On the Use of the Power Series Algorithm for General Markov Processes, with an Application to a Petri Net

Ger Koole
Vrije Universiteit, Faculty of Mathematics and Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
tel. (31) 20 4447755, email koole@cs.vu.nl

INFORMS Journal on Computing 9:51-56, 1997

Abstract

The power series algorithm has been developed as a numerical procedure for solving queueing models. This paper shows that it can be used for each Markov process with a single recurrent class. This applies in particular to finite state processes, which is illustrated with the analysis of a bounded stochastic Petri net model.

Analytically obtaining performance measures of multi-dimensional queueing systems is often very difficult. Explicit solutions are only available for some very special models, like Jackson networks. Some specific two-dimensional models can also be solved analytically, for example by showing that solving the problem is equivalent to solving a well-studied complex analysis problem. See Boxma et al. [6] for an overview. The drawbacks of the analytical methods can be summarized as follows: the resulting problems are non-trivial to solve, we are confined to two dimensions, and small changes in the model usually lead to analytically intractable models.

On the other hand, simply numerically solving the steady state equations usually does not work well either. Often the state space is countable, giving need to truncation of the state space at an appropriate level. These truncated state spaces are very big, especially if the dimension grows large and if the accuracy must be high, resulting in very long running times. Thus there is a need for efficient numerical methods to solve large Markov processes. The power series algorithm (psa) aims to be such a method. It was first developed by Hooghiemstra et al. [7] for a model in which several queues share the same servers, and later applied to several other queueing models in a series of papers by Blanc and co-authors (summarized in [5]). For a complete list of references see [10].

The idea behind the power series algorithm is the introduction of an artificial parameter in the transition rates. Then the stationary probabilities become functions of this parameter ρ. For several models it has been shown that the coefficients of the power series expansions around $\rho = 0$ of these probabilities can be computed recursively, as long as ρ is incorporated in the model in a suitable way. Based on these coefficients approximations of the stationary probabilities and other performance measures can be computed.

Section 1 contains the main result of the paper. There it is shown that the psa can (formally) be applied to any Markov process with a single recurrent class. Formally, as there is no guarantee that the obtained power series converge. However, in section 2 we study the ϵ-algorithm which is capable of finding a limit for a divergent series based on its partial sums. It is shown that this works especially well for finite state processes; the psa can even be used to produce exact results. This is illustrated by applying the psa with the ϵ-algorithm to a small bounded Petri net in section 3. For this specific example it is shown that the psa is well suited to produce both approximate and exact results.

Recently, Van den Hout & Blanc took a somewhat different approach in applying the psa to general Markov processes ([8]).

1 General Markov processes

We start the analysis with an arbitrary Markov process with state space X (possibly countable) to which we want to apply the power series algorithm. We denote the transition rate from x to y by q_{xy} (for ease of notation we assume throughout that $q_{xx} = 0$). To use the psa, we construct a family of Markov processes indexed by a parameter ρ, with the same state space X, and with transition rates $\rho^{f(x,y)}q_{xy}$. The problem is how to choose $f(x, y)$ such that the algorithm can be applied to these additional processes. By inserting $\rho = 1$ afterwards, we get results for the original process. Of course, we can choose other values for ρ, if that amounts to a useful model.

For most processes there are several choices of $f : X^2 \to \mathbb{N}$ which make the psa work. In the sequel we will construct such an f based on another function $l : X \to \mathbb{N}$. We call $l(x)$ the *level* of state x.

Definition 1.1 *We call a level function l* computable *if the following three conditions hold:*
(C1) There is a single state $x \in X$ (denoted by 0) with $l(x) = 0$.
(C2) For each k, the states with level k can be ordered such that there are no transitions to higher ordered states within that level.
(C3) $\sum_{y:l(y)\leq l(x)} q_{xy} > 0$ for all $x \in X$, i.e., in each state there are transitions to the same or a lower level.

Often a suitable choice of level function is obvious. In many queueing model for example it can be shown that the total number of customers in the system is a computable level function. And even if there is no obvious choice, then we still can construct one.

