Continuous PEPA Queues: Individual behaviour in continuous queueing networks

Ashok Argent-Katwala
Dept. of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, UK
ashok@doc.ic.ac.uk

Jeremy T. Bradley
Dept. of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, UK
jb@doc.ic.ac.uk

ABSTRACT

The PEPA Queues formalism augments queueing networks with customers that have behavioural characteristics defined in the popular stochastic process algebra, PEPA. Networks of PEPA Queues suffer from the traditional state space explosion that affects both closed queueing networks and PEPA models. We present a technique that converts a discrete extension of PEPA Queues into a continuous state space approximation. We show how Continuous PEPA Queues can be analysed using ordinary differential equations, the solution of which provides a time-series plot of the number of components in a particular state in a particular buffer.

Keywords

PEPA, queueing networks, continuous approximation, ODEs

1. INTRODUCTION

Queueing networks are a well-used mechanism for capturing interrelated contention for resources in computer and communication systems [1, 2]. In particular, queues are useful when the contention has to be managed and ordered to give fair, often first-come-first-served, access to a service. Many extensions have been created and analysed to simulate individual or class behaviour to jobs or customers in queueing nodes. PEPA Queues seek to add specific behaviour in a systematic and component-oriented way.

A PEPA Queues network [3] is a behavioural extension of a closed queueing network in a similar sense that PEPA nets [4] is a behavioural extension of stochastic Petri nets. In PEPA Queues, the customers and servers are modelled as PEPA components which may evolve independently. The servers can cooperate with queueing customers to determine how they are routed to other queues in the system.

We presented the first steps for a continuous model for PEPA Queues in [5]. Here we develop that approach by formally introducing to PEPA Queues the notion of layers or ranks of customers within each queueing node's buffer. Rank measures the distance that a customer is away from service. In a standard queue, the rank is the queueing position of a customer and we define a customer with rank 0 to be in service. In our extension, many customers can share the same rank and when an individual customer completes service, one customer from each rank is chosen to progress to the rank in front, thus creating system of managed overtaking.

We additionally present a well-defined semantics for discrete PEPA Queues with ranks. We outline a mechanical generation of the ordinary differential equations that represent a network of PEPA Queues and thus provide the ability to perform systematic continuous state-space analysis of large PEPA Queues models.

In Section 2, we give an overview of the supporting technologies, PEPA and PEPA Queues. Section 3 extends PEPA Queues to allow several customers to share a queueing position or rank. In Section 4, we build a continuous analogue of these PEPA Queues, and how we describe a network of Continuous PEPA Queues, including a translation into ordinary differential equations. A short example modelling a Web server with a waiting pool of clients follows in Section 5. We discuss future work and conclude in Section 6. Appendix A contains the syntax and structural operational semantics of Continuous PEPA Queues.

2. BACKGROUND

2.1 PEPA

PEPA [6] is a parsimonious stochastic process algebra that can describe compositional stochastic models. As in all process algebras, systems are represented in PEPA as the composition of *components* which undertake *actions*. In PEPA the actions are assumed to have a duration, or delay. Thus the expression $(\alpha, r).P$ denotes a component which can undertake an α action at rate r to evolve into a component P. Here $\alpha \in \mathcal{A}$ where $\mathcal{A}$ is the set of action types and $P \in \mathcal{C}$ where $\mathcal{C}$ is the set of component types. The rate r represents the parameter of an exponential distribution, and the duration is assumed to be a random variable.

PEPA has a small set of combinators, allowing system descriptions to be built up as the concurrent execution and interaction of simple sequential components. The syntax of the type of PEPA model considered in this paper may be formally specified using the following grammar:

$$\begin{aligned} S &::= (\alpha, r).S \mid S + S \mid C_S \\ P &::= P \underset{L}{\bowtie} P \mid P/L \mid C \end{aligned}$$

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a

Valuetools '07, October 23-25, 2007, Nantes, France
Copyright 2007 ICST 978-963-9799-00-4.

constant which denotes either a sequential component or a model component as introduced by a definition. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes. The shorthand $\parallel$ is used to represent cooperation over the empty set, $\underset{\emptyset}{\bowtie}$. More information including the structured operational semantics for PEPA can be found in [6].

2.2 Immediate actions and SM-PEPA

In this paper, we use an extension to PEPA which includes immediate actions. We take a subset of semi-Markov PEPA, which allows a limited form of generally distributed delays giving rise to an underlying semi-Markovian process. We limit these general delays to immediate delays, and have just two priority levels. If there are any enabled immediate actions, they take precedence. Where there is a choice of immediate evolutions, we make a probabilistic choice according to weights assigned to the actions. We do not allow cooperation between the immediate priority level and ordinary timed, Markovian activities.

Semi-Markov PEPA [7] (SM-PEPA), is a version of PEPA that allows general distributions as well as exponential distributions from the standard PEPA model. The syntax for SM-PEPA is given below:

$$P \quad ::= \quad (a^{[n]}, D).P \;\Big|\; P + P \;\Big|\; P \underset{L}{\bowtie} P \;\Big|\; P/L \;\Big|\; A$$

where:

$$D \quad ::= \quad \lambda \;\Big|\; \omega : L(s)$$

where λ is the standard PEPA exponential rate parameter:

$$\lambda \in \mathbb{R}^+ \cup \{r\top \mid r \in \mathbb{Q}, r > 0\}$$

The action a is annotated with a priority $n \in \mathbb{N}$ (where a larger n indicates a higher priority). SM-PEPA introduces a notion of priority enabling where an action is priority enabled only if it is enabled in the normal PEPA sense and there are no higher priority actions that are enabled at the same time. The D variable indicates a duration, either an exponential rate or a weighted general distribution. The general distribution is specified in terms of its Laplace transform for numerical convenience. The weights, ω, are used to select probabilistically between concurrently priority-enabled generally distributed actions.

The use of priorities in activities (action-duration pairs) is restricted so that within a particular priority level, either Markovian activities are available (containing standard PEPA) or generally-distributed activities are. This prevents the simultaneous racing of exponential and generally-distributed distributions. A detailed semantics for SM-PEPA can be found in [7].

The immediate transition model required for use with PEPA Queues can be derived from a subset of SM-PEPA; it uses a similar approach as that used in generalised stochastic Petri nets (GSPNs) [8]. For this purpose only two priority levels are required, level 1 for Markovian activities and level 2 for immediate actions. We use the standard PEPA prefix notation $(a, \lambda).P$ to mean $(a^{[1]}, \lambda).P$ in SM-PEPA and the reduced immediate prefix notation $a.P$ to mean $(a^{[2]}, 1 : 1).P$ (mainly used in the translation Appendix B). This gives each immediate transition equal weight. The immediate transition aspect is represented by the Laplace transform, $L(s) = 1$.

