambda_1 = 1.00$ and $\lambda_2 = 1.01$. Observe that in this case it is trivial to find mergeable static subclasses since neither functions of type $S_{subclass}$ appear on any arc nor predicates involving any static subclass are used.

Let's consider some transition instances to see how the lower bound matrix $\mathcal{Q}^-$ has been computed. There are two colored transitions that may be instanced to objects of *Lines*, namely

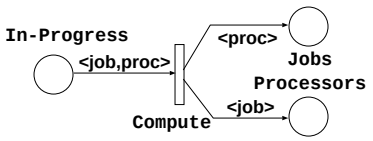


Figure 4: A simple SWN transition example.

When a symbolic transition instance $[t, z]$ is fired, its rate $\lambda([t, z])$ is computed as follows: (1) derive from z the tuple d of corresponding static subclasses (each dynamic subclass is associated with exactly one static subclass, see [1]); (2) $\lambda([t, z]) = m\ \theta_t(d)$ where m is a factor that depends on the cardinality of both the subclasses in z and in d. The reason for the multiplicative factor is that a symbolic firing instance is an aggregation of m ordinary firing instances all with the same rate $\theta_t(d)$. In our example a possible symbolic instance for transition $Compute$ could be $[Compute, \langle Z_P^1, Z_J^2 \rangle]$ with $|Z_P^1| = 2, Z_P^1 \in P_s$, and $|Z_J^2| = 1, Z_J^2 \in J_l$ meaning that there are $2\ (= |Z_P^1|)$ slow processors each of which is processing $1\ (= |Z_J^2|)$ long job. This symbolic instance stands for 2 "ordinary" instances that are grouped in the lumping process.

In the following we show how to derive the modified rate for a firing instance that involves objects belonging to some merged static subclass. We denote $D_{i,j}$ the new static subclasses and $\{D_{i,j}^l\}$ the set of the original subclasses that are aggregated into $D_{i,j}$. The rates of the resulting new transition instances, in general will depend on the cardinality of the dynamic subclasses in the color instance. Let $[t, z]$ be the symbolic transition instance for which a rate has to be computed. Let d be the associated static subclasses tuple. Let d' be the subtuple of d composed of merged static subclasses and z' the corresponding subtuple of z. For each element $d_k = D_{i,j}$ in d', compute the set $\mathcal{Z}_k$ of possible partitions of dynamic subclass z_k into one or more new dynamic subclasses, each associated with a different static subclass $D_{i,j}^l$ of $D_{i,j}$ (see Figure 5). The Cartesian product of the $\mathcal{Z}_k$s leads to a set of sums of original transition instances. From

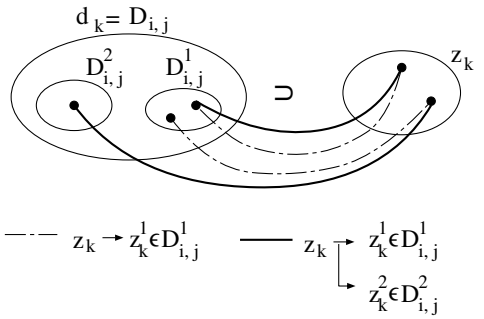


Figure 5: Possible partitions of z_k with respect to the "merged" static subclass $D_{i,j}$.

each sum a rate can be computed that corresponds to a value for a row sum in the aggregate transition matrix corresponding to $[t, z]$. The minimum and maximum in this set of rates gives the value for the corresponding transition rate in the aggregate matrices.

