A Passage-time Preserving Equivalence for Semi-Markov Processes

J.T.Bradley

Department of Computing,
Imperial College of Science, Technology and Medicine,
Huxley Building, 180 Queen's Gate, London SW7 2BZ.
email: jb@doc.ic.ac.uk

Abstract. An equivalence for semi-Markov processes is presented which preserves passage-time distributions between pairs of states in a given set. The equivalence is based upon a state-based aggregation procedure which is $O(n^2)$ per state in the worst case.

1 Introduction

An equivalence compares two systems and defines whether they are in some way *equal in behaviour* for a given property. The concept of an equivalence for a stochastic model which preserves stochastic behaviour is not new (i.e. Markovian lumping [1] and within Markovian process algebras [2–5]). As far as we are aware, the existence of similar useful equivalences for more distribution-rich stochastic models, such as semi-Markov processes is new.

In this paper, the property we are seeking to preserve is that of having identical passage-time distributions between pairs of states. This, in stochastic terms, is a very strong constraint compared to say having similar steady-state behaviour or reward behaviour.

The equivalence is based on an aggregation technique for SMPs which is fully defined here and this provides us with opportunities for model reduction. Controlling SMP state space size so that transient [6], steady-state and passage-time techniques [7] can be undertaken tractably is an important goal. We make heavy use of Laplace transforms in manipulating the semi-Markov state space; this is justified by the fairly recent developments in Laplace inversion algorithms [8] that can provide numerically stable inversions for even discontinuous functions.

The paper is structured as follows: the equivalence is defined in the next section and related to the aggregation function. An example aggregation is given for a 4-complete transition graph. The individual aggregation steps are discussed and justified stochastically. Finally, the whole algorithm is formally presented and the complexity is discussed.

2 Definition of Equivalence

2.1 Introduction

We specify a semi-Markov process, $M \in SMP$, as made up of the tuple (S, P, L) where S is the finite set of states, P is the underlying DTMC and L is the distribution matrix. M_X is used to access the X element of the tuple.

We define the equivalence operator, $(\overset{m,n}{\sim})$, for two semi-Markov processes, M, N:

$$M \overset{m,n}{\sim} N \tag{1}$$

where $m \subseteq M_S$ and $n \subseteq N_S$. This says that two systems M and N are passage-time equivalent over the state subsets m and n. This means that all passage-time distributions between states in m over M are identical to a specific reordering of passage-time distributions between states in n over N.

2.2 Formal Definition

In formal terms, $M \overset{m,n}{\sim} N$ iff M^* and N^* are isomorphic upto state relabelling.

M^* and N^* are defined by using the aggregation function, *aggregate_smp* and the subsidiary function *fold* (defined in section 3.3).

$$\begin{aligned} M^* &= \mathit{fold}(\mathit{aggregate\_smp}, M, M_S \setminus m) \\ N^* &= \mathit{fold}(\mathit{aggregate\_smp}, N, N_S \setminus n) \end{aligned} \tag{2}$$

where *aggregate_smp* satisfies the following property. Given a first passage-time distribution function between two states:

$$\mathit{passage\_time} : SMP \times (S \times S) \to ([0,1] \to \mathbb{R}^+) \tag{3}$$

For $M_2 = \mathit{aggregate\_smp}(M_1, i)$ such that $M_{1_S} = \{i\} \cup M_{2_S}$:

$$\forall r, s \in M_{2_S}, \mathit{passage\_time}(M_2, (r, s)) = \mathit{passage\_time}(M_1, (r, s)) \tag{4}$$

3 SMP State Lumping

3.1 An Example of SMP State Lumping

We illustrate the state-aggregation of SMPs with an example of a 4-complete graph, representing the transitions of an SMP. This situation is shown in figure 1.