2.3 PEPA Queues

PEPA Queues (presented in detail in [3]) uses PEPA to provide a rich description of customers and service centres within a queueing network. They go beyond traditional mechanisms to allocate a simple class to an otherwise opaque customer by giving a clear description of the behaviour of a customer in isolation. All the customers and servers may evolve independently, according to their PEPA description. A customer in service cooperates with the server component at a queueing node, over a set of actions. Customers waiting in a queue cannot perform actions in that queue's cooperation set.

For example, a very simple PEPA Queue, as depicted in Figure 1, consists of a server component, S, cooperating with the head of the queue, P, over a set of actions, L, while several customers, P$'$, P$'$, P$''$, wait their turn.

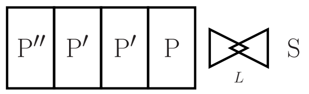

Figure 1: A single PEPA Queue.

Customers move through the queueing network according to where the server sends them. Each queue's server may have a number of routing actions, which determine the destination queue of the customer when it is served. There are no losses in the network, if a destination queue has no room for new customers then that route is simply blocked. For example, Figure 2 depicts a small PEPA Queues network. There are three customers waiting at first queue, and just one (and two empty slots) at the second. The P customer can be sent by the first server if it performs a *send* action. However, the P$'$ customer, waiting at the front of the second queue, cannot be served because the first queue is full. If *send* was in the cooperation set, L, then customer P and server R would need to cooperate to complete the service and send P to the other queue.

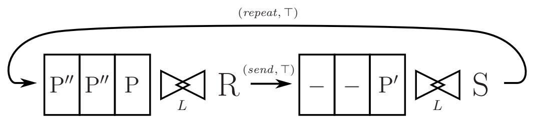

Figure 2: A small network of PEPA Queues.

3. PEPA QUEUES WITH RANKS

In PEPA Queues, the state of a queue is given by the state of all of the customers, in order, together with the state of the server component. For long queues, expressing the exact order of the waiting customers leads to a very large state space. As a compromise, in the extension presented in this paper, we allow customers to share queueing positions.

3.1 Overview

We do this by extending PEPA Queues to have several ranks of waiting customers. Here the ranks have an ordering down

which a customer must progress, before reaching rank 0 where it can be serviced. Within a rank, which may contain many customers, there is no ordering of customers.
By choosing ranks all of size one, we obtain the original PEPA Queues. However, by using larger ranks we can group many similar states together. This allows the modeller to choose an appropriate balance between precision and complexity. For example, we may wish to represent the first few queueing positions exactly, but further back in the buffer we may be happy to aggregate those states with the same customers but a slightly different order.
Consider the system pictured in Figure 3. Each of the ranks has a fixed size. When a customer from the first rank is served, this creates a space at the server rank. One customer is chosen at random from the rank behind to replace them, and so on back through all the ranks.

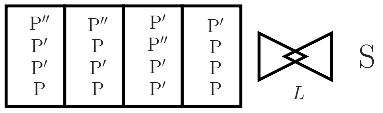

Figure 3: A queue with customers waiting in ranks.

3.2 Extended Syntax to include Ranks

While the picture above is a useful one, we also want a precise syntax for our systems. The description of a system of PEPA Queues has three distinct parts:

- The customers and servers that inhabit the network are described in standard PEPA.
- The routing between queues within the network, describing the actions and rates of the routing actions and where they lead.
- The initial state of the system, including how many ranks there are at each queue and the size of each rank, is part of the *queueing network system equation*.

The routing description contains one definition for each queue in the network, with routing actions with rates leading to particular queues. This is as in the original PEPA Queues. If the only routing action out of queue A is an α-action, at rate r, leading to queue B, then we would say:

$$\mathrm{Q\!:\!A} \overset{def}{=} (\alpha, r) \rightarrow \mathrm{Q\!:\!B}$$

The queueing network system equation is a little different than in the earlier PEPA Queues, since we need to describe the ranks of waiting customers at each queue. We do this by representing the state of a queue's buffer using a multiset for each of the ranks. The queue pictured in Figure 3 is written:

$$[\{P'', P', P', P\}|\{P'', P', P{*}2\}|\{P'', P'{*}3\}|\{P', P{*}3\}] \underset{L}{\bowtie} S$$

$P{*}3$ is a shorthand to represent three copies of P. If we omit the braces, we are implicitly defining ranks of size one. We use bars to separate the different ranks here. Alternatively, the ranks may be separated by commas, as in the original PEPA Queues syntax.
So a simple system of two queues would be described as follows. We omit the server and customer descriptions for brevity:

$$\mathrm{Q\!:\!A} \quad \overset{def}{=} \quad (\alpha, r) \rightarrow \mathrm{Q\!:\!B}$$

$$\mathrm{Q\!:\!B} \quad \overset{def}{=} \quad (\alpha, r) \rightarrow \mathrm{Q\!:\!A}$$

$$\begin{aligned} \mathrm{QNet\!:\!Sys} \quad \overset{def}{=} \quad & (\mathrm{Q\!:\!A}[\{-*\mathrm{m}\}|\{\mathrm{P}\}] \underset{L}{\bowtie} \mathrm{S_A}) \\ & \mid\mid (\mathrm{Q\!:\!B}[\{-*\mathrm{n_w}\}|\{\mathrm{P}*\mathrm{n_s}\}] \underset{L}{\bowtie} \mathrm{S_B}) \end{aligned}$$

The first queue has a single space at the server, with an empty buffer of m spaces of places to wait. The second queue has n_s P-customers in service, and space for n_w to wait. The two queues are composed in parallel, but we could allow them to synchronise on a set of actions, which would be useful in modelling a system with communicating servers.

3.3 Translation

To analyse a network of PEPA Queues we translate it into an equivalent, flat PEPA model and analyse that. The full details of the translation are given in Appendix B. In this section we focus on how we fairly promote a single customer from each rank to the rank in front.

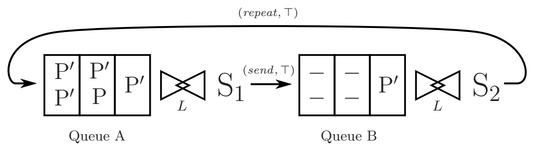

Figure 4: A small network of PEPA Queues with ranks.