In the example of Figure 4 assume that $|P_s| = |P_f| = 2$ and $|J_s| = 2, |J_l| = 1$, and suppose we want to merge slow and fast processors into a unique class P and short and long jobs into a unique class J. Hence the symbolic transition instance $[Compute, \langle Z_P^1, Z_J^2 \rangle]$ with $|Z_P^1| = 2, Z_P^1 \in P$ and $|Z_J^2| = 2, Z_J^2 \in J$ incorporates the following six possible instances with respect to the old classes partition.

```
while old_static_i ≠ ∅ do
        ⟨ remove a subclass sc form old_static_i⟩
        lsc = sc
        sc_list = emptylist
        append(sc, sc_list)
        for each  sc' ∈ old_static_i do
                if semilumpable(sc', sc_list) then
                        lsc = lsc ∪ sc'
                        ⟨ remove sc' form old_static_i⟩
                        append(sc', sc_list)
                end if
        end for
        old_sc_list[lsc] = sc_list
        ⟨ add static subclass lsc to new_static_i⟩
end while
```

A set of static subclasses old_static_i is the input for this procedure. The output is a new set new_static_i, of static subclasses. The cardinality of the new set is less than or equal to that of the old set. For each new static subclass lsc, a list $old_sc_list[lsc]$ of the old static subclasses that have been merged into lsc is maintained.

Function $semilumpable(sc', sc_list)$ implements the most important part of the algorithm. Let ξ' be the set of symmetry functions that satisfy the constraint of allowing only permutation of objects within the same static subclass for all classes $C_j \neq C_i$, while allowing the exchange of objects in $\{sc'\} \cup sc_list$; function $semilumpable$ is meant to return true iff for any $s \in \xi'$, every predicate Φ occurring in the model satisfies the equation $\Phi(c) = \Phi(s.c)$ and every arc function f occurring in the model satisfies $f(s.c) = s.f(c)$. This is surely true when no references to the subclasses in $sc' \cup sc_list$ occur in any arc function and predicate of the model, and predicates do not contain clauses like $[d(x) = d(y)]$ (where x and y are transition parameters of type C_i). More complex sufficient conditions may be defined, for example predicates Φ/function elements f_i containing occurrences of the subclasses to be merged may be allowed if they have the form $\Phi = (\bigvee_{\forall sbc \in sc' \cup sc_list}[d(x) = sbc]) \wedge \Phi'$ and $f_i = \alpha \sum_{sbc \in\ sc' \cup sc_list} S_{sbc} + f_i'$ respectively, and the subclasses to be merged do not occurr in Φ' or in f_i', furthermore Φ' does not contain clauses like $[d(x) = d(y)]$.

Modified MC transition rates computation rule The aggregate matrices $\mathcal{Q}^-$ and $\mathcal{Q}^+$ are computed by applying the SRG generation algorithm to a modified model where static subclasses containing objects with the same qualitative behavior but different associated rates are merged.

In a SWN, the transition firing rate of a generic transition t is defined as a function θ_t from tuples of static color classes to reals. For example transition *Compute* in Figure 4 takes a token with its two component color from its input place: the first component represents a processor and the second represents a job. If the jobs class J is divided into two subclasses of short jobs J_s and long jobs J_l and the processors are divided into two subclasses of fast processors P_f and slow processors P_s, then a possible definition for the transition rate could be: $\theta_t(\langle < P_s, J_s > \rangle) = 0.3$, $\theta_t(\langle < P_f, J_s > \rangle) = 1$, $\theta_t(\langle < P_s, J_l > \rangle) = 0.15$, $\theta_t(\langle < P_f, J_l > \rangle) = 0.5$.

4 Automatic derivation of the aggregated MC for bounds computation

In order for the bounding method to be convenient (from a computational cost point of view), the ideal situation would be to derive the lumped matrices $\mathcal{Q}^-$ and $\mathcal{Q}^+$ directly from the high level model without having to compute (or at least to store) the large complete matrix. In this section we present a method that allows to reach this goal for a subclass of SWN models called *quasi-lumpable* SWNs.