Theorem 1.2 *For each Markov process with a single absorbing recurrent class there exists a computable level function.*

Proof. As X is countable, it is possible to construct a list in which each element of X will eventually occur. Now we construct a second list, for which the elements are numbered $0, 1, \ldots$, with as element 0 a recurrent state.

Assume now that the second list has n elements, $0, 1, \ldots, n-1$. Element n is chosen to be the first element in the first list not yet selected, from which there is a transition possible

to one or more states in $\{0, 1, \ldots, n-1\}$. Note that every element of X will eventually occur in the second list, as state 0 can be reached from any state in a finite number of steps. Now let $l(x)$ be the number of state x in this second list. Then l is a computable level function. □

Now take $f(x, y) = (l(y) - l(x))^+$ for all x and y, i.e., all transitions are of the form $\rho^{(l(y)-l(x))^+} q_{xy}$. Thus transitions to lower level states are not changed, but transitions to higher levels get a factor ρ for each level the next state is higher.

Note that every computable level function implies a partial ordering of the states: $x \prec y$ if $l(x) < l(y)$ or if $l(x) = l(y)$ and there is no transition from x to y. Throughout, when we consider a computable level function, we assume that the states are numbered $0, 1, 2, \ldots$ such that if $x \prec y$, then $x < y$. We write p_x for the stationary probability of state x.

Lemma 1.3 *If l is computable, then $p_x = O(\rho^{l(x)})$.*

Proof. We are going to use the idea of the equivalent proof given in [9] for the $BMAP|PH|1$ queue studied there. We use induction, first considering p_0. Because $p_0 = 1$ if $\rho = 0$ it is clear that $p_0 = O(1)$. Define $L_z = \{x | x \leq z\}$. Assume that $p_x = O(\rho^{l(x)})$ for all $x \in L_z$. We complete the induction step by looking at the balance equation between states in L_z and states in $X \backslash L_z$:

$$\sum_{x \in L_z} \sum_{y \notin L_z} \rho^{l(y)-l(x)} q_{xy} p_x = \sum_{x \notin L_z} \sum_{y \in L_z} q_{xy} p_x.$$

Now we show that $z' = z + 1$ is of the required order. Using the induction hypothesis, and the structure of the transitions, it is clear that the left hand side of the equation is of order $O(\rho^{l(z')})$. For z' we have that $\sum_{y \in L_z} q_{z'y} > 0$, and thus (using that all coefficients are non-negative) $p_{z'} = O(\rho^{l(z')})$. □

The last lemma implies that we can write the stationary probabilities p_x as

$$p_x = \sum_{k=0}^{\infty} b_{kx} \rho^{l(x)+k},$$

assuming that these series converge.

Theorem 1.4 *If l is computable, all b_{kx} can be computed recursively.*

Proof. We derive the equations from which we can compute the b_{kx}. The equilibrium equations are:

$$\sum_y \rho^{(l(y)-l(x))^+} q_{xy} p_x = \sum_y \rho^{(l(x)-l(y))^+} q_{yx} p_y.$$

Inserting $p_x = \rho^{l(x)} \sum_k \rho^k b_{kx}$ gives:

$$\sum_y \rho^{(l(y)-l(x))^+} q_{xy} \rho^{l(x)} \sum_k \rho^k b_{kx} = \sum_y \rho^{(l(x)-l(y))^+} q_{yx} \rho^{l(y)} \sum_k \rho^k b_{ky}.$$

Consider for fixed x the terms with $\rho^{l(x)+k}$:

$$\sum_{y: l(y) \leq l(x)} q_{xy} b_{kx} + \sum_{y: l(y) > l(x)} q_{xy} b_{k-l(y)+l(x),x} =$$

$$\sum_{y:l(y)\le l(x)} q_{yx} b_{ky} + \sum_{y:l(y)>l(x)} q_{yx} b_{k-l(y)+l(x),y}. \qquad (1)$$

From this equation we can derive b_{kx}, for $x \neq 0$, assuming we have already calculated b_{ky} for $y < x$ and b_{ly} for $l < k$ and sufficiently many y (depending on the model at hand). This can only be done if the coefficient of b_{kx} is positive (which is guaranteed by (C3)) and if $q_{yx} = 0$ if $y > x$ and $l(y) = l(x)$ (guaranteed by (C2)). This procedure can be repeated until all coefficients which are needed have been calculated.