Each queue in the system depicted in Figure 4 has three ranks of customers: the service rank has one slot, the two waiting ranks have two slots.
Our translation represents each customer as a single component. The state of the component represents both the customer's internal state, and also their position within the queueing network. Thus, a P′ customer in rank 1 of queue A is represented by $\mathrm{P}'_{\mathrm{A1}}$ in the flat model. There are no explicit components to represent the blank spaces.
As part of our flat model, we have a cooperation of each of these customers, cooperating over all the actions which route customers in the network, L_c:

$$\begin{aligned} \mathrm{Customers} \quad \overset{def}{=} \quad & \mathrm{P}'_{\mathrm{A2}} \underset{L_c}{\bowtie} \mathrm{P}'_{\mathrm{A2}} \underset{L_c}{\bowtie} \mathrm{P}'_{\mathrm{A1}} \underset{L_c}{\bowtie} \mathrm{P}_{\mathrm{A1}} \\ & \underset{L_c}{\bowtie} \mathrm{P}_{\mathrm{A1}} \underset{L_c}{\bowtie} \mathrm{P}_{\mathrm{A0}} \underset{L_c}{\bowtie} \mathrm{P}_{\mathrm{B0}} \end{aligned}$$

When the P at the head is served, we would like to pick one of the customers in rank 1 to replace it. In the original PEPA Queues each waiting customer cooperated on the routing actions for its queue. When a waiting customer witnessed a service, it moved forward a place in the queue. Now, however, we want just one of the waiting customers in a rank to move forward. We do this using immediate actions. We have the same cooperation as before, but instead of moving forward a place, all the customers in the rank move to an intermediate state. In the intermediate state each offers a choice of immediate actions, either to fall back to the previous rank, or move forward to the rank in front.
In the component descriptions that follow, the labels in_A and out_A are shorthand for a choice of all the routing actions into and out of queue A. In this example the in_A actions will correspond to the *send* actions, and out_A to the *repeat* actions. In the translation we have a distinct action-type for each rank of the destination queue. This ensures that we move a customer to the first rank with space in, and never

leave gaps in front of a waiting customer. This is detailed explicitly in Appendix B.

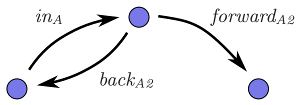

Figure 5: Example of a customer moving from rank 2 to 1 of queue A.

This collection of components cooperate with Rank_{A1}, depicted in Figure 6, over $forward_{A1}$ and $back_{A1}$. This ensures that exactly one of the components moves forward, and all the rest fall back.

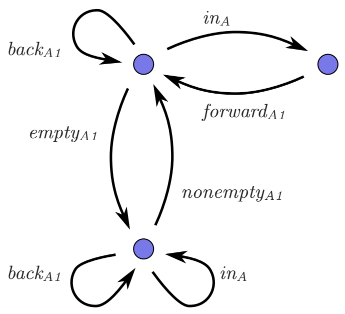

Figure 6: Governing component for Rank A1.

The governing components for each rank cooperate over all the routing actions for that queue. Thus, it is important that when there are no components in the rank that it still allows those actions to take place, without trying to move a customer forward. To achieve this, the population tracker for queue A signals to each rank when it becomes empty or non-empty, as depicted in Figure 7.

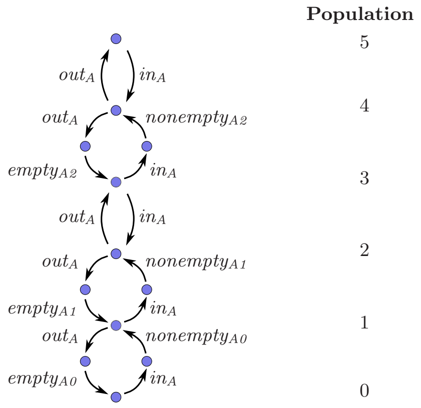

Figure 7: Population tracker for queue A in example.

In the whole system, we have one of these governing components per rank, cooperating with one another over the routing actions for their queue.

4. CONTINUOUS PEPA QUEUES

When the PEPA Queue ranks hold many customers of the same type, we can sensibly approximate the state of an individual rank using real counts of the number of component types, rather than integers. This gives us a continuous representation of PEPA Queues. This is based on the same principle as presented in [9], where components of the same type in a parallel cooperation are represented as a tuple of real variables, one variable for each distinct component state. We adopt a similar approach below.

4.1 An individual Continuous PEPA Queue

Suppose we have an isolated Continuous PEPA Queue, with just two ranks for customers: in service, or waiting. We have arrivals from some external source of P components at a rate λ, as depicted in Figure 8.
The customers and server are defined as follows. The P components decay to P′ independently, but must cooperate with the server to go back to being P.

$$\begin{aligned} \mathrm{P} &\overset{def}{=} (decay, \mu).\mathrm{P}' \\ \mathrm{P}' &\overset{def}{=} (reset, \top).\mathrm{P} \\ \mathrm{S} &\overset{def}{=} (reset, \nu).\mathrm{S} + (send, \top).\mathrm{S} \end{aligned}$$

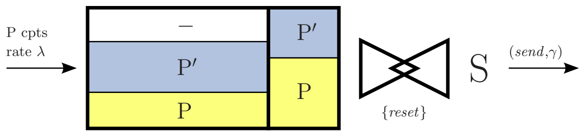

Figure 8: A graphical representation of an individual Continuous PEPA Queue, where P components are arriving at rate λ.

The state of this queue can be given by the number of P, P′ and blank spaces there are in each of the ranks. We denote the number of P components in rank i of queue A at time, t, as $n(A, i, P, t)$. Rank 0 contains the customers who are in service. As with discrete PEPA Queues, each rank has a fixed size, so $n(A, i) = n(A, i, P, t) + n(A, i, P', t) + n(A, i, -, t)$ is constant for a given queue A and rank i. For this reason, when writing down the differential equations describing the evolution of $n(A, i, X, t)$ we omit the equation for empty slots in a rank, since it can be derived from the other equations.
The differential equations for the single PEPA queue of Figure 8 are set out below:

$$\begin{aligned} n'(A,0,P,t) &= T(A,0,t)\lambda + n(A,0,P',t)\nu - n(A,0,P,t)\mu \\ &\quad - Ra(A,0,P,t)\gamma + Ra(A,1,P,t)\gamma \\ n'(A,0,P',t) &= -n(A,0,P',t)\nu + n(A,0,P,t)\mu \\ &\quad - Ra(A,0,P',t)\gamma + Ra(A,1,P',t)\gamma \\ n'(A,1,P,t) &= T(A,1,t)\lambda - n(A,1,P,t)\mu - Ra(A,1,P,t)\gamma \\ n'(A,1,P',t) &= n(A,1,P,t)\mu - Ra(A,1,P',t)\gamma \end{aligned}$$

where $T(A, i, t) = 1$ if rank i of queue A is capable of accepting a new component and 0 otherwise. $Ra(A, i, X, t)$ represents the ratio of X components in the ith rank of queue A relative to the other non-blank components. It is used to determine the rate of advancement of components from one rank to the next when a quantity of components leaves the head of the queue, rank 0. Both of these quantities are defined formally below.