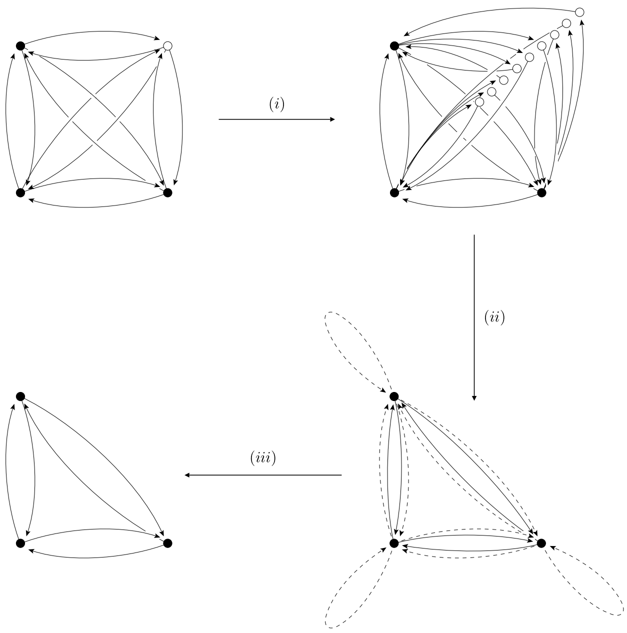

Fig. 1. Reducing a complete 4 state graph to a complete 3 state graph.

In the first instance, a state is selected for aggregation. For a state with m in-transitions and n out-transitions, there are mn two-transition paths that pass through the state. In the worst case (for a k-complete graph) there are $(k-1)^2$ such transitions.

After stage (i) of the process in figure 1, these transitions are represented explicitly by replicating the chosen node (shown in white) and transitions.

In step (ii), the 9 two-transition paths are aggregated to give 9 single transitions consist of cycles and non-cycles (shown as dashed edges). The non-cyclic transitions shadow transitions from the original SMP graph.

In the final step, the non-cyclic transitions are combined with their shadow transitions and the cyclic transitions are absorbed into the non-cycle distributions. This leaves us with a 3-complete graph.

The process for a single aggregation is worst case $O(k^2)$, and for aggregating the state space of most of an SMP, $O(k^3)$.

In the next sections, we will show how the aggregation process preserves the stochastic effect of the aggregated state in such a way that passage-time distributions between unaggregated states are kept in tact.

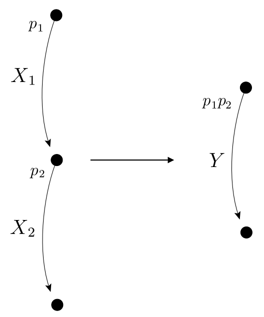

Fig. 2. Aggregating sequential transitions in an SMP.

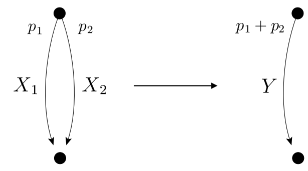

Fig. 3. Aggregating branching transitions in an SMP.

3.2 Basic Reduction Steps

Sequential Reduction From figure 2: $Y = X_1 + X_2$ is a convolution and therefore in Laplace form, $L_Y(z) = L_{X_1}(z)L_{X_2}(z)$. If there are probabilistic weightings p_1 and p_2 on respectively of the transitions, then the overall weighting for the path is p_1p_2. This is used in the *agg_seq* function, equation (13).

Branch reduction To aggregate the type of branch structure shown in figure 3, we can sum the respective probabilities and distributions. Thus the aggregate probability for the branch is $p_1 + p_2$ and the overall distribution Y for the sojourn is given by $\frac{p_1}{p_1+p_2}L_{X_1}(z) + \frac{p_2}{p_1+p_2}L_{X_2}(z)$. This is used in the *agg_branch* function, equation (14).

Cycle Reduction In a semi-Markov process, cycle reduction may be necessary (and indeed is in the next section) if we create a state with at least one out-transition and a transition to itself (or cycle). We can remove the cycle by making its stochastic effect part of the out-going transitions.