The b_{k0} can be determined from $\sum_x p_x = 1$: it easily follows that $b_{00} = 1$ (if $\rho = 0$ this is the only recurrent state, by (C1)) and that for $k > 0$ b_{k0} can be computed from

$$\sum_{x:l(x)\le k} b_{k-l(x),x} = 0. \qquad \square$$

Combining the theorems 1.2 and 1.4 gives:

Corollary 1.5 *For each Markov process with a single absorbing recurrent class levels l can be chosen such that the coefficients of the power series of the stationary probabilities can be computed recursively.*

If there is more than one recurrent class each of them should be handled separately. What remains is to compute the probabilities of entering the different recurrent classes, given the initial distribution. Whether this is a numerically difficult task depends on the model at hand.

So far, we have only talked about continuous time Markov processes, and not about discrete time Markov chains. They can be dealt with as well, simply by taking $\rho = 1$ in a model with $\sum_y q_{xy} = 1$ for each x. The only complication is that we cannot assume $q_{xx} = 0$. However, it is readily seen that the term $q_{xx} b_{kx}$ cancels on both sides of (1).

1.1 Implementation

Theorem 1.4 gives us a numerical procedure to compute the coefficients. We give an algorithm in pseudo code for the case that each level consists of a single state, i.e., $l(x) = x$. The algorithm computes all coefficients up to a certain power K, i.e., all coefficients b_{kx} with $k + l(x) = k + x \le K$.

Power Series Algorithm
for $k = 0$ **to** K **do**
  if $k = 0$ **then** $b_{00} \leftarrow 1$
    else $b_{k0} \leftarrow -\sum_{0<x\le k} b_{k-x,x}$
  endif
  for $x = 1$ **to** $K - k$ **do**
    $b_{kx} \leftarrow \Big(\sum_{0\le y<x} q_{yx} b_{ky} + \sum_{x<y\le k+x} q_{yx} b_{k-y+x,y} - \sum_{x<y\le k+x} q_{xy} b_{k-y+x,x} \Big) \Big/ \sum_{0\le y<x} q_{xy}$
  endfor
endfor

Based on these coefficients the partial sums of p_x can be computed, which serve as approximations of the steady state probabilities. For convergence issues we refer to the discussion of the ϵ-algorithm. Often however we are interested in a single performance measure $g : [0,1]^X \to \mathbb{R}$, like the expected queue length in queueing models. If g is a linear combination of the stationary probabilities the coefficients of the power series of $g(p.)$ can easily be computed from the b_{kx}. The ϵ-algorithm can now be applied to this power series.

The complexity of the psa is discussed in section 2.

1.2 Examples

In this subsection we discuss the choice of l for several well known queueing models. Note however that in order to decide whether a certain level function is computable it suffices to know which transitions have a positive rate: the actual value is not important. This gives the possibility to change the models considerably without choosing another l.

We use the following notation: $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, with the 1 in ith position, and, for $x = (x_1, \ldots, x_m)$, $|x| = x_1 + \ldots + x_m$.

Birth-death processes. The m-dimensional birth-death process consists of m queues, where arrival and departure rates depend on the state, and can occur in batches (in different queues simultaneously). However, no arrivals and departures can occur simultaneously, avoiding transitions between queues. Thus the possible transitions out of state $x \in \mathbb{N}^m$ are of the form $x \to x + y$ or $x \to x - y$, with $y \geq 0$ (and $x - y \geq 0$). We assume that for each $x \neq 0$ there is a $y \neq 0$ such that $q_{x,x-y} > 0$. If we take $l(x) = |x|$, it is easily seen that these m-dimensional birth-death processes satisfy definition 1.1. It is also a rich class. The fork-join queue (to which the psa is applied in [10]), the shortest queue model ([1]), the coupled processor model ([2, 7]) and numerous other systems belong to it.