4.2 Translating to differential equations

Based on the top level PEPA Queue model, we can construct coupled differential equations to describe how the number of each X-component, $n(Q,r,X,t)$ varies in each queue, Q, rank r, with time t. In general, this construction is generated for $r > 0$:

$$\begin{aligned} & n'(Q,r,X,t) \\ = \quad & -\text{rate of change of } X \text{ to } X' \text{ in rank } r \\ & +\text{rate of change of } X'' \text{ to } X \text{ in rank } r \\ & -\text{rate of departure of } X \text{ from rank } r \text{ to } r-1 \\ & +\text{rate of arrival of } X \text{ from rank } r+1 \text{ to } r \\ & +\text{rate of arrival of } X \text{ from upstream queue} \\ & \qquad \text{into rank } r \end{aligned}$$

and for $r = 0$:

$$\begin{aligned} & n'(Q,r,X,t) \\ = \quad & -\text{rate of change of } X \text{ to } X' \text{ in rank } r \\ & +\text{rate of change of } X'' \text{ to } X \text{ in rank } r \\ & -\text{rate of departure of } X \text{ to downstream queue} \\ & +\text{rate of arrival of } X \text{ from rank } r+1 \text{ to } r \\ & +\text{rate of arrival of } X \text{ from upstream queue} \\ & \qquad \text{into rank } r \end{aligned}$$

The rate of change of a component type within a rank is determined by the allowed component evolutions within the context of the PEPA Queue it is in. The rate of departure of a component type from a rank is either the rate of departure of that component type from the queue if it is in the 0 rank or the rate of transfer of that component type to a lower rank. For the 0 rank, the component that may depart the queue is defined by the PEPA Queue process. For a component type in one of the higher ranks, the rate of flow into the next lower rank is a proportion of the total component exit rate from the queue, that is, as components leave the queue (from the 0 rank), this will trigger a flow of components from higher to lower ranks within the PEPA Queue. The exact proportion of the overall departure rate that governs this flow is given by the proportion of components of that type that exist in that rank, $Ra(Q,r,X,t)$, defined below.

The T-function used in the earlier example is expressed in terms of indicator functions, $I(Q,r,t)$. The indicator function determines whether there is any space in the rank r for components in rank $r+1$ to flow into; defined by:

$$\begin{aligned} I(Q,r,t) &= I(n(Q,r,-,t) > 0) \\ n(Q,r,-,t) &= \underbrace{n(Q,r) - \sum_i n(Q,r,X_i,t)}_{\text{no. of blank slots available in rank } r \text{ at time, } t} \end{aligned}$$

where $n(Q,r)$ is the static size of rank r as before. This allows us to define the T-function more formally. For rank $r = 0$, $T(Q,0,t) = I(Q,0,t)$ and for rank $r > 0$:

$$T(Q,r,t) = I(Q,r,t) \times \prod_{i=0}^{r-1} (1 - I(Q,i,t))$$

Finally, we can formally define $Ra(Q,r,X,t)$ which gives the ratio of component X that currently exists in the rank r relative to all the other components in the same rank.

$$Ra(Q,r,X,t) = \frac{n(Q,r,X,t)}{\sum_i n(Q,r,X_i,t)}$$

5. EXAMPLE

The network depicted in Figure 9 has a pool of up to m Web clients waiting to connect to a Web server. After connecting, they send a request and receive a page in return, after which they return to the pool of waiting clients. The cooperation set at each queue is $L = \{send\_page, send\_request\}$. Since Distributor never performs a *send_page*, and Server never performs *send_request*, those actions are effectively blocked when at the respective queue.

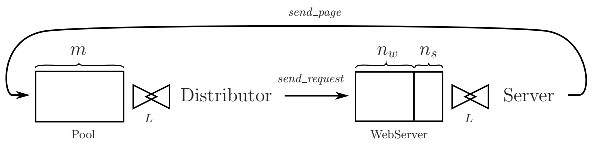

Figure 9: A Web server with a pool of waiting clients.

$$\begin{aligned}
\text{Client} &\stackrel{def}{=} (dns\_lookup, \lambda_{dns}).\text{Client}_{\text{ready}} \\
\text{Client}_{\text{ready}} &\stackrel{def}{=} (tcp\_connect, \lambda_{conn}).\text{Client}_{\text{connected}} \\
&\qquad + (dns\_expire, \lambda_{expire}).\text{Client} \\
\text{Client}_{\text{connected}} &\stackrel{def}{=} (send\_request, \lambda_{req}).\text{Client}_{\text{waiting}} \\
&\qquad + (tcp\_close, \lambda_{close}).\text{Client}_{\text{ready}} \\
\text{Client}_{\text{waiting}} &\stackrel{def}{=} (send\_page, \top).\text{Client}_{\text{connected}} \\
\\
\text{Distributor} &\stackrel{def}{=} (send\_request, \top).\text{Distributor} \\
\\
\text{Worker} &\stackrel{def}{=} (page\_update, \mu_{update}).\text{Worker}_{\text{writing}} \\
&\qquad + (send\_page, \mu_{send}).\text{Worker}_{\text{sent}} \\
\text{Worker}_{\text{writing}} &\stackrel{def}{=} (write\_page, \mu_{write}).\text{Worker} \\
\text{Worker}_{\text{sent}} &\stackrel{def}{=} (write\_log, \mu_{writelog}).\text{Worker} \\
\text{Server} &\stackrel{def}{=} \text{Worker}[\text{n}_\text{s}] \\
\\
\text{Q:Pool} &\stackrel{def}{=} (send\_request, \top) \rightarrow \text{Q:WebServer} \\
\text{Q:WebServer} &\stackrel{def}{=} (send\_page, \top) \rightarrow \text{Q:Pool} \\
\text{QNet:Sys} &\stackrel{def}{=} (\text{Q:Pool}[\{\text{Client} * \text{m}\}] \underset{L}{\bowtie} \text{Distributor}) \\
&\qquad || \; (\text{Q:WebServer}[\{- * \text{n}_\text{w}\}|\{- * \text{n}_\text{s}\}] \\
&\qquad\qquad \underset{L}{\bowtie} \text{Server})
\end{aligned}$$

Clients wait in the pool, and must make a DNS lookup, then establish a connection with the server. Only then can the client send the request, in cooperation with the distributor, and join the queue for service. After being serviced, the client may re-use the same connection for the next request, or the connection may be closed, and the client will need to re-connect. The server has the capacity to serve n_s clients, and a further n_w can be waiting in the server's buffer. Once those buffers are full, clients cannot send their requests until some space becomes available.

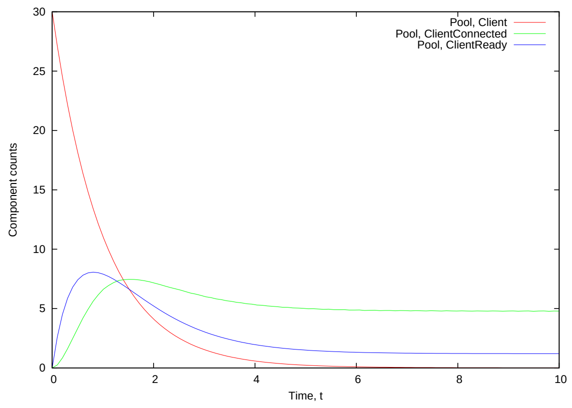

Figure 10: Example solutions of the number of component types in the Pool queue from the web server example.

