Performance Modelling with Process Algebras

Jane Hillston
Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland
Structure of this talk

- Stochastic Process Algebra (PEPA)

- Benefits of Compositionality:
  - Model Construction
  - Model Manipulation
  - Model Solution

- Quasi-reversibility

- Term-rewriting to uncover quasi-reversible PEPA models

- Conclusions
Performance Modelling using CTMC

Model Construction

- describing the system using a high level modelling formalism
- generating the underlying CTMC

Model Manipulation

- model simplification
- model aggregation

Model Solution

- solving the CTMC to find steady state probability distribution
- deriving performance measures
Stochastic Process Algebra

Attractive features of process algebras + Quantification

- Compositionality
- Formal definition
- Parsimony

Models are constructed from components who engage in activities with an associated stochastic delay

\[(\alpha, F).P\]

SPA MODEL \(\xrightarrow{\text{SOS rules}}\) LABELLED MULTI-TRANSITION SYSTEM \(\xrightarrow{\text{state transition diagram}}\) STOCHASTIC PROCESS

Jane Hillston
Performance Modelling with Process Algebras
6th Nov 2002
Markovian Process Algebra

When we are interested in a Continuous Time Markov Chain (CTMC) as the underlying stochastic process we must restrict all delays to be governed by a negative exponential distribution:

\[(\alpha, r).P\]

Performance Evaluation Process Algebra (PEPA)

In PEPA probabilistic branching is introduced by the race policy: there are no explicitly probabilistic combinators.
PEPA

\[ S ::= (\alpha, r).S \mid S + S \mid A \]
\[ P ::= S \mid P \otimes_L P \mid P/L \]

PREFIX: \((\alpha, r).S\) designated first action

CHOICE: \(S + S\) competition between components (race policy)

CONSTANT: \(A \overset{\text{def}}{=} S\) assigning names to components

COOPERATION: \(P \otimes_L P\) \(\alpha \notin L\) concurrent activity (individual actions)
\(\alpha \in L\) cooperative activity (shared actions)

HIDING: \(P/L\) abstraction \(\alpha \in L \Rightarrow \alpha \to \tau\)

PEPA MODEL \(\xrightarrow{\text{SOS rules}}\) LABELLED MULTI-TRANSITION SYSTEM \(\xrightarrow{\text{state transition diagram}}\) CONTINUOUS TIME MARKOV CHAIN \(Q\)
Example

\[
\begin{align*}
    Browser & \stackrel{\text{def}}{=} (\text{display}, p_1 \lambda). (\text{cache}, m). Browser + \\
    & \quad (\text{display}, p_2 \lambda). (\text{get}, g). (\text{download}, \top). (\text{rel}, r). Browser \\
    Server & \stackrel{\text{def}}{=} (\text{get}, \top). (\text{download}, \mu). (\text{rel}, \top). Server \\
    WEB & \stackrel{\text{def}}{=} ((Browser \parallel Browser)/\{\text{cache}\}) \boxtimes L Server \\
    \text{where } L & = \{\text{get}, \text{download}, \text{rel}\}
\end{align*}
\]
Benefits of Formality

Model Construction: Compositionality leads to

- ease of construction
- reusable submodels
- easy to understand models

Model Manipulation: Equivalence relations lead to

- term rewriting/state space reduction techniques
- aggregation techniques based on lumpability

Model Solution: Characterisation of “ideal” forms: identifying classes of models susceptible to efficient solution based on compositional structure.
Model Manipulation

Model simplification: replace one model by another which is more attractive from a solution point of view, e.g. smaller state space, special class of model, etc. Here we look for a model-model equivalence to justify the substitution.

Model aggregation: establish a partition of the state space of a model, and replace each set of states by one macro-state, i.e. take a different stochastic representation of the same model. Here we look for a state-state equivalence to define a sensible partition.
Model Manipulation in PEPA

Various equivalence relations have been defined for PEPA and their suitability for model manipulation established.

**Simplification:** *weak isomorphism* allows intermediate states in a sequence of $\tau$ actions to be eliminated and thus allows the state space to be reduced.