Networks of queues. A tandem of queues is an example of a model which does not fall in the class of problems described above, but where we can take $l(x) = |x|$. Indeed, if customers enter queue 1, and join after service queue 2, ..., up to m, then the possible transitions within each level are all of the form $x \to x - e_i + e_{i+1}$, giving an ordering within level k: $(k, 0, \ldots, 0) \prec \ldots \prec (0, \ldots, 0, k)$.

For models with a more general routing structure, as in Jackson networks, this does not work any more; cycles within a level become possible. A solution is to take as state space $(x_1 + \ldots + x_m, x_2 + \ldots + x_m, \ldots, x_m)$, or, equivalently, to take $l(x) = x_1 + 2x_2 + \ldots + mx_m$. For this choice of l we can allow transitions from one queue to another, i.e., transitions of the form $x \to x - e_i + e_j$ (for x with $x_i > 0$), in addition to the batch arrivals and departures from the m-dimensional birth-death process.

Another approach, which does not fit into our framework, is when we take again $l(x) = |x|$, but a transition of the form $x \to x - e_i + e_j$ with $i > j$ gets a factor ρ. Thus we have given a transition from queue i to queue j a factor ρ, although the states lie within the same level. For the other transitions the factors are taken normally, including the transitions from queue i to j if $i < j$. The psa works again in this case, and the ordering within a level is the same as for the tandem model.

Now we study models where the state of the system is not completely described by the

queue lengths only: we consider polling models, where the position of the server is part of the state description, and models with an additional Markov process representing the environment (generalizing the arrival or service processes).

Markov arrival processes. First consider a single queue with arrivals according to a Markov arrival process (MAP). Assume that the states y of the MAP are numbered, such that the psa can be applied to the Markov process underlying the MAP, with levels $l(y) = y$ (which gives the restriction that there must be a single recurrent class). The states are of the form (x, y), with y the state of the MAP and x the number of customers in the queue. State (x, y) has level $x + y$. The only possible transitions within a level are of the form $(x, y) \to (x + 1, y - 1)$, thus (C2) is easily satisfied. The same holds for (C3), assuring that the psa works for this model. Note that the rates at which arrivals occur do not necessarily have a factor ρ in it (as in the transition above), because the state of the MAP changes also. Only if the state of the MAP remains the same at arrival instants (the special case of a Markov modulated Poisson process), then each arrival has factor ρ. The term MAP is somewhat misleading, as it suggests that the transition rates within the Markov process governing the arrivals must be independent of x. As this is not true, it is perhaps better to speak of an auxiliary Markov process.

General service times. Such an auxiliary Markov process (AMP) can also be used to model (potential) departures from a queue with Poisson arrivals. For example, the state of the AMP can represent the current service phase. Note that the service times need not be independent. For example, we can easily model the consecutive service times to be dependent, or let the service times depend on the queue length. When modeling departures, it is natural to freeze the AMP (i.e., keep it in the same state until a customer arrives) when the queue is empty, instead of letting it make transitions without having customers in the queue to serve. But, as transitions to lower level states must be possible from each state except 0, it can only be frozen in state 0, which is therefore of the form $(0, 0)$. Thus the AMP can only be frozen if the transition in the AMP generating the departure is of the form $y \to 0$. If we want to be able to freeze the AMP in different states, a less obvious choice of levels has to be made.

Polling models. An interesting generalization of an auxiliary Markov process governing departures (and possibly also arrivals) is to multiple queues. As Blanc [4] shows, an important class of models which can then be modeled are the polling models. In its simplest form, the state of the AMP denotes the position of the server (i.e., at or between which queues the server is), but generalizations in different directions are possible, like the AMP denoting the service phase, or even the number of customers already served at the current queue, to be able to model for example the limited service discipline. When the server in a polling system finds an empty queue, the server usually moves to the next server; therefore the problem with freezing the departure process occurs only in the single queue case.