The server process is made up of n_s worker threads, each of which may process any of the customers in service. After serving a customer with the *send_page* action, the worker thread performs some housekeeping tasks (*write_log*) before returning to service other customers. An individual thread may also update some local information, so be unavailable to service customers until writing the changes back to disk. Figure 10 shows some example solutions of the ODEs from the web example. We can obtain plots of the numbers of component-types in particular queues: in this case the Pool queue. The web example was run with $m = 30$ client components, and buffer sizes of $n_w = n_s = 5$. The explicit state space of even this modest model far exceeds the ability of existing discrete state-space analysers. This model alone would require at least $4^m \times 4^{n_w} \times 4^{n_s}$ discrete states to keep track of all the possible combinations of states of client in each of the 3 client queueing locations.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented and extended the PEPA queues formalism as a methodology for describing process behaviour in a queueing environment. Our main extension has seen the introduction of ranks, where components of the same rank in a queue are seen as a fixed distance away from service. This idea extends the normal queueing notion of queueing position by essentially allowing many customers to occupy the same queueing position. We further introduce an analysis technique to cope with the large state spaces generated by the ranked PEPA queues. This analysis technique, based on the PEPA model continuisation of [9], allows PEPA queues to be transformed into a series of ordinary differential equations. These ODEs can be solved using standard techniques to return plots of component counts in given queues in a network.

This represents preliminary work in a new area of behavioural queues, and we would like to develop it by formalising the generation of ODEs. As future work, we would also like to perform comparisons with modelling styles and results from related fluid formalisms such as fluid queueing networks [10], FSPNs [11, 12] and continuous Petri nets [13]. Although, in these cases we will not need most of the behavioural features of Continuous PEPA Queues.

7. REFERENCES

[1] L. Kleinrock, *Queueing Systems, Volume I: Theory.* John Wiley & Sons, 1975.

[2] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications.* John Wiley & Sons, 1976.

[3] A. Argent-Katwala and J. T. Bradley, "PEPA Queues: Capturing customer behaviour in queueing networks," in *QAPL'07, 5th Workshop on Quantitative Aspects of Programming Languages (Preliminary Proceedings)*, pp. 7–23, March 2007.

[4] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo, "PEPA nets: A structured performance modelling formalism," *Performance Evaluation*, vol. 54, no. 2, pp. 79–104, 2003.

[5] A. Argent-Katwala and J. T. Bradley, "A Continuous State Space Approximation for PEPA Queues," in *Process Algebra and Stochastically Timed Activities 2006*, June 2006.

[6] J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996.

[7] J. T. Bradley, "Semi-Markov PEPA: Modelling with generally distributed actions," *International Journal of Simulation*, vol. 6, pp. 43–51, January 2005.

[8] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, pp. 93–122, May 1984.

[9] J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.

[10] A. J. Field and P. G. Harrison, "An Approximate Compositional Approach to the Analysis of Fluid Queue Networks," in *Performance'07, 26th International Symposium on Computer Performance, Modelling, Measurements and Evaluation*, (Cologne), Elsevier, 2007. (To appear).

[11] G. Horton, D. Nicol, V. Kulkarni, and K. Trivedi, "Fluid stochastic Petri nets: Theory applications and solution techniques," *European Journal of Operational Research*, vol. 105, pp. 184–201, February 1998.

[12] G. Ciardo, D. Nicol, and K. Trivedi, "Simulation of fluid stochastic Petri nets," *IEEE Transactions on Software Engineering*, vol. 25, pp. 207–217, March/April 1999.

[13] M. Silva and L. Recalde, "Petri nets and integrality relaxations: A view of continuous Petri net models," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, no. 4, pp. 314–327, 2002.

APPENDIX

A. SEMANTICS

These rules describe how a system of PEPA Queues with ranks evolves. We represent the buffer of an individual queue as a list of multisets. Each element of the list represents a single rank, and through any evolution of the network will always have the same size. The items in the multiset are customer components, or the blank ($-$) type to explicitly represent free space in the rank.

Definition A.1. *elem*(R) *selects one of the non-blank elements of the multiset at random, with a probability proportionate to the number of that element in the multiset. It is undefined if R only contains blanks.*

Definition A.2. $r_\alpha(P)$ *is the apparent rate function, as defined in [6].*

Definition A.3. $r'_\alpha(A)$ *is a secondary apparent rate function, for queue routing actions. α must be a routing action for* Q : A *and $r'_\alpha(A)$ is the sum of the rates of all the enabled α-activities in the queueing network, for* Q : A *in the present state. It does not include routing actions leading to a queue with a full buffer.*

Customer alone
This rule governs a customer evolving independently, anywhere in the queue. Note that this only allows actions that are not in the queue's cooperation set. Customers that are not in service may not perform those actions, and customers that are in service perform them via rule *Local cooperation*.

$$\frac{P \xrightarrow{(\alpha,\lambda)} P'}{Q : A[\ldots|R|\ldots] \bowtie_{L} S \xrightarrow{(\alpha,\lambda)} Q : A[\ldots|R'|\ldots] \bowtie_{L} S} \quad {\scriptstyle P\in R, \alpha\notin L}$$

where $R' = R - \{\!|\, P \,|\!\} + \{\!|\, P' \,|\!\}$
Server alone
The server process may perform actions not in the queue cooperation set independently, no matter the state of the buffer.

$$\frac{S \xrightarrow{(\alpha,\lambda)} S'}{Q : A[\ldots] \bowtie_{L} S \xrightarrow{(\alpha,\lambda)} Q : A[\ldots] \bowtie_{L} S'} \quad {\scriptstyle \alpha\notin L}$$

Local cooperation
Only the head of the queue may cooperate with the server process for actions in L, and they must perform them together.

$$\frac{P \bowtie_{L} S \xrightarrow{(\alpha,\lambda)} P' \bowtie_{L} S' \quad Q : A \not\mapsto^{\alpha}}{Q : A[\ldots|R] \bowtie_{L} S \xrightarrow{(\alpha,\lambda)} Q : A[\ldots|R'] \bowtie_{L} S'} \quad {\scriptstyle \alpha\in L, P\in R}$$

where $R' = R - \{\!|\, P \,|\!\} + \{\!|\, P' \,|\!\}$
Server routing 1
Declares that in order for a customer to move between queues, the server component at the first queue must perform a routing action, leading to a queue which has a rank with at least one space. R is the rank closest to the front with any spaces. By filling up from the front we ensure that any ranks behind R (denoted with the shorthand Xs) are completely empty. n is the largest such n to describe the blank spaces in $Q : B$. Routing actions may not appear in the inter-queue cooperation set, L_Q. The definition of *rate* is explained below.