The issue of automatic construction of the lumped matrix is closely related to that of the detection of symmetries on the state space that induce a partition into state aggregates. The same problem has to be faced when exact lumping methods are used, the above requirement of constructing automatically the aggregated CTMC for bounds computation, is similar to that of directly obtaining the lumped CTMC from the SRG. We now show that the same approach used to derive the lumped CTMC from the SRG can be extended to our bounding method, that is the upper and lower bound aggregated matrices $\mathcal{Q}^-$ and $\mathcal{Q}^+$ can be automatically built using a modified version of the SRG algorithm. The bounding matrices can be computed at different accuracy levels; as usual higher accuracy can be achieved with higher computational cost.

In Section 2 we introduced the concept of color classes and subclasses in SWN. The color classes are used to define sets of "similar" objects and the partition of a class into static subclasses is used to identify subsets of objects in a class that share the same behavior. As already pointed out, it is possible to distinguish between two possible situations: (1) the objects in different subclasses have different *qualitative* behavior, i.e., they cannot play the same "role" in the system because they have different possible evolutions, (2) the objects in different subclasses have the same *qualitative* behavior, but the event sequences happen with different rates depending on which subclass the object belongs to. For example, in the model of Figure 2 the colored tokens representing customers in the "multiserver" station follow the same route through transition $Serv1$, independently of their color, however, the firing rate of the transition depends on the token color.

Hence, given a SWN model of the system under study, a first analysis is needed to detect all the subclasses in each class with similar "qualitative" behavior. This permits automatic determination of the candidate quasi-lumpable state aggregates. Then two approaches are possible, (1) apply the usual Symbolic Reachability Graph generation algorithm to the system as it is specified and use the information about aggregate states to compute the lumped matrices $\mathcal{Q}^-$ and $\mathcal{Q}^+$ in a second step; (2) apply the SRG generation algorithm to a modified model where some of the homogeneously behaving subclasses are merged, and apply a modified CTMC transition rate computation rule to directly derive $\mathcal{Q}^-$ and $\mathcal{Q}^+$ from the SRG.

In the sequel we present the subclasses merge algorithm and the modified MC transition rate computation rule.

Subclasses merge algorithm For each basic color class, the following static classes merge procedure has to be performed:

old_static_i = ⟨set of static subclasses in the original C_i⟩
$new_static_i = \emptyset$

of bounds on the steady state probability vector of a large system by decomposition into smaller subsystems: this is called the *bounded aggregation method.* This method consists of two steps: (a) computation of bounds on the conditional state probabilities within each aggregate using the method just explained; (b) computation of bounds on the probability of being in each aggregate (this requires the derivation derivation of a lower bound $\mathcal{P}^-$ for the inter-aggregate probability matrix $\mathcal{P}$ using the results of previous step);

Finally the results obtained in the two steps can be combined to compute bounds on the steady state probability of the original states.

The proposed method The method proposed in [4] allows to compute bounds for quasi-lumpable Markov chains. We first give a definition of quasi-lumpable MC, then intuitively describe the bounding method.

Definition 2 *A Continuous Time Markov Chain is said to be* ϵ-quasi-lumpable *with respect to a given state space partition $\mathcal{A}$ if its infinitesimal generator Q can be rewritten as $Q = Q^- + Q^\epsilon$, where Q^- is a maximal lower bound (componentwise) for Q that satisfies the strong lumpability conditions [8] with respect to $\mathcal{A}$, and no element in Q^ϵ is greater than ϵ in value.*

The intention is that Q^ϵ is a matrix with many more zero elements than Q^- and with relatively small non-zero elements. Henceforth we use the term "quasi-lumpable CTMC" for a CTMC that is ϵ-quasi-lumpable for some ϵ.

The CTMC corresponding to the RG of the SWN model of Figure 2 when the servers in the multiserver station do not have all the same rate, is ϵ-quasi-lumpable with respect to the aggregation induced by the SRG generation algorithm when *Lines* is not partitioned into static subclasses; in this case ϵ is proportional to the difference $\lambda_2 - \lambda_1$.