2 The ϵ-algorithm and finite state processes

In general the power series expansions of steady state probabilities will not converge at $\rho = 1$. This is not surprising: the psa develops each stationary probability as a power series around $\rho = 0$, and the radius of convergence of such a series is in general unknown. This section is devoted to the study of the convergence properties. To improve the convergence properties we make use of an algorithm applicable to arbitrary power series, the *ϵ-algorithm*, which was first used by Blanc ([3]) in conjunction with the psa.

After introducing the ϵ-algorithm, we restrict to finite state processes. This allows us to write the stationary probabilities as quotients of polynomials in ρ. From this we conclude that the ϵ-algorithm, if applicable, produces exact results. This is illustrated in the next section with the analysis of a bounded Petri net.

2.1 The ϵ-algorithm

The ϵ-algorithm was introduced by Wynn (see e.g. [12]) to accelerate the convergence of power series. Given the partial sums $S_m = \sum_{k=0}^{m} c_k \rho^k$, a two-dimensional array with entries $\epsilon_r^{(m)}$ is computed, using the formula

$$\epsilon_{r+1}^{(m)} = \epsilon_{r-1}^{(m+1)} + (\epsilon_r^{(m+1)} - \epsilon_r^{(m)})^{-1},$$

with initial conditions

$$\epsilon_{-1}^{(m)} = 0, \ m = 1, 2, \ldots,$$

and

$$\epsilon_0^{(m)} = S_m, \ m = 0, 1, \ldots.$$

Now $\epsilon_r^{(m)}$ with r even is used instead of S_m to approximate the limit S_∞. The numbers $\epsilon_r^{(m)}$ with r odd are only used as intermediate results.

The idea behind the ϵ-algorithm is that $\epsilon_{2r}^{(m)}$ approximates S_∞ by a quotient of polynomials, the numerator of degree $m + r$, the denominator of degree r, which are completely determined by the first $2r + m$ coefficients of the power series to be approximated. In the cases considered in this paper, the zeros of the denominator apparently converge to the singularities of S_∞, thereby extending the region of convergence.

Although the ϵ-algorithm involves repeated subtraction and division, Wynn [12] states that it is often remarkably stable. This is in compliance with our findings.

2.2 Finite state processes

Consider a Markov process with $N < \infty$ states. Let G be its infinitesimal generator, i.e., $g_{ij} = q_{ij}$ if $i \neq j$, and $g_{ii} = -\sum_j q_{ij}$. Construct G' from G by replacing the last column by $e = (1, \ldots, 1)$. Then the steady state vector is the unique solution of the equation $pG' = e_N$, if we assume that the process consists of a single recurrent class. Note that all elements of G' are polynomials of ρ.

To compute the stationary probabilities p_x we can apply Cramer's rule, that is,

$$p_x = \frac{|G'_x|}{|G'|},$$

where G'_x is obtained from G' by replacing the xth row by e_N, and where we denote by $|\cdot|$ the determinant of a matrix. As all entries of both matrices are polynomials in ρ, we conclude that p_x is a quotient of polynomials in ρ, i.e., it is a rational function.

Again, assume that the maximum number of levels a transition can go up is $\overline{k}$. Then all entries are of order $\leq \overline{k}$, and as the last column consists of 1's, both determinants are of order $\leq (N-1)\overline{k}$. As it is useless to have more levels than states, and thus $\overline{k} \leq N-1$, we can assume in general that each determinant is of order $\leq (N-1)^2$.

From [12] we know that $\epsilon_{2r}^{(0)}$ approximates S_∞ with a uniquely determined rational function where both the numerator and the denominator are of order r. Thus to compute the stationary probabilities exactly it is sufficient to compute $\epsilon_{2(N-1)\overline{k}}^{(0)}$. If $\overline{k}$ is small (in most examples we had $\overline{k}=1$), this can often be done, even for reasonably sized models.

Another interesting implication of p_x being a rational function is that p_x is analytic in $\rho = 0$. Indeed, p_x has a finite number of poles, each of which is unequal to 0, because $p_x = 1$ (0) for $x = 0$ ($x > 0$), for $\rho = 0$. Thus, for ρ small enough, the power series converge, without applying the ϵ-algorithm.