$$\frac{S_A \xrightarrow{(\alpha,\lambda)} S'_A \quad Q : A \overset{(\alpha,\lambda_q)}{\longmapsto} Q : B}{\begin{array}{r}(Q : A[R_n|\ldots|R_0] \bowtie_{L_A} S_A) \bowtie_{L_Q} (Q : B[Xs|R|\ldots] \bowtie_{L_B} S_B) \xrightarrow{(\alpha,rate)} \\ (Q : A[R'_n|\ldots|R'_0] \bowtie_{L_A} S'_A) \bowtie_{L_Q} (Q : B[Xs|R'|\ldots] \bowtie_{L_B} S_B)\end{array}} \quad \begin{array}{c}\alpha\notin L_A,\ \alpha\notin L_Q,\\ n>0,\ P\in R_0,\\ -\in \mathrm{R}\end{array}$$

where:

$$\begin{array}{rcl} rate &=& \dfrac{\lambda}{r_\alpha(S_A)}\dfrac{\lambda_q}{r'_\alpha(A)}\min(r_\alpha(S_A), r'_\alpha(A))) \\ C_i &=& elem(R_i) \quad \text{for } 0 < i \le n \\ R'_n &=& R_n - \{\!|\, C_n \,|\!\} + \{\!|\, - \,|\!\} \\ R'_i &=& R_i - \{\!|\, C_i \,|\!\} + \{\!|\, C_{i+1} \,|\!\} \quad \text{for } 0 < i < n \\ R'_0 &=& R_0 - \{\!|\, P \,|\!\} + \{\!|\, C_1 \,|\!\} \\ R' &=& R - \{\!|\, - \,|\!\} + \{\!|\, \mathrm{P} \,|\!\} \end{array}$$

Server routing 2
As *Server routing 1* but where the routing action leads to the same queue. Again, R is the foremost rank with an empty slot, with Xs representing any completely empty ranks behind R. The $n+1$ ranks in front of R are denoted $R_i, 0 \le i \le n$. We must take particular care in the case where R is completely empty, as P then actually joins rank R_n. We require that there is an empty slot, even though the customer does not fill it, to avoid unduly prioritising local routing. Otherwise, a customer could be allowed to move to the back of the queue it has just left even while arrivals from other queues are disallowed.

$$\frac{S_A \xrightarrow{(\alpha,\lambda)} S'_A \quad Q : A \overset{(\alpha,\lambda_q)}{\longmapsto} Q : A}{(Q : A[Xs|R|R_n|\ldots|R_0] \bowtie_{L_A} S_A) \xrightarrow{(\alpha,rate)} (Q : A[Xs|R'|R'_n|\ldots|R'_0] \bowtie_{L_A} S'_A)} \quad {\scriptstyle \alpha\notin L_A,\ n>0}$$

where:

$$\begin{array}{rcl} rate &=& \dfrac{\lambda}{r_\alpha(S_A)}\dfrac{\lambda_q}{r'_\alpha(A)}\min(r_\alpha(S_A), r'_\alpha(A))) \\ C_i &=& elem(R_i) \quad \text{for } 0 < i < n \\ C_R &=& \begin{cases} P & \text{if R is completely empty} \\ elem(R) & \text{otherwise} \end{cases} \\ R'_n &=& R_n - \{\!|\, C_n \,|\!\} + \{\!|\, C_R \,|\!\} \\ R'_i &=& R_i - \{\!|\, C_i \,|\!\} + \{\!|\, C_{i+1} \,|\!\} \quad \text{for } 0 < i < n \\ R'_0 &=& R_0 - \{\!|\, P \,|\!\} + \{\!|\, C_1 \,|\!\} \\ R' &=& \begin{cases} R & \text{if R is completely empty} \\ R - \{\!|\, - \,|\!\} + \{\!|\, \mathrm{P} \,|\!\} & \text{otherwise} \end{cases} \end{array}$$

Coop routing 1
For a routing action that is also in the queue's cooperation set, the action must occur in cooperation between the lead customer and the queue's server. Again, routing actions may not appear in the inter-queue cooperation set, L_Q. Otherwise, the only difference with *Server Routing 1* is that P evolves to P', so we place P' in R', rather than P.

$$\frac{P \bowtie_{L_A} S_A \xrightarrow{(\alpha,\lambda)} P' \bowtie_{L_A} S'_A \quad Q : A \overset{(\alpha,\lambda_q)}{\longmapsto} Q : B}{\begin{array}{r}(Q : A[R_n|\ldots, R_0] \bowtie_{L_A} S_A) \bowtie_{L_Q} (Q : B[Xs|R|\ldots] \bowtie_{L_B} S_B) \xrightarrow{(\alpha,rate)} \\ (Q : A[R'_n|\ldots|R'_0] \bowtie_{L_A} S'_A) \bowtie_{L_Q} (Q : B[Xs|R'|\ldots] \bowtie_{L_B} S_B)\end{array}} \quad \begin{array}{c}\alpha\in L_A,\ \alpha\notin L_Q,\\ n>0,\ P\in R_0,\\ -\in \mathrm{R}\end{array}$$

where:

$$\begin{array}{rcl} rate &=& \dfrac{\lambda}{r_\alpha(S_A)}\dfrac{\lambda_q}{r'_\alpha(A)}\min(r_\alpha(S_A), r'_\alpha(A))) \\ C_i &=& elem(R_i) \quad \text{for } 0 < i \le n \\ R'_n &=& R_n - \{\!|\, C_n \,|\!\} + \{\!|\, - \,|\!\} \\ R'_i &=& R_i - \{\!|\, C_i \,|\!\} + \{\!|\, C_{i+1} \,|\!\} \quad \text{for } 0 < i < n \\ R'_0 &=& R_0 - \{\!|\, P \,|\!\} + \{\!|\, C_1 \,|\!\} \\ R' &=& R - \{\!|\, - \,|\!\} + \{\!|\, \mathrm{P'} \,|\!\} \end{array}$$

Coop routing 2
Just as for *Coop routing 1*, but routing to the same queue. The equations are the same as for *Server routing 2*, except the new customer is P', not P.