The bounds computation method we have proposed in [4], can be described as a simplified version of the bounded aggregation method. Actually we apply only the second step of the bounded aggregation algorithm, because our aim is to deal directly with the lumped process. The lower bound $\mathcal{Q}^-$ for the aggregate matrix $\mathcal{Q}$ is obtained by computing the minimum row sum in each submatrix $Q_{i,j}$ of transition rates from states in the i^{th} aggregate to states in the j^{th} aggregate; from matrix $\mathcal{Q}^-$ lower and upper bounds on the aggregates steady state probability can be computed by means of the Courtois and Semal method. As we'll see later, matrix $\mathcal{Q}^-$ can be computed without ever computing the complete matrix Q. Observe that it is also possible to compute an upper bound $\mathcal{Q}^+$ for the transition rates between aggregates, by taking the maximum among the row sums of each submatrix. Using the following theorem it is possible to exploit the knowledge of an upper bound $\mathcal{Q}^+$ to get improved bounds:

Theorem 1 *[4] Let Q be the infinitesimal generator of an ergodic CTMC with n states. Let Q^- be a lower bound (componentwise) for Q, i.e., $Q^- \leq Q$. Let $\mathbf{y}^T = (Q - Q^-)\mathbf{e}^T$ and finally let $Q_s = Q^- + \mathbf{y}^T\mathbf{x}$ where $\mathbf{x}$ is an unknown row vector such that $\mathbf{x}\mathbf{e}^T = 1$. There exists a vector $\mathbf{x}$ such that the steady state probability of Q_s is equal to that of Q.*

Proof: The theorem is proven by showing that $x = \frac{\pi(Q-Q^-)}{\pi(Q-Q^-)\mathbf{e}^T}$ is a vector that satisfies the required property. The complete proof can be found in [4]. □

The bounds improvement method can be applied after a first set of bounds π^- and π^+ have been computed by means of the Courtois and Semal's method.

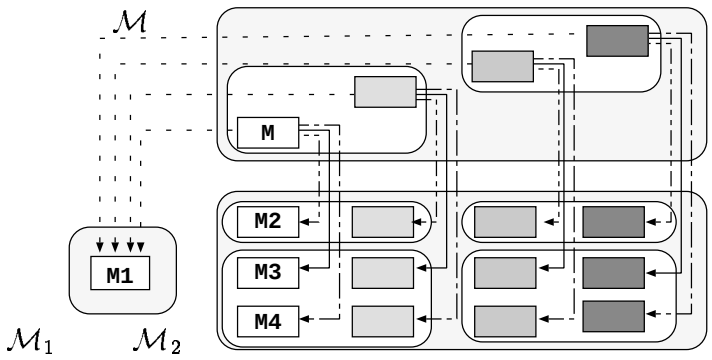


Figure 3: Difference in state aggregation when changing the color class partitioning.

instance without changing the reached symbolic marking, this is the case of objects l_1, l_2 and l_3 in marking M. In order to exploit this property, it is convenient to use a representation for symbolic markings that keeps objects with the same distribution of tokens grouped into sets. We call these sets *dynamic subclasses* and denote them Z^j_{class}. The new representation of $\mathcal{M}$ using dynamic subclasses is: $\mathcal{M} = Line1(\langle Z^1_{Lines}\rangle, 2\langle Z^2_{Lines}\rangle)$ where $|Z^1_{Lines}| = 3$ and $|Z^2_{Lines}| = 1$. The *symbolic firing* instances enabled in $\mathcal{M}$ are $[Serv1, Z^1_{Lines}]$ and $[Serv1, Z^2_{Lines}]$, the former actually represents three ordinary firing instances.

If the servers in the multiserver queue had not the same service rates, for example if two servers were slower (rate λ_1) while the other two were faster (rate $\lambda_2 > \lambda_1$), then we should have partitioned class $Lines$ into two static subclasses, $Lines_i, i = 1, 2$, each containing two servers with service rate λ_i. This partition causes a splitting of some symbolic marking as shown in Figure 3 (equivalence classes are represented by white boxes). Symbolic marking $\mathcal{M}$ is now split into two symbolic markings: this is needed to distinguish the case where the server with two customers in its queue is a slow one from the case in which that server is a fast one. Similarly $\mathcal{M}_2$ is now split into four symbolic markings.

