Automated product-forms with MEERCAT

Ashok Argent-Katwala
Department of Computing, Imperial College London
ashok@doc.ic.ac.uk

***Abstract*— The Reversed Compound Agent Theorem (RCAT) is a compositional result that uses Markovian process algebra to derive the reversed process of two cooperating continuous time Markov chains at equilibrium, under certain conditions. From this reversed process, together with the given, forward process, the joint state probabilities can be expressed as a product-form.**

We introduce MEERCAT, the first implementation of the RCAT, which classifies models where the theorem can be applied, and generates their product-form solutions.

I. INTRODUCTION

In this paper, we present MEERCAT, our tool for finding and solving Markovian models which are amenable to product-form solution at steady state.

We introduce PEPAinf – an extension to PEPA [1], a popular Markovian process algebra, to allow for certain processes with infinite, but regular, state-spaces. In particular, this allows us to represent networks of queues with infinite buffers which includes many well-known product-form systems.

II. BACKGROUND

Queueing networks

Queueing theory is a very well established discipline, with roots in work by Erlang at the beginning of the last century.

A traditional queue has an arrival process delivering customers into a (possibly finite) buffer which contains any waiting customers, a queueing discipline which determines their order and a service centre with an associated output process.

This may be extended to allow multiple arrival streams, which all feed into the buffer. We may have multiple service centres. Where the departure process is non-Markovian, and jobs may be interrupted, we must consider what happens to an interrupted job. Is it pre-empted, or allowed to run to completion? If pre-empted must it later restart it's task from the beginning. In purely Markovian systems, we have the convenient property that the history of each job makes no difference: pre-emption with restart is indistinguishable from remembering how long the job had in service previously. For generally distributed service-times, we need to keep track of the time served thus far for every customer in service.

Consider a very simple queue – as pictured in Fig. 1. This is an $M/M/1$ queue, in the notation named for Kendall. The Ms indicate that the arrival and departure process are both Markovian, and the 1 indicates that there is a single service centre. When connected together in networks there are a few

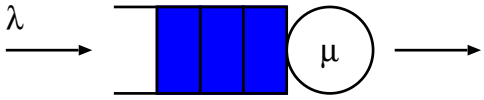

Fig. 1. An $M/M/1$ queue: Both arrival and service processes have a negative-exponential distribution.

choices as to how customers are moved between queues. In the simplest case, a queue's output is routed to a single queue. Where it leads to several possible queues we would typically choose probabilistically, perhaps with weights between the available output actions.

When dealing with finite buffers, a queue's output may be blocked if the destination queue is full, or the customer could be discarded – depending on what discipline the queueing model follows.

There are many choices of the precise behaviour of a queueing system, and so notations that capture all of the possible options may become cumbersome. Moreover, queueing models are often specified relatively informally, relying on the common knowledge within a particular field. When the problem description is in natural language, then there is scope for confusion.

Negative customers: Gelenbe extended queueing networks to allow richer interactions using negative customers [2]. In such generalised queues, dubbed G-queues, a negative customer, on arriving at a non-empty queue, eliminates one of the waiting customers. We apply the RCAT to networks of G-queues (G-networks) within a unified framework, together with other models specified in PEPAinf.

Cascades of triggers are another possibility within G-networks. These allow an arrival, positive or negative, to cause a further effect, either positive or negative, instantaneously at any other queue. A further extension is to allow triggers to cause other triggers, allowing a cascade of triggers to, for example, repeatedly flush a customer from each of a group of queues until one of the queues is empty. We do not consider such instantaneous cascades in this paper.

Product-form solutions

When analysing a queueing network, or any other equivalent formalism, there are conditions under which the behaviour of the whole system can be described in terms of sub-parts of the system operating independently.

For example, consider a pair of queues, as pictured in Fig. 2.

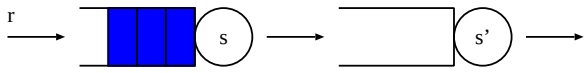

Fig. 2. A tandem pair of $M/M/1$ queues.

This is the simplest form of an *open Jackson network*, which were shown in [3] to always have a product-form steady-state.

In the 1970s, the BCMP theorem [4] described product-form networks of queues similar to Jackson's waiting lines. BCMP covers a much larger class of queues, in particular allowing non-exponential service times, provided the queueing discipline was processor sharing, infinite server or last-come first-served with preemptive-resume.

Gelenbe's seminal work on negative customers [2] led to renewed interest in the area of product-form results, by providing a surprising new class of product-form networks in queues. This has since been extended, particularly in [5], to provide well-characterised, product-form results for complex queueing networks, with signals, triggers, resets and negative customers.