$$\frac{P \underset{L_A}{\bowtie} S_A \xrightarrow{(\alpha,\lambda)} P' \underset{L_A}{\bowtie} S'_A \quad Q:A \overset{(\alpha,\lambda_q)}{\longmapsto} Q:A}{Q:A[Xs|R|R_n|\ldots|R_0] \underset{L_A}{\bowtie} S_A \xrightarrow{(\alpha,rate)} Q:A[Xs|R'|R'_n|\ldots|R'_0] \underset{L_A}{\bowtie} S'_A} \quad \alpha \in L_A,\ \alpha \in L_Q,\ n > 0$$

where:

$$\begin{aligned}
rate &= \frac{\lambda}{r_\alpha(S_A)} \frac{\lambda_q}{r'_\alpha(A)} \min(r_\alpha(S_A), r'_\alpha(A))) \\
C_i &= elem(R_i) \quad \text{for } 0 < i < n \\
C_R &= \begin{cases} P' & \text{if R is completely empty} \\ elem(R) & \text{otherwise} \end{cases} \\
R'_n &= R_n - \{\!|\, C_n \,|\!\} + \{\!|\, C_R \,|\!\} \\
R'_i &= R_i - \{\!|\, C_i \,|\!\} + \{\!|\, C_{i+1} \,|\!\} \quad \text{for } 0 < i < n \\
R'_0 &= R_0 - \{\!|\, P \,|\!\} + \{\!|\, C_1 \,|\!\} \\
R' &= \begin{cases} R & \text{if R is completely empty} \\ R - \{\!|\, - \,|\!\} + \{\!|\, \mathrm{P}' \,|\!\} & \text{otherwise} \end{cases}
\end{aligned}$$

For the last four rules, *rate* represents the rate of active cooperation between the service component and the queueing network. As in PEPA, it reflects the rate of the slower component in the cooperation.

B. TRANSLATION

We analyse our discrete PEPA Queues by translating the network into a completely flat PEPA model. To do this we create a number of components to track the state of the servers and buffers, together with an instance of each customer component for each of the possible queueing positions in the network.
Let $R_{in}(X)$ be the set of routing actions in to queue X; $R_{out}(X)$ be the routing actions out of queue X and $Ranks(A)$ be the number of non-service ranks at queue A (so the ranks are indexed $0..Ranks(A)$). $RSize(A, i)$ is the number of queueing positions in rank i or queue A. QNames is set containing the names of all the queues in the system.
The population trackers for a queue, A, are as follows. There are two indices, the first is the furthest-forward (i.e. lowest numbered) rank with an empty slot, the second is the number of customers in that rank. Recall that an action name with no associated rate, e.g. $nonempty_{A1}$, denotes an immediate action, as described in Section 2.2. These will cooperate only with other immediate actions, never with timed activities.

$$\begin{aligned}
\mathrm{QA}_{0,0} &\overset{def}{=} \sum_{a \in R_{in}(A)} (a_{A0}, \top).\mathrm{QA}_{0,1} \\
\mathrm{QA}_{i,0} &\overset{def}{=} \sum_{a \in R_{in}(A)} (a_{Ai}, \top).nonempty_{Ai}.\mathrm{QA}_{i,1} \\
&\quad + \sum_{A \overset{(d,r)}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (d_{Bk}, \top).\mathrm{QA}_{i-1,RSize(\mathrm{A},\mathrm{i}-1)} \\
&\quad \text{where } 0 < i < Ranks(A) \\
\mathrm{QA}_{i,1} &\overset{def}{=} \sum_{a \in R_{in}(A)} (a_{Ai}, \top).\mathrm{QA}_{i,2} \\
&\quad + \sum_{A \overset{(d,r)}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (d_{Bk}, \top).empty_{Ai}.\mathrm{QA}_{i-1,0} \\
&\quad \text{where } 0 < i < Ranks(A) \\
\mathrm{QA}_{i,j} &\overset{def}{=} \sum_{a \in R_{in}(A)} (a_{Ai}, \top).\mathrm{QA}_{i,j+1} \\
&\quad + \sum_{A \overset{(d,r)}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (d_{Bk}, \top).\mathrm{QA}_{i,j-1} \\
&\quad \text{where } 0 < i < Ranks(A); 1 < j < RSize(A, i) \\
\mathrm{QA}_{i,RSize(A,i)} &\overset{def}{=} \sum_{a \in R_{in}(A)} (a_{Ai}, \top).\mathrm{QA}_{i+1,0} \\
&\quad + \sum_{A \overset{(d,r)}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (d_{Bk}, \top).\mathrm{QA}_{i,RSize(A,i)-1} \\
&\quad \text{where } 0 < i < Ranks(A) \\
\mathrm{QA}_{Ranks(A),\mathrm{Full}} &\overset{def}{=} \sum_{A \overset{(d,r)}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (d_{Bk}, \top).\mathrm{QA}_{Ranks(A),Full-1} \\
&\quad \text{where } Full = RSize(A, Ranks(A))
\end{aligned}$$

The server components are altered to account for the expanded names for routing actions and to ensure that the local cooperation actions are distinct for each queue. So, for a server component, S, residing at queue A, we make a new component $\mathrm{S_A}$. The local cooperation set at A is L_A. For each action-type, a, that S performs, there are three cases:

1. $a \notin R_{out}(A) \wedge a \notin L_A$
2. $a \notin R_{out}(A) \wedge a \in L_A$
3. $a \in R_{out}(A)$

For case 1, we don't need to modify the action at all. For case 2, we add the queue name to the action name, to ensure this action cannot occur when a customer is at a different queue. For case 3, we replace each occurrence of a with a choice of new action-types representing all the destination ranks the action could lead to. We denote these four sets of action names L_1, L_2, L_3 respectively.

$$\begin{aligned}
\mathrm{S_A} &\overset{def}{=} \sum_{\mathrm{S} \xrightarrow{(a,r)} \mathrm{S}' \wedge a \in \mathrm{L}_1} (a, r).\mathrm{S'_A} \\
&\quad + \sum_{\mathrm{S} \xrightarrow{(a,r)} \mathrm{S}' \wedge a \in \mathrm{L}_2} (a_A, r).\mathrm{S'_A} \\
&\quad + \sum_{\mathrm{S} \xrightarrow{(a,r)} \mathrm{S}' \wedge a \in \mathrm{L}_3} \sum_{A \overset{a}{\longmapsto} B} \sum_{k=0}^{Ranks(B)} (a_{Bk}, r).\mathrm{S'_A}
\end{aligned}$$

We repeat the above translation for all the derivative states of S.
For each queue, A, with $0 \leq i \leq Ranks(A)$:

$$\begin{aligned}
\mathrm{Rank_{Ai}} &\overset{def}{=} empty_{Ai}.\mathrm{Rank'_{Ai}} + back_{Ai}.\mathrm{Rank_{Ai}} \\
&\quad + \sum_{c \in R_{out}(A)} (c_A, \top).\mathrm{Rank''_{Ai}} \\
\mathrm{Rank'_{Ai}} &\overset{def}{=} nonempty_{Ai}.\mathrm{Rank_{Ai}} + back_{Ai}.\mathrm{Rank'_{Ai}} \\
&\quad + \sum_{c \in R_{out}(A)} (c_A, \top).\mathrm{Rank'_{Ai}}
\end{aligned}$$

$$\mathrm{Rank}''_{\mathrm{Ai}} \overset{\mathit{def}}{=} \mathit{forward}_{Ai}.\mathrm{Rank}_{\mathrm{Ai}}$$

Thus for each queue, A, with $n = \mathit{Ranks}(A)$ and $L = R_{out}(A)$:

$$\mathrm{Ranks}_{\mathrm{A}} \overset{\mathit{def}}{=} \mathrm{Rank}_{\mathrm{A0}} \underset{L}{\bowtie} \mathrm{Rank}_{\mathrm{A1}} \ldots \underset{L}{\bowtie} \mathrm{Rank}_{\mathrm{An}}$$

For each customer component, P we make a version of that component in each queueing position of each queue. $\mathrm{P_A}0$ represents the customer, P, in service at queue A; $\mathrm{P_A}i$, for $1 \le i \le \mathit{Ranks}(A)$ is customer P waiting in rank i of A. When in service, the translated component has all the behaviour of P, but routing actions lead to the appropriate next queueing rank. While a customer waits, we passively allow all routing actions, and promote one of the customers in the rank using the mechanism described in Section 3.