Therefore we need to modify those out-transitions, and this section describes that modification. We picture our state system as being in the first stage of figure 4: with $(n-1)$ out-transitions with probability p_i of departure along edge i; each out-transition carries a sojourn of X_i; the cycle probability is p_n and carries a delay of X_n.

The first step, (i), is to isolate the cycle and treat is separately from the branching out-transitions. We do this by introducing an instantaneous delay and extra state after the cycle, $Z \sim \delta(0)$; the introduction of an extra state is only to aid our visualisation of the problem and is not necessary or indeed performed in the actual aggregation algorithm. Clearly the probability of doing this instantaneous transition is $(1 - p_n)$ (this is just the probability of performing any of the out-transitions from our original state). We now have to renormalise the p_i probabilities on the branching state to become $q_i = p_i/(1 - p_n)$.

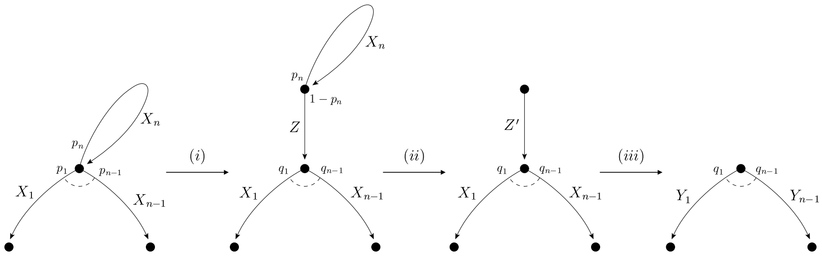

Fig. 4. The three-step removal of a cycle from an SMP.

In step (ii) of figure 4, we aggregate the delay of the cycle into the instantaneous transition creating a new transition with distribution Z'. This can be done in one of two ways: either by treating the system as a random sum of the random variable, X_n, or by considering a linear system of Laplace transforms. In the latter case:

$$L_{Z'}(z) = p_n L_{X_n}(z) L_{Z'}(z) + (1 - p_n) \underbrace{L_Z(z)}_{1} \tag{5}$$

This gives us:

$$L_{Z'}(z) = \frac{1 - p_n}{1 - p_n L_{X_n}(z)} \tag{6}$$

In stage (iii) of the process, the Z' delay is simply sequentially convolved onto the X_i events to give us our final system.

In summary then: we have reduced an n-out-transition state where one of the transitions was a cycle to an $(n-1)$-out-transition state with no cycle such that:

$$q_i = \frac{p_i}{1 - p_n} \tag{7}$$

and:

$$L_{Y_i}(z) = \frac{1 - p_n}{1 - p_n L_{X_n}(z)} L_{X_i}(z) \tag{8}$$

This form is used explicitly in equation (15), the *agg_cycle* function of the next section.

3.3 State Lumping Definition

Notation In the description below, the following notation is used:

A semi-Markov process, $M \in SMP$, is made up of the tuple (S, P, L) where S is the finite set of states, P is the underlying DTMC and L is the distribution matrix in Laplace form.

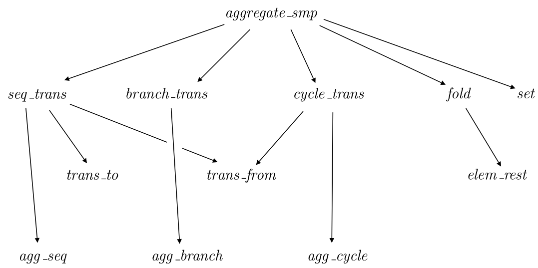

Fig. 5. The functional dependency diagram for the algebraic description of *aggregate_smp*.

Also: $Prob = [0, 1]$, $LaplaceT = \mathbb{C} \to \mathbb{C}$, $PD = (Prob, LaplaceT)$ and finally $T = (S \times S) \times PD$.

So more specifically, $P : S \times S \to [0, 1]$ such that $\sum_{j \in S} P(i, j) = 1$ and $L : S \times S \to LaplaceT$.