Observe that in this case qualitative analysis can be performed on the more compact SRG, while exact performance results can be obtained only from the larger SRG. In the following sections we'll show a method to compute bounds for the performance indices, working with a MC of size equal to that of the more compact SRG.

3 Bounds for quasi-lumpable Markov chains

In this section the method presented in [4] to compute bounds for *quasi-lumpable* Markov chains is summarized. It is based on the bounds computation method proposed by Courtois and Semal in [2]. The main result from [2] we use, is a theorem stating that it is possible to compute upper and lower bounds for the steady state probability vector of a DTMC P, when only a (componentwise) lower bound $P^- \leq P$ of the transition probability matrix is available.

In [2] two applications of the above theorem are presented: (1) computation of bounds on conditional steady state probability of a subset of states $\mathcal{S}'$ in a DTMC when the transition probability among the states in the subset is known while only partial or no information is available about the transition probability between states in $S - \mathcal{S}'$ and states in $\mathcal{S}'$; (2) computation

preserving static subclass partition. A symmetry function s applicable to place markings and transition color instances is a family of symmetry functions on color classes: $s = \{s_1, s_2, \ldots, s_n\}$. We denote ξ the set of all such functions.

The marking equivalence classes, called *symbolic markings*, are defined as follows:

Definition 1 (Symbolic marking) *Let Eq be the equivalence relation defined by:*

$$M \; Eq \; M' \iff \exists s \in \xi, \; M' = s.M$$

An equivalence class of Eq is called a symbolic marking, denoted with $\mathcal{M}$.

In [1] some propositions can be found, stating that the possible future qualitative and quantitative evolution of the model is the same for all the markings in an equivalence class.

Let us illustrate the above concepts through an example. Consider a closed queueing system composed of two service centers in tandem; let's assume that there are 5 customers in the system. Customers cycle between the two service centers. The first one is a single server machine with an associated exponentially distributed random delay with parameter μ. The second is a four servers machine, and each server S_i has the same exponentially distributed delay with parameter λ. A customer chooses randomly one of the four servers on each visit to the multiserver node; the same probability is associated with each server. In Figure 2 a SWN representation of the system is depicted.

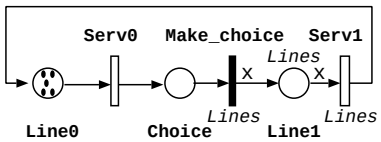


Figure 2: A SWN model of a queueing system

Only one color class is used in this model, namely $Lines = \{l_1, \ldots, l_4\}$, representing the four servers in the multiserver queue. The presence of n tokens with associated color l_i in place $Line1$ means that n customers are queued up to get service from server S_i. A possible marking of this model is $M = Line1(\langle l_1 \rangle, \langle l_2 \rangle, \langle l_3 \rangle, 2\langle l_4 \rangle)$. . There are three markings equivalent to M, that can be obtained by applying all the possible $Lines$ objects permutations to M. Thus M belongs to a symbolic marking $\mathcal{M}$ of cardinality four, that could be represented as follows: $\mathcal{M} = Line1(\langle z_1 \rangle, \langle z_2 \rangle, \langle z_3 \rangle, 2\langle z_4 \rangle)$, where z_i are variables representing objects in $Lines$ and all the ordinary markings belonging to $\mathcal{M}$ can be obtained by assigning actual objects to the variables. Four transition instances are enabled in all the markings belonging to $\mathcal{M}$: $[Serv1, l_i], i = 1, \ldots, 4$, all with the same rate λ. It is possible to relate the arcs exiting from pairs of equivalent markings: in Figure 3 related arcs are represented with similar dashed lines (equivalence classes are represented by dotted boxes). In the same figure it is possible to observe that from the symbolic marking $\mathcal{M}$, two symbolic markings can be reached: $\mathcal{M}_1$ (containing only one ordinary marking) and $\mathcal{M}_2$ (containing twelve ordinary markings). Three out of four firing instances exiting from M end up in the same symbolic marking $\mathcal{M}_2$. Actually it is possible to know in advance which firing instances lead to markings in the same equivalence class since all the objects that have the same *distribution of tokens* in places can be interchanged in a firing