We partition the actions a customer performs into three sets at each queue, A:

L^A_{C1}: Actions that are not in the local cooperation set and are not outbound routing actions. $a \notin L_A \wedge a \notin R_{out}(A)$.

L^A_{C2}: Actions that are in the local cooperation set, but are not outbound routing actions. $a \in L_A \wedge a \notin R_{out}(A)$.

L^A_{C3}: Actions that are outbound routing actions. $a \in R_{out}(A)$.

The customer at the front gets to actually perform the actions, and change state. For actions in L^A_{C1}, waiting customers also perform the actions in straightforward manner. For actions in L^A_{C2} waiting customers cannot perform them, but must passively witness any of the local cooperation actions for any queue. Actions in L^A_{C3} are the most complicated. Waiting customers must passively witness these actions, then potentially move forward to the rank in front. For this we use the intermediate 'pot' state, which immediately offers both forward and back actions. This is as described in Section 3.

$$\begin{aligned}
\mathrm{P_{A0}} &\overset{\mathit{def}}{=} \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P}' \wedge a \in L^A_{C1}} (a, r).\mathrm{P}'_{\mathrm{A0}} \\
&+ \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P}' \wedge a \in L^A_{C2}} (a_A, r).\mathrm{P}'_{\mathrm{A0}} \\
&+ \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P}' \wedge a \in L^A_{C3} \wedge \mathrm{A} \overset{\mathrm{a}}{\longmapsto} \mathrm{B}} \sum_{k=0}^{\mathit{Ranks}(B)-1} (a_{Bk}, r).\mathrm{P}'_{\mathrm{B}k+1} \\
\mathrm{P_{A}}_i &\overset{\mathit{def}}{=} \sum_{\mathrm{P}\xrightarrow{(a,r)}\mathrm{P}' \wedge a \in L^A_{C1}} (a, r).\mathrm{P}'_{\mathrm{A}i} \\
&\text{where } 1 < i \le \mathit{Ranks}(A) \\
&+ \sum_{\mathrm{X} \in \mathrm{QNames}} \sum_{a \in L_X} (a_X, \top).\mathrm{P}_{\mathrm{A}i} \\
&+ \sum_{\mathrm{P}\xrightarrow{(a,\top)}\mathrm{P}' \wedge a \in L^A_{C3} \wedge \mathrm{A} \overset{\mathrm{a}}{\longmapsto} \mathrm{B}} \sum_{k=0}^{\mathit{Ranks}(B)-1} (a_{Bk}, \top).\mathrm{P}^{\mathrm{pot}}_{\mathrm{A}i} \\
\mathrm{P}^{\mathrm{pot}}_{\mathrm{A}i} &\overset{\mathit{def}}{=} \mathit{back}_{Ai}.\mathrm{P}_{\mathrm{A}i} + \mathit{forward}_{Ai}.\mathrm{P}_{\mathrm{A}i-1}
\end{aligned}$$

Inspecting the queueing system equation of the original model, we place the customers at the correct initial positions. Thus the customers are represented by a cooperation over all of the routing actions, L_r.

$$\begin{aligned}
\mathrm{Customers} &\overset{\mathit{def}}{=} \mathrm{P}_{Xi} \underset{L_r}{\bowtie} \ldots \mathrm{P}'_{Yj} \\
L_r &= \bigcup_{Z \in \mathrm{QNames}} \bigcup_{c \in R_{in}(Z)} \bigcup_{0 \le i \le \mathit{Ranks}(Z)} \{c_{Zi}\}
\end{aligned}$$

Suppose QNames $= \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{Z}\}$, with server components is $\mathrm{S_A}, \mathrm{S_B}, \ldots, \mathrm{S_Z}$ and the intra-queue cooperation set is L_{Qnet}. The foremost rank with spaces in in queue X is x_X, and there are y_X customers waiting in that rank. The servers, population trackers and queue rank managers are defined as follows:

$$\begin{aligned}
\mathrm{Servers} &\overset{\mathit{def}}{=} \mathrm{S_A} \underset{L_{Qnet}}{\bowtie} \mathrm{S_B} \ldots \underset{L_{Qnet}}{\bowtie} \mathrm{S_Z} \\
\mathrm{Trackers} &\overset{\mathit{def}}{=} \mathrm{QA}_{x_A, y_A} \underset{L_r}{\bowtie} \mathrm{QB}_{x_B, y_B} \underset{L_r}{\bowtie} \ldots \underset{L_r}{\bowtie} \mathrm{QZ}_{x_Z, y_Z} \underset{L_r}{\bowtie} \\
\mathrm{Ranks} &\overset{\mathit{def}}{=} \mathrm{Ranks_A} \underset{L_r}{\bowtie} \mathrm{Ranks_B} \ldots \underset{L_r}{\bowtie} \mathrm{Ranks_Z}
\end{aligned}$$

The trackers and ranks communicate with one another over all the routing actions, and also the signals for *empty* and *nonempty* at each rank. We call this set L_{track}:

$$\begin{aligned}
L_{track} &= L_r \cup \bigcup_{X \in \mathrm{QNames}} \bigcup_{i=0}^{\mathit{Ranks}(X)} \{\mathit{empty}_{Xi}, \mathit{nonempty}_{Xi}\} \\
\mathrm{Queues} &\overset{\mathit{def}}{=} \mathrm{Trackers} \underset{L_{track}}{\bowtie} \mathrm{Ranks}
\end{aligned}$$

The servers, queues and customers cooperate over all the routing actions, with the union of all the cooperation sets at each queue. Recall that these local cooperation actions were decorated with the name of the queue where they operate.

$$L_{sys} = L_r \cup \bigcup_{X \in \mathrm{QNames}} \bigcup_{a \in L_X} \{a_X\}$$

The whole system is then given by:

$$\mathit{Sys} \overset{\mathit{def}}{=} \mathrm{Customers} \underset{L_{sys}}{\bowtie} (\mathrm{Servers} \underset{L_{sys}}{\bowtie} \mathrm{Queues})$$