2.3 Complexity

The psa is above all a method to derive approximations for performance measures of queueing systems. However, as it can also be used to derive exact results for finite processes, it is of interest to study its complexity. We do this for the case that each level consists of a single state. As is derived above, in the worst situation we have to compute the b_{kx} for all k and x such that $k + x \leq 2(N-1)^2$. Together with the fact that there are N states, and $N-1$ possible transitions, this results in a complexity of $O(N^4)$. Thus compared to standard methods which are based on the inversion of a matrix (with complexity $O(N^3)$) the psa behaves poorly.

However for special cases the situation can be better; in the next section a Petri net is studied for which the psa has a complexity of $O(N^2)$. The reason for this is that the psa utilizes the sparseness of the transition matrix.

3 A Petri Net Example

To illustrate the ideas of the previous section, we analyze the simple stochastic Petri net depicted in figure 1. We denote its markings with $(x_1, \ldots, x_5)$, where x_i is the number of tokens at place P_i. As initial markings we take $(n, 0, 0, 0, 0)$, for various n. This marked graph is live and bounded, and to represent its reachability set we can restrict ourselves to (x_1, x_2, x_3), as $x_4 = n - x_1 - x_2$ and $x_5 = n - x_1 - x_3$. (For an introduction to Petri nets, see [11].) Transition t_i has an exponential firing time with rate λ_i.

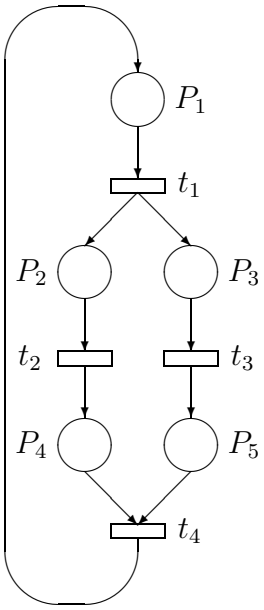

Figure 1. A stochastic Petri net

Note that this Petri net is strongly related to the fork-join queue. Indeed, transition t_1 corresponds to the fork primitive. Transitions t_2 and t_3 correspond to the distributed processing of the tasks, and transition t_4 is only enabled if there is both a token at P_4 and at P_5, that is, if a job has finished service in the fork-join queue. Thus, in queueing terms, the Petri net consists of a closed cycle of three centers, one of which is a fork-join queue, and two of them are simple single server queues.

To apply the psa, we have to partition the state space into levels. We took as levels $l(x_1, x_2, x_3) = n - x_1$. Consequently, transition t_1 gets a term ρ, thus λ_1 is replaced by $\rho\lambda_1$. Equation (1) becomes:

$$b_{kx}\{\lambda_2(x_2 > 0) + \lambda_3(x_3 > 0) + \lambda_4(x_1 + x_2 < n, x_1 + x_3 < n)\}+$$

$$b_{k-1,x}\lambda_1(x_1 > 0, k > 0) =$$

$$b_{k,(x_1+1,x_2-1,x_3-1)}\lambda_1(x_2 > 0, x_3 > 0)+$$

$$b_{k,(x_1,x_2+1,x_3)}\lambda_2(x_1 + x_2 < n) + b_{k,(x_1,x_2,x_3+1)}\lambda_3(x_1 + x_3 < n)+$$

$$b_{k-1,(x_1-1,x_2,x_3)}\lambda_4(x_1 > 0, k > 0).$$

We are interested in the throughput of the system, i.e., the average number of firings of t_1 per unit of time. This is equivalent to computing the stationary probability of having 0 tokens in P_1 (denoted by p), as the throughput is equal to $(1 - p)\lambda_1$.