chosen among the set of basic color classes. Transition *end_serv* has two parameters, *srv* and *queue*, of type *Servers* and *Queues* respectively. A *transition instance* is obtained assigning actual objects of proper type to the transition parameters; enabling and firing is defined only for transition instances. We denote $[t, c]$ an instance of t, where c is a tuple belonging to the transition color domain whose elements are the objects assigned to the transition parameters. The set of "colored" tokens that are withdrawn from/added to the input/output places of a given transition when one of its enabled instances fires, are defined through *arc expressions* labelling the net arcs.

The enabling of a transition instance depends both on the multiset of colored tokens contained in its input and inhibitor places and on the evaluation of an optional *predicate* associated with the transition. In the polling system model, predicates are associated with three transitions: $start_srv_1$, $start_srv_2$ and $Walk$. The predicate $[d(queue) = OneSrvQ]$ ($[d(queue) = MltSrvQ]$) indicates that the instances of $start_srv_1$ ($start_srv_2$) that may be enabled are only those with an object from subclass $OneSrvQ$ ($MltSrvQ$) assigned to parameter *queue*. Indeed this transition represents the start of a service for a customer in a queue allowing only one (multiple) service at a time. Finally, the predicate $[nextq = \oplus queue]$ indicates that in order for a $Walk$ instance to be enabled, the parameter *nextq* must be assigned the successor ($\oplus$) of the object assigned to parameter *queue*; this predicate is used to model the cyclic path followed by the servers.

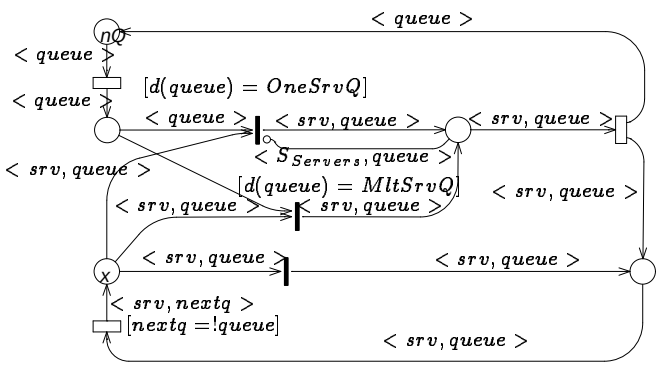


Figure 1: SWN model of a polling system

There are two more functions to be defined to complete the SWN definition: the transition priority and weight functions (denoted π_t and θ_t respectively). Transition instances may have different priority levels, the enabling rule must be modified to take into account priority.

The transition weight function θ_t defines the firing rates of transition instances and is used to solve probabilistically the conflicts that may arise during the net evolution. The transition rates may depend on the particular transition instance, however there are constraints on the kind of dependence allowed, necessary to guarantee that all the objects belonging to the same static subclass behave homogeneously *by construction*.

The Symbolic Reachability Graph A major interest of SWNs is that they provide a modeling framework in which symmetries appear naturally as a way of reducing both the complexity of the representation and the state space explosion problem [1]. Symmetries in SWN are implicitly defined at the color class level, by means of *symmetry functions*. A symmetry function s_i on an ordered/unordered color class C_i is any rotation/permutation of the objects in the class,

2 Stochastic Well-Formed Colored Petri Nets: an informal introduction

In this section the SWN formalism and the Symbolic Reachability Graph generation technique are informally introduced by means of two examples.