**Aggregation:** *strong equivalence* detects repeated patterns of behaviour and allows one representative of each such pattern to be chosen.

Since these equivalence relation are congruences the techniques are complementary to the compositional structure

$\Rightarrow$ submodels may be simplified or aggregated in isolation and then replaced within the original model expression
Decomposed Solution of MPA models

Partition the model $M$ into $n$ submodels $m_1, m_2, \ldots, m_n$. Treating each submodel as a single state form the aggregated model $A$.

In isolation, find the steady state distribution $\pi$ for each of the submodels $m_i$ and for the aggregated model $A$.

Apply disaggregation function $f$ to the $\pi(m_i)$ and $\pi(A)$ to find the steady state distribution of the original model $M$ (approximately).

To what extent do the submodels relate to existing components within SPA models.....?
MPA models with Product Form Distribution

Partition the model $M$ into $n$ statistically independent submodels $m_1, m_2, \ldots, m_n$

In isolation, find the steady state distribution $\pi$ for each of the submodels $m_i$

Form the steady state distribution of $M$ as the product of the solutions for each submodel $m_i$ and a normalising constant

When do SPA components behave as if they were statistically independent.....?
Product Form MPA Models

\[ P \equiv S_1 \parallel S_2 \]

Add direct interaction of a restricted type between components with a particular structure

\[ P \equiv S_1 \bowtie L S_2 \]

\( S_1, S_2 \) and \( L \) all restricted

- Quasi-reversibility
- Reversibility
- Routing process approach

Add indirect interaction via another component which has a particular structure and type of interaction

\[ P \equiv (S_1 \parallel S_2) \bowtie L R \]

\( L \) and \( R \) restricted (wrt \( S_1 \) and \( S_2 \))

- "Boucherie" resource contention
- Queueing discipline models
- Quasi-separability
Quasi-Reversibility (QR)

Quasi-reversibility: the conditions under which the stations of a queueing network behave as if they were independent
Recognising QR Product Form in PEPA

- define a notion of action $\alpha$ and complementary action $-\alpha$

- a QR component is a component which enables actions $\alpha_1, \ldots, \alpha_n$ at constant rates in all derivatives and each $\alpha_i$-derivative enables a complementary actions $-\alpha_i$

- in the component $P \boxdot^L Q$, $\alpha \in L$ is a channel from $P$ to $Q$ if $\alpha$ is an action of $Q$ and a complementary action of $P$

- $P \boxdot^L Q$ is termed a flow cooperation if every $\alpha \in L$, which is an action of $P$ or $Q$, forms a channel
Theorem (Open Interactions)

In steady state, an (open) flow cooperation of QR components has the following properties:

1. the underlying Markov process is quasi-reversible
2. the marginal states of the individual components are independent
3. for each component the steady state distribution is as it would be if the component were in isolation
4. in time reversal the model is also a flow cooperation of QR components

Consequences

"Traffic equations" can be used to determine the influence of the rest of the model on each component (input rates). Then each component may be solved separately and the results combined to obtain the complete steady state distribution.
Characterising QR PEPA models

PEPA models with quasi-reversible MP

Quasi-reversible PEPA QR processes

Markov processes

Quasi-reversible Markov processes
Expanding the characterisation

PEPA models with quasi-reversible MP

PEPA QR processes

equivalent, alternative PEPA models

Quasi-reversible Markov processes
Transformation Approach

- The objective is to take a model (which does give rise to a quasi-reversible Markov process but which is not evidently a QR PEPA model) and systematically transform it using the equivalence relation strong equivalence.

- Previous general attempts at defining transformation systems have not been successful.

- Here we work within a more constrained system since we assume that all the models considered are QR in some representation, e.g. all cooperations are channels etc.

- If the considered model is not QR this will become apparent because applying the rules will lead to inconsistent conclusions about channels and/or states.
State vectors

- Given that we often want to separate components from a sequential form it is useful to work in terms of vectors of states.
- In the case we know that there are two or more components the entries in the vector represent the local states of the individual components.
- In the case where we are presented with a single component which we suspect of being compound we use the vector in a purely syntactic way.