In [6], Boucherie produced a new product-form, centred on agents competing over a shared resource, essentially describing a form of semaphore. The driving example in this case was from Petri-nets, rather than queueing theory.

The Reverse Compound Agent Theorem (RCAT) [7] offers a single framework in which many existing product-form results can be expressed. It works by considering the *reverse process* of the model in question, and – under appropriate conditions – declares that the complete solution can be determined by only considering the reversed processes of its component parts. We examine the RCAT in more detail in III.

PEPA

PEPA [1] is a parsimonious Markovian process algebra that can describe compositional stochastic models. These models consist of components whose actions incorporate random exponential delays, with an underlying interpretation as a continuous-time Markov chain. As such it is an ideal base upon which to build tools for separable solutions of Markov models, in particular for the RCAT.

The syntax of a PEPA component, P, is represented by:

$$\mathrm{P} \quad ::= \quad (a, r).\mathrm{P} \;\Big|\; \mathrm{P} + \mathrm{P} \;\Big|\; \mathrm{P} \underset{S}{\bowtie} \mathrm{P} \;\Big|\; \mathrm{P}/\mathrm{L} \;\Big|\; \mathrm{A}$$

$(a, r).\mathrm{P}$ is a prefix operation. It represents a process which performs the *activity*, (a,r) a, and then becomes a new process, P. An activity has a name and a rate – a and r here respectively. The name is either a visible action, with an alphanumeric name or the silent action τ, which is invisible and cannot be used to synchronise with other components. The activity rate is either a positive, real parameter to an exponentially distributed random variable, or $\top$, denoting an action which is passive in a cooperation (see below).

$\mathrm{P}_1 + \mathrm{P}_2$ is a choice operation. A race is entered into between components P_1 and P_2. If P_1 evolves first then any behaviour of P_2 is discarded and vice-versa.

$\mathrm{P}_1 \underset{S}{\bowtie} \mathrm{P}_2$ is the cooperation operator. P_1 and P_2 run in parallel and synchronise over the set of actions in the set S. If P_1 is to evolve with an action $a \in S$, then it must first wait for P_2 to reach a point where it is also capable of producing an a-action, and vice-versa. In an active cooperation, the two components then jointly produce an a-action with a rate that reflects the slower of the two components (usually the minimum of the two individual a-rates). In a passive cooperation, where P_1, say, can evolve with an $(a, \top)$-transition, the joint a-action inherits its rate from the P_2 component alone.

P/L is a hiding operator where actions in the set L that emanate from the component P are rewritten as silent τ-actions (with the same delays as before). The actions in L can no longer be used in cooperation with other components.

A is a constant label and allows, amongst other things, recursive definitions to be constructed.

The common subset of PEPA components in use today are *cyclic* PEPA components, given by the following syntax rules, as presented in [1]:

$$\begin{array}{lcl} \mathrm{P} & ::= & \mathrm{P} \underset{L}{\bowtie} \mathrm{P} \;\Big|\; \mathrm{P}/\mathrm{L} \;\Big|\; \\ \mathrm{S} & ::= & (a, r).\mathrm{S} \;\Big|\; \mathrm{S} + \mathrm{S} \;\Big|\; \mathrm{A} \end{array}$$

PEPAinf

MEERCAT is designed to help modellers apply product-form techniques in solving their models. Since there are many queueing networks with infinite buffers which have well-known product-form solutions at steady-state, it would be unfortunate if we could not express them..

In order to handle these systems, we introduce PEPAinf, an extension to PEPA syntax to allow components with regular, infinite state spaces. Component names can be augmented with a list of subscripts which range over the integers. This allows for unbounded, but regular state spaces. We also force finite

branching in or out of any state.

P	::=	Name:	|	Name:	'(' VarList	')'
VarList	::=	Var	|	VarList	',' Var	
Var	::=	Varname		[Arg]	[Guard]	
Arg	::=	'+'Num	|	'−'Num		
Guard	::=	'⟩'Num	|	'⟨'Num		

Our aim is to cover a suitable set of unbounded processes to support our work on reverse processes and product-form. For this reason we would like to include, at the very least, support for unbounded $M/M/1$ queues.

The intention of PEPAinf is to formalise definitions like:

$$\begin{aligned} \mathrm{P}_0 &\stackrel{\text{def}}{=} (a, s_1).\mathrm{P}_1; \\ \mathrm{P}_\mathrm{i} &\stackrel{\text{def}}{=} (a, s_1).\mathrm{P}_{\mathrm{i}+1} + (d, s_2).\mathrm{P}_{\mathrm{i}-1} \quad \forall \mathrm{i} > 0; \end{aligned}$$

We deliberately exclude infinite branching either in or out of a state. Infinite branching out of a state would mean introducing difficult problems of how to determine the outbound apparent rate for a transition. Infinite inbound transitions seem to be less troublesome and may be found, for example, in a queue which can flush all of the customers away. However, since we are working with reverse processes infinite branching in either direction means accepting them both in and out of a state. This remains an area of possible future investigation.