As before, for $M \in SMP$, $M_S = S$, $M_P = P$ and $M_L = L$ so that we can define $i \stackrel{M}{\to} j$ to mean that, there exists a transition from state i to state j in M, or formally $M_P(i, j) > 0$.

Finally, figure 5 shows the functional dependencies in the definition of *aggregate_smp*.

Main aggregation algorithm In this section, we define the overarching aggregation function, *aggregate_smp*, which is defined on an SMP and a state. The lumping procedure can be applied to any state of an irreducible semi-Markov process or to any except absorbing or initial states of a transient SMP.

The general algorithm for aggregating a single state into an SMP such that the passage-times are preserved follows the pattern seen in the example aggregation of section 3.1.

Reading from the bottom up in equation (9), the general method works as follows, for a state i to be aggregated in an SMP, M:

1. Find all valid paths of length two that have i as the middle state and combine into a set of single transitions using the techniques of section 3.2. Call this set ts.
2. Separate ts transitions into a set of cycles, ts_c, and a set of branching transitions to other states, ts_b.
3. Apply the branching reduction of section 3.2 to the set of branching transitions, ts_b. Where a new transition from ts_b that does not already exist in M, then the transition from ts_b is to be placed unchanged into the new SMP. Call the new set of transitions, us.

4. Overwrite all the transition mentioned in us in M and call the new SMP, M'.
5. Aggregate all the cyclic transitions in ts_c, using the method shown in section 3.2. Place rewritten transitions of M' in vs.
6. Finally, integrate all the vs transitions into M' and return the final SMP.
7. The result will have a disconnected state, i, which can be removed.

$$
\begin{aligned}
&\mathit{aggregate\_smp} : SMP \times S \rightarrow SMP \\
&\mathit{aggregate\_smp} \quad (M, i) = \mathit{fold}(\mathit{set}, M', vs) \;.\; vs = \mathit{cycle\_trans}(M', ts_c), \\
&\qquad\qquad\qquad\qquad M' = \mathit{fold}(\mathit{set}, M, us), \\
&\qquad\qquad\qquad\qquad us = \mathit{branch\_trans}(M, ts_b), \\
&\qquad\qquad\qquad\qquad ts_b = ts \setminus ts_c, \\
&\qquad\qquad\qquad\qquad ts_c = \{((i,j),t) \;.\; ((i,j),t) \in ts, i = j\}, \\
&\qquad\qquad\qquad\qquad ts = \mathit{seq\_trans}(M, i)
\end{aligned}
\tag{9}
$$

Higher-level aggregation functions From figure 5, we define the key functions used by *aggregate_smp*: *seq_trans*, *branch_trans* and *cycle_trans*.

seq_trans takes a state, i, to be aggregated and the SMP and returns a set of transitions which represent all the paths (from one state preceding to one state after i) which pass through i. The paths have been aggregated into single transitions using *agg_seq*. The number of transitions generated by *seq_trans* is equal to the product the number transitions leading into i and the number leaving the state.

$$
\begin{aligned}
&\mathit{seq\_trans} : SMP \times S \rightarrow \mathcal{P}(T) \\
&\mathit{seq\_trans} \quad (M, i) = \{((i,j), \mathit{agg\_seq}(t, t')) \\
&\qquad\qquad . \; (i, t) \in \mathit{trans\_to}(i, M), (j, t') \in \mathit{trans\_from}(i, M)\}
\end{aligned}
\tag{10}
$$

branch_trans processes an SMP, M, and a set of transitions, R, which must not contain any cycles. The transitions are to be integrated into the SMP, M. If there is a pre-existing transition in M for a given member of R, then the two are combined using *agg_branch*, otherwise the transition in R is just returned unchanged.