Let's consider a polling system comprising a set of waiting rooms (of limited capacity) where customers can queue up, and a set of servers that cyclically visit the waiting rooms to serve the customers. A GSPN model of such system can be built up by properly linking several subnets: one subnet for each waiting room and one subnet to represent the servers behavior. If the system is homogeneous, i.e., if the system components of the same type behave similarly, a more compact representation can be obtained by explicitely modeling only one component of each type, and adding some annotation to specify how many instances of the submodel are present in the whole net, and how they are connected.

The SWN model of the polling system in Figure 1 implements this idea: the net represents the possible states of a generic queue and of a generic server: tokens in place *Thinking* represent customers still out of the waiting room, tokens in place *Waiting* represent customers queued up in the waiting room, tokens in place *busy_servers* represent customers being served, tokens in place *pos_server* represent idle servers looking for some customer to be served, while tokens in place *serv_out* represent a server moving from a queue to the next one. Transitions represent the possible state changes.

The tokens contained into places are no more undistinguishable, they carry some information needed to distinguish either customers associated with different queues or different servers. The type of data associated with tokens in a given place - the *place color domain* - can be structured, i.e., it can comprise several "fields", each with an associated basic data type. The basic data types are finite and non empty sets called *basic color classes*; in the polling system example there are two basic color classes: $Servers = \{w_1, \ldots, w_n\}$ and $Queues = \{q_1, \ldots, q_m\}$. The color domain of place *busy servers* is structured: it comprises a type *Queues* field and a type *Servers* field representing the customer being served and the server performing the service respectively. Hence the tokens contained in this place are pairs $\langle q_i, w_j \rangle$.

The basic color classes may be *ordered* (in this case a successor function must be defined on the class) and may be partitioned into several *static subclasses*, each containing homogeneously behaving objects. In our example, if the servers visit the queues in cyclic order, the class of queues could be ordered according to the visit sequence: the successor function on the *Queues* class could be defined as $\oplus\ q_i = q_{(i+1) \bmod m}$. If some queues must be served by at most one server at a time, while others can be served by many servers concurrently, then the corresponding class could be partitioned into two subclasses: $Queues = OneSrvQ \cup MltSrvQ$. If two types of servers exist that have different service speed, class *Servers* should be partitioned into two subclasses: $Servers = FastSrv \cup SlowSrv$. Observe that the *Queues* class partitioning is used to distinguish objects with different *qualitative* behavior, while the *Servers* class partitioning is used to distinguish objects with same qualitative but different *quantitative* behavior.

A transition in a SWN model actually represents several transitions in the corresponding "unfolded" GSPN model: for example transition *end_serv* represents a generic end of service happening at any queue. In SWNs transitions are parameterized: the parameter types are

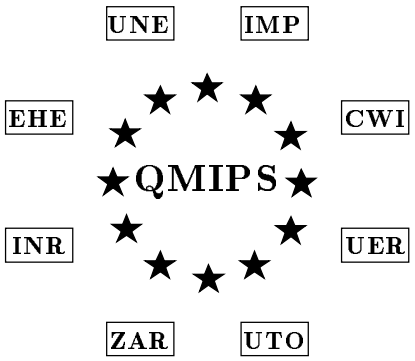
1 Introduction

The Coloured Petri Net (CPN) formalism [7] was initially introduced for model design convenience: indeed more compact and parametric models can be built using CPN instead of the classic PN. Structural symmetries in CPN models, often lead to behavioral symmetries that can be exploited to reduce the computational cost of analysis methods based on the reachability graph construction [6, 5]. When stochastic (exponential) timing is associated to transitions, the modeled system performance can be computed by analyzing the continuous time Markov chain (CTMC) isomorphic to the model reachability graph, hence in case behavioral symmetries apply also at the CTMC level, they can be used to lower the cost of performance analysis as well [3, 9].