Any PEPA process is a valid PEPAinf process, as our only change is to allow some new, structured names. We retain many of PEPA's restrictions – in particular retaining a finite alphabet of action-types, finite branching and a fixed, finite number of processes in any model – while including new, useful processes, particularly with respect to product-form results.

Example: *Counting process*

The simple counting process depicted in Fig. 3 is defined as:

$$\begin{aligned} \mathrm{P}_{\mathrm{i}=0} &\stackrel{\text{def}}{=} (up, s).\mathrm{P}_{\mathrm{i}+1} \\ \mathrm{P}_{\mathrm{i}>0} &\stackrel{\text{def}}{=} (down, r).\mathrm{P}_{\mathrm{i}-1} + (up, s).\mathrm{P}_{\mathrm{i}-1} \end{aligned}$$

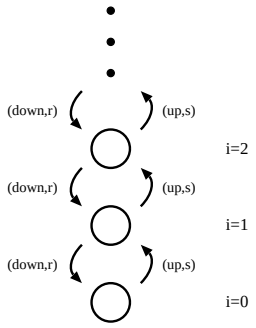

Fig. 3. A simple counting process

III. Reverse Compound Agent Theorem (RCAT)

The RCAT [7] allows us to take a model of two components cooperating which satisfies certain conditions and find a product-form in terms of the individual solutions to the components. It allows us to cover a broad range of previous product-form results in a unified, mechanistic manner.

Before describing the details of our tool, we first consider a simple application of the theorem. This also serves to introduce the reverse processes that are key to how RCAT works.

Consider again the tandem queue as presented in section II, and depicted in Fig. 4 which we now analyse compositionally using the RCAT.

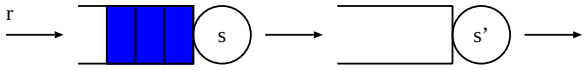

Fig. 4. A tandem pair of queues.

This has a well-known product-form, but to gain some insight into how the RCAT works, we apply it directly.

Original RCAT

From [7]; suppose that the cooperation $P \underset{L}{\bowtie} Q$ has a derivation graph, i.e. Markov chain, with an irreducible subgraph G. Given that:

1) every passive action type in L is always enabled (i.e. enabled in all states of the transition graph);
2) every reversed action of any given active action type in L is always enabled;
3) every occurrence of a reversed action of an active cooperating action type in P (respectively Q) has the same rate in $\overline{P}$ (respectively $\overline{Q}$).

Then, at equilibrium – if it exists – the reversed agent, $\overline{P \underset{L}{\bowtie} Q}$, has a derivation graph containing the reversed subgraph $\overline{G}$ with rates that can be obtained from those in the reversed processes – $\overline{P}$, $\overline{Q}$. We may solve these reverse processes individually, and the steady-state occupation probabilities (SSPs) of our original model are proportionate to the product of the SSPs of $\overline{P}$ and $\overline{Q}$.

A unique solution is guaranteed to exist for models that satisfy the three conditions above [8].

Example: *A tandem-queue*

Let us return to the tandem queue, but with more rigour. In PEPAinf, the system in Fig. 4 is specified as:

$$\begin{aligned}
\mathrm{P_{i=0}} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P_{i+1}};\\
\mathrm{P_{i>0}} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P_{i+1}} + (int, s).\mathrm{P_{i-1}};\\
\mathrm{Q_{i=0}} &\stackrel{\text{def}}{=} (int, \top).\mathrm{Q_{i+1}};\\
\mathrm{Q_{i>0}} &\stackrel{\text{def}}{=} (int, \top).\mathrm{Q_{i+1}} + (out, s').\mathrm{Q_{i-1}};\\
Sys &\stackrel{\text{def}}{=} \mathrm{P_{i=0}} \underset{\{int\}}{\bowtie} \mathrm{Q_{i=0}};
\end{aligned}$$

By convention we name the reverse process of X as as $\overline{\mathrm{X}}$, and an action a corresponds to the reversed action $\overline{a}$ in the reverse process.

Now, to apply RCAT to this we look first at the top-level composition – $P_{i=0} \underset{\{int\}}{\bowtie} Q_{i=0}$ – and check that that the action int is passively enabled in all the derivative states of $Q_{i=0}$ and $\overline{int}$ is passively enabled in all the derivative states of $\overline{\mathrm{P_{i=0}}}$.