For a model expression, we define the vector form inductively over the structure of the expression: let $P$ and $Q$ be expressions and $C$ be a constant denoting a sequential component.

1. $\text{vf}(P \parallel_L Q) = (\text{vf}(P), \text{vf}(Q))_L$
2. $\text{vf}(C) = C$
PEPA operational semantics for state vectors (1)

Prefix

\[(\alpha, r) \cdot (P, Q)_L \xrightarrow{(\alpha, r)} (P, Q)_L\]

Choice

\[
\begin{align*}
(P, Q)_L & \xrightarrow{(\alpha, r)} (P', Q')_L \\
(P, Q)_L + (R, S)_K & \xrightarrow{(\alpha, r)} (P', Q')_L \\
(R, S)_K & \xrightarrow{(\alpha, r)} (R', S')_K \\
(P, Q)_L + (R, S)_K & \xrightarrow{(\alpha, r)} (R', S')_K
\end{align*}
\]
Cooperation

\[
\begin{align*}
(P, Q)_L \xrightarrow{(\alpha, r)} (P', Q')_L \\
((P, Q)_L, (R, S)_K)_M \xrightarrow{(\alpha, r)} ((P', Q')_L, (R, S)_K)_M
\end{align*}
\]

\[
\begin{align*}
(R, S)_K \xrightarrow{(\alpha, r)} (R', S')_K \\
((P, Q)_L, (R, S)_K)_M \xrightarrow{(\alpha, r)} ((P, Q)_L, (R', S')_K)_M
\end{align*}
\]

\[
\begin{align*}
(P, Q)_L \xrightarrow{(\alpha, r_1)} (P', Q')_L \\
(R, S)_K \xrightarrow{(\alpha, r_2)} (R', S')_K
\end{align*}
\]

\[
\begin{align*}
((P, Q)_L, (R, S)_K)_M \xrightarrow{(\alpha, s)} ((P', Q')_L, (R', S')_K)_M
\end{align*}
\]

where \( s = \frac{r_1}{r_{\alpha}(P \boxtimes L Q)} \cdot \frac{r_2}{r_{\alpha}(R \boxtimes K S)} \cdot \min(r_{\alpha}(P \boxtimes L Q), r_{\alpha}(R \boxtimes K S)) \)
Rule 1: State Vector Prefix

Assume $P$ and $Q$ are a pair of flow cooperating components, in the cooperation $P \congset{L} Q$, with derivative state vector $(P', Q')_L$.

**Rule 1.1** If $\alpha$ is a channel, such that $\alpha \in \vec{A}(P) \cup \vec{A}(Q)$ and $\alpha \in L$, then

$((\alpha, r).P', (\alpha, r).Q')_L = (\alpha, r).(P', Q')_L$

**Rule 1.2** If $\alpha \notin L$, then

$((\alpha, r).P', Q)_L = (\alpha, r).(P', Q)_L$

**Rule 1.3** If $\alpha \notin L$, then

$(P, (\alpha, r).Q')_L = (\alpha, r).(P, Q')_L$

This rule dictates when a state vector may be prefixed by an activity.
Rule 2: State Vector Choice

Assume $P$ and $Q$ are a pair of flow cooperating components, in the cooperation $P \rhd_L Q$, with derivative state vectors $(P', Q')_L$ and $(P'', Q'')_L$. If $\alpha$ and $\beta$ are channels such that $\{\alpha, \beta\} \subset \vec{A}(P) \cup \vec{A}(Q)$ and $\{\alpha, \beta\} \subset L$, then

$$(\alpha, r).P' + (\beta, s).P'' + (\beta, s).Q'' + (\alpha, r).Q')_L = (\alpha, r).(P', Q')_L + (\beta, s).(P'', Q'')_L$$

This rule shows how choice between state vectors arises, when there are two channels between the components.
Rule 3: Activity Blocking