$$
\begin{aligned}
&\mathit{branch\_trans} : SMP \times \mathcal{P}(T) \rightarrow \mathcal{P}(T) \\
&\mathit{branch\_trans} \quad (M, R) = \{((i,j), \mathit{agg\_branch}(t, t')) \;.\; ((i,j), t)) \in R, \\
&\qquad\qquad t' = \; \mathsf{if}\; i \xrightarrow{M} j \;\mathsf{then}\; (M_P(i,j), M_L(i,j)) \;\mathsf{else}\; (0,0)\}
\end{aligned}
\tag{11}
$$

cycle_trans processes an SMP, M, and a set of transitions, R, which must only contain cyclic transitions. For each cycle defined in R from i to i, the set of leaving transitions is selected from the SMP, M. For each member of that set, the aggregation function *agg_cycle* is applied. All the modified transitions are unified into a single set and returned.

$$
\begin{aligned}
&cycle\_trans : SMP \times \mathcal{P}(T) \to \mathcal{P}(T) \\
&cycle\_trans \quad (M, R) = \bigcup_{i:((i,i),t')\in R} X \\
&\qquad\qquad . \ X = \{((i,j), agg\_cycle(t,t')) \ . \ (j,t) \in trans\_from(i,M)\}
\end{aligned} \tag{12}
$$

Basic aggregation functions This section describes the three basic structure aggregation operations, displayed in section 3.2. These are:

Sequential Aggregation Used to represent two transitions in sequence as a single transition.

$$
\begin{aligned}
&agg\_seq : PD \times PD \to PD \\
&agg\_seq \quad ((p, d(z)), (p', d'(z))) = (pp', d(z)d'(z))
\end{aligned} \tag{13}
$$

Branching Aggregation Used to represent two branching transitions which terminate in the same state as a single transition.

$$
\begin{aligned}
&agg\_branch : PD \times PD \to PD \\
&agg\_branch \quad ((p, d(z)), (p', d'(z))) = \left(p + p', \frac{p}{p+p'}d(z) + \frac{p'}{p+p'}d'(z)\right)
\end{aligned} \tag{14}
$$

Cycle Aggregation Used to represent a cycle to the same state and a leaving transition as a single transition. The first argument is the leaving transition and the second is the cyclic transition. As described in section 3.2, if there is more than one out-transition then the transformation will need to be applied to each in turn.

$$
\begin{aligned}
&agg\_cycle : PD \times PD \to PD \\
&agg\_cycle \quad ((p, d(z)), (p_c, d_c(z))) = \left(\frac{p}{1-p_c}, \frac{(1-p_c)d(z)}{1-p_c d_c(z)}\right)
\end{aligned} \tag{15}
$$

Subsidiary Functions The function *trans_from* takes a state, i, and returns the set of states, probabilities and distributions which succeed i.

$$\begin{aligned} &\mathit{trans\_from} : S \times SMP \rightarrow \mathcal{P}(S \times PD) \\ &\mathit{trans\_from} \quad (i, M) = \{(j, (M_P(i,j), M_L(i,j))) \;.\; j \in M_S, i \overset{M}{\rightarrow} j\} \end{aligned} \tag{16}$$

The function *trans_to* takes a state, i, and returns the set of states, probabilities and distributions which connect to i.

$$\begin{aligned} &\mathit{trans\_to} : S \times SMP \rightarrow \mathcal{P}(S \times PD) \\ &\mathit{trans\_to} \quad (i, M) = \{(j, (M_P(j,i), M_L(j,i))) \;.\; j \in M_S, j \overset{M}{\rightarrow} i\} \end{aligned} \tag{17}$$

The *set* function is used to set a given transition in the DTMC and distribution matrix of an SMP. If necessary the current transition is overwritten.

$$\begin{aligned} &\mathit{set} : T \times SMP \rightarrow SMP \\ &\mathit{set} \quad (((i,j),(p,d(z))), M) = (M_S, P', L') \\ &\qquad . \; P'(k,l) = \begin{cases} p & \text{if } (k,l) = (i,j) \\ M_P(k,l) & \text{otherwise} \end{cases} \\ &\qquad\;\; L'(k,l) = \begin{cases} d(z) & \text{if } (k,l) = (i,j) \\ M_L(k,l) & \text{otherwise} \end{cases} \end{aligned} \tag{18}$$

fold is used in this context to take one transition at a time, from a set of probability-distribution transitions, and apply each of the new transitions to a given SMP. This occurs across the entire supplied transition set until there are none left to apply.