If we needed it, we would make the whole reverse process by reversing cooperating active actions to become passive actions. We instead build two separate components (that is, with no passive actions) introducing new, unknown rates. Due to the conditions of RCAT – these processes, $\overline{\mathrm{P}}$ and $\overline{\mathrm{Q}}$, will have the same steady-state behaviour as the cooperation of the whole reversed process, given in Eq. (1).

The rate of $\overline{out}$ in $\overline{\mathrm{Q}}$ is x_{int}, since this is the rate into the Q component in the forward process. $\overline{int}$ in $\overline{\mathrm{Q}}$ proceeds at rate s' and in $\overline{\mathrm{P}}$ as rate r by a similar argument. Thus, we define our $\overline{\mathrm{P_{i=0}}}$ and $\overline{\mathrm{Q_{i=0}}}$ as:

$$\begin{aligned}
\overline{\mathrm{P_{i=0}}} &\stackrel{\text{def}}{=} (\overline{int}, r).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{P_{i>0}}} &\stackrel{\text{def}}{=} (\overline{arr}, s).\overline{\mathrm{P_{i-1}}} + (\overline{int}, r).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{Q_{i=0}}} &\stackrel{\text{def}}{=} (\overline{out}, x_{int}).\overline{\mathrm{Q_{i+1}}};\\
\overline{\mathrm{Q_{i>0}}} &\stackrel{\text{def}}{=} (\overline{out}, x_{int}).\overline{\mathrm{Q_{i+1}}} + (\overline{int}, s').\overline{\mathrm{Q_{i-1}}};
\end{aligned}$$

Notice that in the reversed action of $\mathrm{P_i}$ Where:

$$x_{int} = r$$

by the traffic equations.

The unnormalised SSP that the joint, forward process is in state $\mathrm{P_i}$, $\mathrm{Q_j}$ is the product-form:

$$\pi_{ij} = \pi_P(i)\pi_Q(j)$$

where $\pi_P(i)$ is the SSP that queue P is in state i, and similarly $\pi_Q(j)$ the SSP that Q is in state j.

Should it be needed, the actual reverse process is defined as:

$$\begin{aligned}
\overline{\mathrm{P'_{i=0}}} &\stackrel{\text{def}}{=} (\overline{int}, \top).\overline{\mathrm{P'_{i+1}}};\\
\overline{\mathrm{P'_{i>0}}} &\stackrel{\text{def}}{=} (\overline{arr}, x_{int}).\overline{\mathrm{P'_{i-1}}} + (\overline{int}, \top).\overline{\mathrm{P'_{i+1}}};\\
\overline{\mathrm{Q'_{i=0}}} &\stackrel{\text{def}}{=} (\overline{out}, x_{int}).\overline{\mathrm{Q'_{i+1}}};\\
\overline{\mathrm{Q'_{i>0}}} &\stackrel{\text{def}}{=} (\overline{out}, x_{int}).\overline{\mathrm{Q'_{i+1}}} + (\overline{int}, s').\overline{\mathrm{Q'_{i-1}}};\\
Sys &\stackrel{\text{def}}{=} \overline{\mathrm{P'_{i=0}}} \underset{\{\overline{int}\}}{\bowtie} \overline{\mathrm{Q'_{i=0}}};
\end{aligned} \tag{1}$$

$\overline{\mathrm{P'}}$ and $\overline{\mathrm{Q'}}$ are identical to $\overline{\mathrm{P}}$ and $\overline{\mathrm{Q}}$, replacing the explicit rates we insert for the cooperation with a cooperating action.

Extended RCAT

The theorem has been extended in [9] and [8] to cover a wider range of product-form results. In particular conditions 1 and 2 have been relaxed, which allows the Boucherie product-form of [6] to be handled in a uniform fashion, but at greater expense in checking those conditions. In [8], the theorem is extended to handle multiple agents cooperating together explicitly. In [10], cooperations with joint state-dependent rates (comparable to functional rates) give succinct, separable (but non-product-form) solutions. The BCMP theorem [4] with processor sharing is an example.

IV. Automation and implementation

MEERCAT, takes the description of the top-level cooperation of a pair of RCAT-compliant processes specified in PEPA or PEPAinf and works through several rounds to produce the unnormalised stationary probabilities, π.

- Parsing – to read in the PEPA specification and populate our tables of agents, transitions and cooperations.
- Checking – that the given processes are properly formed and satisfy the conditions of RCAT.
- Reversal – forming the relabelled, reversed processes.
- Deriving – walking the newly-built reversed components to generate the equations we need to solve for the rates, x_a, and so give π.

We now examine the data structures used, then consider these stages in turn.