First we have computed the coefficients of the power series of p. As $p = \sum_{x_2+x_3\leq n} p_{(0,x_2,x_3)}$, this is the sum of the stationary probabilities of all level n states. Thus, the first n coefficients of this power series are 0. There are no transitions 2 or more levels up, and therefore only 2 arrays the size of the state space (which is $N = 1^2+2^2+\ldots+(n+1)^2 = (n+1)(n+2)(2n+3)/6$) and 1 array with the coefficients of p need to be kept in memory (see [10]). After computing

all coefficients of p up to a certain K, we have applied the ϵ-algorithm, after omitting the trailing zeros. To apply this algorithm, 3 arrays of size K have to be stored.

Let us consider the complexity of the psa for this specific problem. The time to compute each b_{kx} is $O(1)$ as the number of transitions is bounded in each state. To get exact results only $O(N)$ coefficients need to be computed for each state, as $\overline{k} = 1$. As there are N states, this results in an overall complexity of $O(N^2)$. The complexity of the ϵ-algorithm is also $O(N^2)$, as we start with $O(N)$ partial sums. Thus the psa allows us to make use of the special structure of the problem to reduce its complexity. Also Gaussian elimination can be seen to have complexity $O(N^2)$ by using the special structure of the problem. The memory requirements are much bigger however, as the whole transition matrix needs to be stored in memory.

Typical output for $\lambda_i = 1$ and $n = 3$ (30 states) can be found in table 1, where $\epsilon_r^{(m)}$ can be found for the series without trailing zeros. For reasons of space we left out the $\epsilon_r^{(m)}$ with $m > 7$.

	$m=0$	1	2	3	4	5	6	7
$r=0$	6.125000	-8.593750	1.455729	0.643993	51.176851	-128.409830	99.173984	-14.747156
1	-0.067941	0.099508	-1.231927	0.019789	-0.005568	0.004394	-0.008778	0.002312
2	-2.621754	0.704660	1.442896	11.740703	-28.031678	23.255298	75.427968	63.409885
3	0.400132	0.122653	0.116897	-0.030711	0.023892	0.010389	-0.080896	-0.003497
4	-2.899217	-172.300284	4.966029	-9.717844	-50.802362	64.473330	76.329890	-84.021912
5	0.116750	0.122538	-0.098813	-0.000448	0.019064	0.003445	-0.009733	0.002105
6	0.448322	0.448332	0.448334	0.448328	0.448333	0.448330	0.448332	
7	1.03×10^5	6.14×10^5	-1.73×10^5	1.78×10^5	-2.56×10^5	4.30×10^5		
8	0.448334	0.448332	0.448331	0.448331	0.448331			
9	-1.14×10^4	-7.77×10^5	2.38×10^6	-6.91×10^6				
10	0.448331	0.448331	0.448331					
11	-2.95×10^7	3.65×10^7						
12	0.448331							

Table 1. Approximations $\epsilon_r^{(m)}$ of p for the Petri net example with $\lambda_i = 1$ and $n = 3$

The table shows some interesting phenomena. First note that the series itself clearly diverges. (Remind that $\epsilon_0^{(m)} = S_m$, so that the first row contains the partial sums.) As r gets large, for r even, the approximation gets better. Note that as $\epsilon_r^{(m)}$ gets close to $\epsilon_r^{(m+1)}$ for r even, then $\epsilon_{r+1}^{(m)}$ (which is only used as intermediate step, as $r+1$ is odd) gets very large. This does not lead to numerical instabilities (at least not in this case), as can be seen from the table. Even if we take r very large, we find the correct answer (e.g., $\epsilon_{100}^{(0)} = 0.448331$) although, theoretically speaking, this leads to repeated division by zero.

Note that, as $\overline{k} = 1$ and the number of states is 30, $\epsilon_{58}^{(0)}$ should give the correct answer (and it does: 0.448331), but in the present case it suffices to compute $\epsilon_{10}^{(0)}$ to obtain an approximation correct up to 6 digits.