The Stochastic Well-Formed Colored Petri Net (SWN) formalism [1] has been introduced to systematize the symmetry exploitation technique: behavioral symmetries of models described with this formalism can be automatically discovered and exploited by defining equivalence classes of markings called *symbolic markings*. A *Symbolic Reachability Graph* (SRG) can be directly generated from a SWN; it retains enough information to study the qualitative properties of the model and to derive a *lumped* CTMC from which all the desired performance indices can be computed. The core of the SRG generation method relies on the definition of object types, the basic color classes, and of their partition into a number of disjoint subsets of homogeneously behaving objects. Equivalence classes of markings are defined as sets of markings that are equal up to a permutation of homogeneously behaving objects. As a consequence of this equivalence relation definition, the potential aggregation decreases as the partitioning of classes into subclasses increases. As we'll see, it may happen that the partitioning of classes into several subclasses is needed only for the correct specification of the *quantitative* behavior of the model (i.e., the transition rates), while they're not used to describe the qualitative behavior (i.e., the token flow in the net). Hence the subclasses used only for the definition of transition rates could be merged when performing a qualitative analysis. By merging them also in the performance analysis phase an error is introduced, and approximated performance results are obtained. In this paper we propose a technique that allows to obtain bounds (rather than just approximations) on the performance indices of SWN models when the states aggregation is performed according to qualitative behavioral symmetries that are not completely reflected at the CTMC level. The bound computation algorithm used, is the one proposed in [4]. The contribution of this paper is a method for defining the *state aggregates* for the application of the above bounding method. Actually the presented result is stronger since the proposed method also allows to directly compute the aggregated MC that is actually solved to compute the bounds, without ever computing the complete quasi-lumpable chain. The method can thus be considered as an extension of the SRG exact aggregation method.

The paper is organized as follows: in Section 2 the SWN formalism and the associated RG aggregation techniques are informally defined; in Section 3 the method for computing bounds of quasi-lumpable MC is summarized; in Section 4 the main contribution of this paper is described: it comprises a structural analysis algorithm to check whether a given SWN is quasi-lumpable, and a modified SRG generation algorithm for the generation of the aggregate MC used for the bounds computation. In Section 5 two application examples are shown. Finally in Section 6 we draw some conclusions and discuss the possible future developments of the method.

can be obtained by avoiding the aggregation of those markings that are qualitatively but not quantitatively equivalent. An alternative approach consists of aggregating all the qualitatively equivalent states, and computing approximated performance indices. In this paper a technique is proposed to compute bounds on the performance of SWN models of this kind, using the results presented in [4]. the technique is based on the Courtois and Semal's bounded aggregation method [2].



Quantitative
Modeling
In
Parallel
Systems

Computing bounds for the performance indices of quasi-lumpable Stochastic Well-Formed Nets

Giuliana Franceschinis
Dip. di Informatica
Università di Torino

Richard R. Muntz
Computer Science Dept.
UC Los Angeles

D W2.T2.12.v1

September 20, 1993

©University of Torino

Distribution Level: All partners

Approved by: Prof. G. Balbo

Abstract Structural symmetries in Stochastic Well-Formed Colored Petri Nets (SWN) lead to behavioral symmetries that can be exploited using the Symbolic Reachability Graph (SRG) construction algorithm: it allows to compute an aggregated Reachability Graph (RG) and a "lumped" Continuous Time Markov Chain (CTMC) that contain all the information needed to study the qualitative properties and the performance of the modeled system respectively. Some models exhibit qualitative behavioral symmetries that are not completely reflected at the CTMC level, we call them *quasi-lumpable* SWN models. In these cases, exact performance indices

The QMIPS Project: The QMIPS project is a collaborative research project supported by the CEC as ESPRIT-BRA project no 7269. It is being carried out by the following organisations: CWI (Amsterdam), EHEI (University of Paris V), Imperial College (London), INRIA (Sophia-Antipolis), University of Erlangen, University of Newcastle, University of Torino and University of Zaragoza.