Data structures

- **System** a mapping of component names to Agents; where PEPAinf is used, the mapping is from the base, unsubscripted name.
- **Agent** has its component name (with any subscripts) and a list of definitions.
- **Definition** either a transition, with rate, action name and the full name of the successor component, or a synchronisation, with cooperation set – a list action names and the names of the two cooperating components.

Parsing

We read the textual description of the processes, in PEPA or PEPAinf, and populate a System component with the model.

By storing the processes as a keyed, associative array we ensure that in the later stages we only ever need to walk a structure that is the size of the description, never the size of the state space. This is especially important since we are not necessarily working with a finite state space and we have chosen a convenient form of regular, infinite process for just this reason.

Our parser accepts a few variants of the PEPA language (chiefly different symbols for $\top$ and the style of expressing choice through repeated definition) but will always emit the form used by the PEPA workbench. This follows the general advice that a program should be "permissive in what it accepts, but strict in what it emits".

MEERCAT uses an ASCII format for PEPAinf. The tandem queue model is defined of section III as:

```
P(i=0) = (arr,r).P(i+1);
P(i>0) = (arr,r).P(i+1) + (int,s).P(i-1);
Q(i=0) = (int,infty).Q(i+1);
Q(i>0) = (int,infty}).Q(i+1)
         + (out,s').Q(i-1);
Sys = P(i=0) <int> Q(i=0);
```

Testing conditions

The MEERCAT tool performs basic checking that the processes used are well-formed – that successor states are defined, all synchronisations are ultimately bound to rates – and that they satisfy the basic conditions for RCAT. In particular, we expect that each action in a cooperation is all-passive in one half of the cooperation and all-active in the other.

For the original RCAT, testing conditions 1 and 2 is simple. While reversing the processes, we maintain a list of the action names and seen rates, and fail with an explanation where they are not appropriately enabled with the same reversed rate for each action-type.

Testing the conditions for the extended RCAT is somewhat harder as they are a function of both processes. Attacking this naively would forego many of the benefits of compositional solution, as we would still be dealing with a joint state space of the cooperating components; an alternative approach working in the joint space of the definitions is considered as future work.

Throughout, we use lazy testing of conditions, alerting once we find a situation which violates a condition. This will enable us to accommodate the extended RCAT in future.

Forming the reverse process

MEERCAT's PEPA module, provides a `relabel` mechanism, which takes either a map of the old action names and rates to their replacements, or functions to call which map the old name and rate onto new ones. The relabelling mechanism stores the relabelled system in a fresh System object, and can optionally invert the direction of each transition to give us a *reversed*, relabelled system.

We use this to walk the definition of our system, inserting new unknowns x_a as the rate for each passive action-type, a, in the cooperation set. It is a condition of RCAT that all synchronising active actions of the same type have the same reversed rate, so we can work in terms of the base component definitions at this level.

We insert these x_a as unknowns in the definitions of the respective forward processes and construct certain rate equations. These equations are known to have a solution by [8] but it is not always known symbolically. In the case of standard queueing networks, the rate equations are linear (equivalent to the traffic equations) and an explicit solution can be found in simple cases. However, since a unique solution is known to exist, it makes more sense to leave the x_a as symbolic variables, which can be substituted by particular values by a numerical rate equation solver for numerical applications.

By holding the definition of the forward process as a list of tuples, forming the reverse process is cheap – proportional to the number of individual component definitions, not the number of component states, or any combination of the components' state spaces.

For a straightforward case:

$$\mathrm{P_i} \stackrel{\mathrm{def}}{=} (a, \top).P_{i+1}$$

becomes

$$\overline{\mathrm{P_i}} \stackrel{\mathrm{def}}{=} (\overline{a}, x_a).\overline{\mathrm{P_{i-1}}}$$

In more complex cases, we follow the rules from [7]. In particular, multiple inbound arcs to the same state in the forward process becomes choice in the reverse process, and we must apportion the total reversed rate in proportion to the forward rates on the newly formed (reversed) arcs. This is the only stage where the subscripting plays an important role in our implementation, as we typically will have transitions within a particular family of subscripted components, but between the different parameterised levels. Since transitions between subscripted process names always change the value of the subscript by plus or minus some constant, this is straightforward to invert. Note, however, that this is an entirely local problem, so we can still handle it in isolation, and never be exposed to the full state space.

By solving for these x_a, preferably symbolicly, we can find

both the joint SSPs and the reversed process of the cooperation.

MEERCAT produces equations in an ASCII syntax which may be fed to Mathematica, or in a basic ASCII form intended for human readability. Further output types can be added by extending the Equation and Expression classes used to store and emit the equations.

MEERCAT will also handle negative customers. There is no greater difficulty applying RCAT, the same process is used to assigning rates in the reversed processes. However, the rate equations that are produced will in general be nonlinear, and so the solver's task is substantially harder, or we must appeal to numerical techniques.