$$\begin{aligned} &\mathit{fold} : (A \times B \rightarrow B) \times B \times \mathcal{P}(A) \rightarrow B \\ &\mathit{fold} \quad (f, r, \Gamma) = \begin{cases} r & \text{if } \Gamma = \emptyset \\ f(x, r, \mathit{fold}(f, r, \Gamma')) & \\ \quad . \; (x, \Gamma') = \mathit{elem\_rest}(\Gamma) & \text{otherwise} \end{cases} \end{aligned} \tag{19}$$

The function, *elem_rest*, is used by *fold* to select an arbitrary element from a set and return a tuple containing that element and the set minus that element.

$$\begin{aligned} &\mathit{elem\_rest} : \mathcal{P}(A) \rightarrow (A, \mathcal{P}(A)) \\ &\mathit{elem\_rest} \quad (T) = \begin{cases} \perp & \text{if } T = \emptyset \\ (x \;.\; x \in T, \{t \;.\; t \in T, t \neq x\}) & \text{otherwise} \end{cases} \end{aligned} \tag{20}$$

4 Conclusion

We have specified and defined an indexed equivalence over semi-Markov processes which is based on a state aggregation of the SMP. The aggregation is $O(n^3)$ for all states in the worst case and $O(n^2)$ for a single state if heavily interconnected.

By preserving passage-times within an SMP we also preserve transient and steady-state results as well, so if a modeller is seeking to perform multiple analysis on an SMP then expending some initial computational effort to reduce the size of the model to concentrate on the key states would be a useful exercise. It is appreciated that if a system is only being analysed once for say steady-state or transient-state results (for which the complexity is $O(n^3)$) then there is not much to be gained from using this technique unless $< O(n)$ states are to be aggregated.

It is envisaged that in much the same way that lumpability [1] has been embodied in various Markovian equivalences, we would like to embed this passage-time equivalence in a process algebraic context. In doing so, of course, we will either need to perform analysis solely at the component level or construct some technique for approximating interleaved concurrent state space as a sequential state space.

References

1. J. G. Kemeny and J. L. Snell, *Finite Markov Chains.* Van Nostrand, 1960.
2. H. Hermanns and M. Rettelbach, "Syntax, semantics, equivalences, and axioms for MTIPP," in *Process Algebra and Performance Modelling Workshop* (U. Herzog and M. Rettelbach, eds.), pp. 69–88, Arbeitsberichte des IMMD, Universtät Erlangen-Nürnberg, Regensberg, July 1994.
3. J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science.* Cambridge University Press, 1996. ISBN 0 521 57189 8.
4. H. Hermanns, *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, July 1998.
5. M. Bernardo and R. Gorrieri, "Extended Markovian Process Algebra," in *CONCUR'96, Proceedings of the 7th International Conference on Concurrency Theory* (U. Montanari and V. Sassone, eds.), vol. 1119 of *Lecture Notes in Computer Science*, pp. 315–330, Springer-Verlag, Pisa, August 1996.
6. R. Pyke, "Markov renewal processes with finitely many states," *Annals of Mathematical Statistics*, vol. 32, pp. 1243–1259, December 1961.
7. J. T. Bradley and N. J. Davies, "A matrix-based method for analysing stochastic process algebras," in *ICALP Workshops 2000, Process Algebra and Performance Modelling Workshop* (J. D. P. Rolim, ed.), pp. 579–590, University of Waterloo, Ontario, Canada, Carleton Scientific, Geneva, July 2000.
8. J. Abate and W. Whitt, "The Fourier-series method for inverting transforms of probability distributions," *Queueing Systems*, vol. 10, no. 1, pp. 5–88, 1992.