Assume $P$ and $Q$ are a pair of flow cooperating components, in the cooperation $P \bowtie L Q$, with derivative state vector $(P', Q)_L$, $Q \equiv (\beta, s).Q'$. If $\beta$ is a channel such that $\beta \in \vec{A}(Q)$ and $\beta \in L$, then

\[
((\alpha, r).P', (\beta, s).Q')_L = (\alpha, r).(P', Q)_L
\]

where $\alpha \neq \beta$ and $\alpha \notin L$

This rule defines how progress may be blocked by the cooperation set. The activity $(\beta, s)$ affects only the state of the $Q$ component (left hand side). Behaviourally this is not possible as $(\beta, s)$ is a channel: it must affect the state of both cooperating components. Since $(\beta, s)$ does not appear in both tuple elements the transformation rule discards its effect on the combined state as invalid.
**Rule 4: Choice and Cooperation**

Assume $P$ and $Q$ are a pair of flow cooperating components, in the cooperation $P \parallel_L Q$, with derivative state vectors $(P', Q)_L$ and $(P'', Q'')_L$, $Q \equiv (\beta, s).Q''$. If $\beta$ is a channel such that $\beta \in \tilde{A}(P) \cup \tilde{A}(Q)$ and $\beta \in L$, then

$$(\alpha, r).P' + (\beta, s).P'', (\beta, s).Q'')_L = (\alpha, r).(P', Q)_L + (\beta, s).(P'', Q'')_L$$

where $\alpha \neq \beta$ and $\alpha \notin L$.

Here $(\beta, s)$ appears in both tuple elements hence the transformation rule accounts for its effect on the combined state. Note that the term $(\beta, s). (P'', Q')$ describes a flow transition whose input-output nature is reflected in the fact that both the state of the source $P$ and that of the destination $Q$ must change.
Rule 5: Choice, Cooperation and Blocking

Assume $P$ and $Q$ are a pair of flow cooperating components, in the cooperation $P \Join^L Q$, with derivative state vectors $(P', Q)_L$ and $(P'', Q'')_L$, $Q \equiv (\gamma, t).Q'$. If $\beta$ and $\gamma$ are channels such that $\{\beta, \gamma\} \subset L$, $\beta \in \bar{A}(P) \cup \bar{A}(Q)$ and $\gamma \in \bar{A}(Q)$ then

$$(\alpha, r).P' + (\beta, s).P'' + (\beta, s).Q'' + (\gamma, t).Q')_L = (\alpha, r).(P', Q)_L + (\beta, s).(P'', Q'')_L$$

where $\alpha \neq \beta \neq \gamma$ and $\alpha \notin L$.

This rule is a generalisation of Rule 3 and Rule 4.
Implementation

- Having now developed a set of formally defined transformations on PEPA models we have implemented these in a term rewriting system which checks the correct application of each of these rules.

- Our system is called the **PEPA Term Kit** and we have used it to check several example derivations.

- The PEPA Term Kit is implemented in the strongly-typed functional language **Standard ML** which provides a metalanguage for defining other formal languages, whether they are programming languages or modelling languages as is the case for PEPA here.
Using the PEPA Term Kit

- The PEPA Term Kit presents users with an *interactive top-level loop* which allows them to enter model definitions, fold and unfold definitions and apply rewriting rules.

- After experimenting with the interactive application of the rules to strengthen understanding, the user can define *tactics* as in the style of Edinburgh LCF.

- These tactics can be used to apply *combinations of rules* in a single step.

- The necessary uses of rules and tactics to prove a desired goal can be stored in a *proof script* which can be replayed as needed.
Conclusions

• Stochastic process algebras have many attractive features for performance modelling and are gaining acceptance within the performance community.

• Our experience with PEPA has been that the combination of a well-defined formal semantics for the language and (modest, but usable) tool support for the language enabled us and others to use it effectively.

• The compositional structure of the process algebra can be exploited to find efficient solution techniques for some PEPA models.

• However the usefulness of characterisations of ideal forms for solution will always be limited and further work is needed on transformation systems and approximation techniques based on these characterisations.