Generating the reversed process

Using the arguments, guards and rates we generate a set of equations for each Component held in the System object giving the rates in the reversed process in terms of the other rates.

This involves visiting each base component definition once, and no dependence on the partner in the cooperation.

Deriving stationary occupation probabilities

By Theorem 1 of [8], the joint equilibrium probability of states i, j in the cooperation $\mathrm{P}_1 \underset{L}{\bowtie} \mathrm{P}_2$ is proportional to the product of the equilibrium probabilities $\pi_1(i)$ and $\pi_2(i)$ of $\overline{\mathrm{P}}$, $\overline{\mathrm{Q}}$ respectively, where these equilibria exist.

It is assumed that π_1 and π_2 are known – either by explicit calculation as a base case or due to a previous application of the RCAT, which we have stored.

Example: *Tandem queue with feedback*

As depicted in Fig. 5, and in PEPAinf:

$$\begin{aligned}
\mathrm{P_{i=0}} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P_{i+1}} + (repeat, \top).\mathrm{P_{i+1}};\\
\mathrm{P_{i>0}} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P_{i+1}} + (repeat, \top).\mathrm{P_{i+1}} +\\
&\quad (int, s).\mathrm{P_{i-1}};\\
\mathrm{Q_{i=0}} &\stackrel{\text{def}}{=} (int, \top).\mathrm{Q_{i+1}};\\
\mathrm{Q_{i>0}} &\stackrel{\text{def}}{=} (int, \top).\mathrm{Q_{i+1}} + (repeat, \mathrm{s}'/2).\mathrm{Q_{i-1}} +\\
&\quad (out, s'/2).\mathrm{Q_{i-1}};\\
Sys &\stackrel{\text{def}}{=} \mathrm{P_{i=0}} \underset{\{int, repeat\}}{\bowtie} \mathrm{Q_{i=0}};
\end{aligned}$$

Then we build the two reverse processes, replacing the passive rates as described above:

$$\begin{aligned}
\overline{\mathrm{P_{i=0}}} &\stackrel{\text{def}}{=} (\overline{int}, r + x_{repeat}).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{P_{i>0}}} &\stackrel{\text{def}}{=} (\overline{arr}, \frac{sr}{r + x_{repeat}}).\overline{\mathrm{P_{i-1}}} +\\
&\quad (\overline{repeat}, \frac{sx_{repeat}}{r + x_{repeat}}).\overline{\mathrm{P_{i-1}}} +\\
&\quad (\overline{int}, r + x_{repeat}).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{Q_{i=0}}} &\stackrel{\text{def}}{=} (\overline{repeat}, x_{int}/2).\overline{\mathrm{Q_{i+1}}} + (\overline{out}, x_{int}/2).\overline{\mathrm{Q_{i+1}}};\\
\overline{\mathrm{Q_{i>0}}} &\stackrel{\text{def}}{=} (\overline{int}, s').\overline{\mathrm{Q_{i-1}}} +\\
&\quad (\overline{repeat}, x_{int}/2).\overline{\mathrm{Q_{i+1}}} +\\
&\quad (\overline{out}, x_{int}/2).\overline{\mathrm{Q_{i+1}}};
\end{aligned}$$

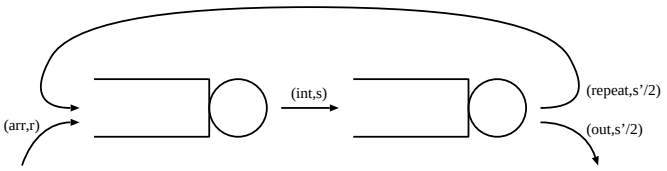

Fig. 5. Tandem queue with feedback

Knowing that these two processes must have the same rates for their cooperating actions, the traffic equations give us:

$$\begin{aligned}
x_{int} &= r + x_{repeat}\\
x_{repeat} &= x_{int}/2
\end{aligned}$$

So,

$$\begin{aligned}
x_{repeat} &= r\\
x_{int} &= 2r
\end{aligned}$$

And we can rewrite our reversed processes as:

$$\begin{aligned}
\overline{\mathrm{P_{i=0}}} &\stackrel{\text{def}}{=} (\overline{int}, 2r).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{P_{i>0}}} &\stackrel{\text{def}}{=} (\overline{arr}, s/2).\overline{\mathrm{P_{i-1}}} + (\overline{repeat}, s/2).\overline{\mathrm{P_{i-1}}} +\\
&\quad (\overline{int}, 2r).\overline{\mathrm{P_{i+1}}};\\
\overline{\mathrm{Q_{i=0}}} &\stackrel{\text{def}}{=} (\overline{repeat}, r).\overline{\mathrm{Q_{i+1}}} + (\overline{out}, r).\overline{\mathrm{Q_{i+1}}};\\
\overline{\mathrm{Q_{i>0}}} &\stackrel{\text{def}}{=} (\overline{int}, s').\overline{\mathrm{Q_{i-1}}} + (\overline{repeat}, r).\overline{\mathrm{Q_{i+1}}} +\\
&\quad (\overline{out}, r).\overline{\mathrm{Q_{i+1}}};
\end{aligned}$$