In the following table results are given for various values of the number of initial tokens n, and again $\lambda_i = 1$. The first column gives n, the second the total number of states N, the third

the computed value of p, and the fourth the lowest value of r for which $\epsilon_r^{(0)}$ approximates p with a precision of 5 digits. Note that this value of r is considerably lower than $2(N-1)$, the value for which $\epsilon_r^{(0)} = S_\infty$. This shows again the value of the psa as a means to approximate performance measures. A good indication that the approximations are close are the values of $\epsilon_r^{(m)}$ for r odd; if they are big, $\epsilon_{r+1}^{(0)}$ is close. To compute $\epsilon_r^{(0)}$ for $n = 100$ and for r up to 500 took ≈ 15 minutes on a fast workstation.

n	N	p	r
1	5	.714286	2
2	14	.551546	6
3	30	.448331	8
5	91	.325768	16
10	506	.193286	30
25	6201	.087018	82
50	45526	.045405	198
100	348551	.023207	382

Table 2. Approximations of p for the Petri net example with $\lambda_i = 1$

For $n = 1$ the computation of the b_{kx} can easily be done by hand, and we find (for general firing rates) that $p = \alpha - \alpha^2\rho + \alpha^3\rho^2 - \ldots$, with $\alpha = \lambda_1(\lambda_2 + \lambda_3 + 3\lambda_4)/((\lambda_2 + \lambda_3)\lambda_4)$. If we apply the ϵ-algorithm once, i.e., if we compute $\epsilon_2^{(m)}$, we find that $\epsilon_2^{(m)} = \alpha(1 + \alpha\rho)^{-1}$ for all m. Thus indeed, if we take $\lambda_i = \rho = 1$, we get $p = \frac{5}{7} \approx 0.714286$, coinciding with our numerical results.

Acknowledgements. I like to thank Prof. J.W. Cohen for many interesting discussions on this subject, and an anonymous referee for his thorough reviews and useful comments.

This research was carried out at CWI, Amsterdam, and supported by the European Grant BRA-QMIPS of CEC DG XIII.

References

[1] J.P.C. Blanc, 1987. A note on waiting times in systems with queues in parallel, *Journal of Applied Probability 24*, 540–546.

[2] J.P.C. Blanc, 1987. On a numerical method for calculating state probabilities for queueing systems with more than one waiting line, *Journal of Computational and Applied Mathematics 20*, 119–125.

[3] J.P.C. Blanc, 1990. A numerical approach to cyclic-service queueing models, *Queueing Systems 6*, 173–188.

[4] J.P.C. Blanc, 1992. Performance evaluation of polling systems by means of the power-series algorithm, *Annals of Operations Research 35*, 155–186.

[5] J.P.C. Blanc, 1993. Performance analysis and optimization with the power-series algorithm, in L. Donatiello and R. Nelson (eds.), *Performance Evaluation of Computer and Communication Systems*, Lecture Notes in Computer Science 729, Springer-Verlag, pp. 53–80.

[6] O.J. Boxma, G.M. Koole and Z. Liu, 1994. Queueing-theoretic solution methods for models of parallel and distributed systems, In O.J. Boxma and G.M. Koole (eds.), *Performance Evaluation of Parallel and Distributed Systems — Solution Methods*, CWI Tract 105, pp. 1–24.

[7] G. Hooghiemstra, M. Keane and S. van de Ree, 1988. Power series for stationary distributions of coupled processor models, *SIAM Journal on Applied Mathematics 48*, 1159–1166.

[8] W.B. van den Hout and J.P.C. Blanc, 1994. The power-series algorithm for a wide class of Markov processes, CentER Discussion Paper 9487, Tilburg University.

[9] W.B. van den Hout and J.P.C. Blanc, 1995. Development and justification of the power-series algorithm for BMAP-systems, *Stochastic Models 11*, 471–496.

[10] G.M. Koole, 1994. On the power series algorithm, In O.J. Boxma and G.M. Koole (eds.), *Performance Evaluation of Parallel and Distributed Systems — Solution Methods*, CWI Tract 105, pp. 139–155.

[11] T. Murata, 1989. Petri nets: Properties, analysis and applications, *Proceedings of the IEEE 77*, 541–580.

[12] P. Wynn, 1966. On the convergence and stability of the epsilon algorithm, *SIAM Journal on Numerical Analysis 3*, 91–122.