Then the stationary probability that the entire system is in state $\mathrm{P_i}$, $\mathrm{Q_j}$ is the product-form:

$$\frac{1}{G}\Big(\frac{2r}{s}\Big)^i\Big(\frac{2r}{s'}\Big)^j$$

where G is a normalisation constant, and $(\frac{2r}{s})^i$ and $(\frac{2r)}{s'})^j$ are the SSPs for the reversed, relabelled processes $\overline{\mathrm{P_i}}$ and $\overline{\mathrm{Q_j}}$ respectively.

In this case, G is also the product of the individual normalising constants, since the system is an open network with no constraints on the set of valid states.

$$\frac{1}{G} = \Big(1 - \frac{2r}{s}\Big)\Big(1 - \frac{2r}{s'}\Big)$$

Example: *Tandem queue with feedback & external interactions*

As depicted in Fig. 6, this is the most general form of a pair of $M/M/1$ queues with positive customers. Without loss of generality, it is equally likely to depart as it is transfer to the other queue. Note also that this system is completely symmetric, int is no more internal than $repeat$, it is merely a matter of how you look at the system. In PEPAinf:

$$
\begin{aligned}
\mathrm{P}_{\mathrm{i}=0} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P}_{\mathrm{i}+1} + (repeat, \top).\mathrm{P}_{\mathrm{i}+1};\\
\mathrm{P}_{\mathrm{i}>0} &\stackrel{\text{def}}{=} (arr, r).\mathrm{P}_{\mathrm{i}+1} + (repeat, \top).\mathrm{P}_{\mathrm{i}+1} +\\
&\quad (int, s/2).\mathrm{P}_{\mathrm{i}-1} + (extdep, s/2).\mathrm{P}_{\mathrm{i}-1};\\
\mathrm{Q}_{\mathrm{i}=0} &\stackrel{\text{def}}{=} (extarr, r').\mathrm{Q}_{\mathrm{i}+1} + (int, \top).\mathrm{Q}_{\mathrm{i}+1};\\
\mathrm{Q}_{\mathrm{i}>0} &\stackrel{\text{def}}{=} (int, \top).\mathrm{Q}_{\mathrm{i}+1} + (repeat, s'/2).\mathrm{Q}_{\mathrm{i}-1} +\\
&\quad (out, s'/2).\mathrm{Q}_{\mathrm{i}-1};\\
Sys &\stackrel{\text{def}}{=} \mathrm{P}_{\mathrm{i}=0} \underset{\{int, repeat\}}{\bowtie} \mathrm{Q}_{\mathrm{i}=0};
\end{aligned}
$$

To build the two reversed processes we again replacing the passive rates as described above and form two independently operating queues:

$$
\begin{aligned}
\overline{\mathrm{P}_{\mathrm{i}=0}} &\stackrel{\text{def}}{=} (\overline{extdep}, \frac{r + x_{repeat}}{2}).\overline{\mathrm{P}_{\mathrm{i}+1}} +\\
&\quad (\overline{int}, \frac{r + x_{repeat}}{2}).\overline{\mathrm{P}_{\mathrm{i}+1}};\\
\overline{\mathrm{P}_{\mathrm{i}>0}} &\stackrel{\text{def}}{=} (\overline{arr}, \frac{sr}{r + x_{repeat}}).\overline{\mathrm{P}_{\mathrm{i}-1}} +\\
&\quad (\overline{repeat}, \frac{sx_{repeat}}{r + x_{repeat}}).\overline{\mathrm{P}_{\mathrm{i}-1}}\\
&\quad +(\overline{extdep}, \frac{r + x_{repeat}}{2}).\overline{\mathrm{P}_{\mathrm{i}+1}} +\\
&\quad (\overline{int}, \frac{r + x_{repeat}}{2}).\overline{\mathrm{P}_{\mathrm{i}+1}};\\
\overline{\mathrm{Q}_{\mathrm{i}=0}} &\stackrel{\text{def}}{=} (\overline{repeat}, \frac{x_{int} + r'}{2}).\overline{\mathrm{Q}_{\mathrm{i}+1}} +\\
&\quad (\overline{out}, \frac{x_{int} + r'}{2}).\overline{\mathrm{Q}_{\mathrm{i}+1}};\\
\overline{\mathrm{Q}_{\mathrm{i}>0}} &\stackrel{\text{def}}{=} (\overline{int}, \frac{s' x_{int}}{x_{int} + r'}).\overline{\mathrm{Q}_{\mathrm{i}-1}} +\\
&\quad (\overline{extarr}, \frac{s' r'}{x_{int} + r'}).\overline{\mathrm{Q}_{\mathrm{i}-1}} +\\
&\quad (\overline{repeat}, \frac{x_{int} + r'}{2}).\overline{\mathrm{Q}_{\mathrm{i}+1}} +\\
&\quad (\overline{out}, \frac{x_{int} + r'}{2}).\overline{\mathrm{Q}_{\mathrm{i}+1}};
\end{aligned}
$$

Where the rate equations are:

$$
\begin{aligned}
x_{int} &= \frac{r + x_{repeat}}{2}\\
x_{repeat} &= \frac{r + x_{int}}{2}
\end{aligned}
$$

So,

$$
\begin{aligned}
x_{int} &= r\\
x_{repeat} &= r
\end{aligned}
$$

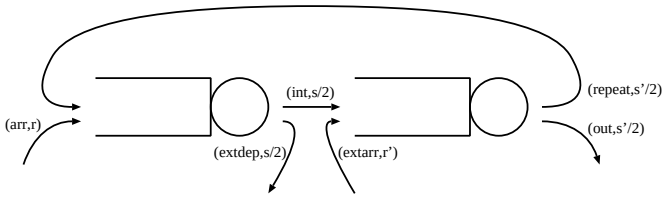

Fig. 6. Tandem queue with feedback and external arrivals and departures to both queues.

V. Conclusion & Future work

We have introduced MEERCAT, the first implementation of the RCAT. We have defined PEPAinf, a modest extension to PEPA to allow certain regular, infinite state-spaces in our Markov processes. This provides a single framework for many product-form steady-state results, and the framework for future automated discovery of such product-forms.

In future, we would like to also describe triggers, potentially involving chains of instantaneous actions. We are investigating using Semi-Markov PEPA [11], to describe our input processes to achieve this.

Handling batches and resets will require an extension to PEPAinf to handle unbounded numbers of transitions in or out of a state. This would need to be accompanied by clear semantics of when a process with unbounded branching was well-defined. In particular, there would need to be a finite apparent rate for each action, and there are likely other complications. Despite these difficulties, defining such structures

would be useful, since they represent a set of behaviours that is not only interesting, but have been shown to have tractable solutions.

The MEERCAT tool may be used interactively on the PerformDB web site, or downloaded to be run locally:
⟨http://performdb.org/tools/meercat/⟩

ACKNOWLEDGEMENTS

The author would like to thank Peter Harrison and Jeremy Bradley for their insight & support and his PhD examiners, Stephen Gilmore and David Rosenblum, for their helpful comments and suggestions. He is supported by EPSRC under the PerformDB grant EP/D054087/1.

REFERENCES

[1] J. Hillston, *A Compositional Approach to Performance Modelling*. PhD thesis, Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK, 1994. CST–107–94.
[2] E. E. Gelenbe, "Queuing networks with negative and positive customers," *Journal of Applied Probability*, vol. 28, pp. 656–663, 1991.
[3] J. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5, pp. 518–521, 1957.
[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, no. 2, pp. 248–260, 1975.
[5] X. Chao, M. Miyazawa, and M. Pinedo, *Queueing Networks: Customers, Signals and Product Form Solutions*. Wiley, 1998.
[6] R. J. Boucherie, "A characterization of independence for competing markov chains with applications to stochastic petri nets," *IEEE Trans. Softw. Eng.*, vol. 20, no. 7, pp. 536–544, 1994.
[7] P. Harrison, "Turning back time in Markovian process algebra," *Theoretical Computer Science*, no. 290, pp. 1947–1968, 2003.
[8] P. Harrison and T. Lee, "Separable equilibrium state probabilities via time reversal in Markovian process algebra," *Theoretical Computer Science*, no. 346, pp. 161–182, 2005.
[9] P. Harrison, "Reversed Processes, product forms and some non-product forms," *Linear Algebra and Its Applications*, pp. 359–381, 2004.
[10] P. Harrison, "Process algebraic non-product-forms," in *Proceedings of the 2nd International Workshop on Practical Applications of Stochastic Modelling*, 2005.
[11] J. Bradley, "Semi-Markov PEPA: Modelling with generally distributed actions," *International Journal of Simulation*, vol. 6, pp. 43–51, February 2005.