UNE IMP EHE CWI ★QMIPS★ INR UER ZAR UTO

Quantitative Modeling In Parallel Systems

Response Time Approximation for Decision-Free TIPP Specifications

Vassilis Mertsiotakis[1] and Manuel Silva[2]

[1] Universität Erlangen-Nürnberg, IMMD VII, Martensstr. 3, 91058 Erlangen, Germany

[2] Universidad de Zaragoza, Dpto. di Informatica, Maria de Luna 3, 50015 Zaragoza, Spain

D W2.T2.15

21. September 1995

Distribution Level: Public

Approved by: Ulrich Herzog

Abstract This report contains characterizations of Stochastic Process Algebra models that correspond to 1-safe Marked Graphs. Two different viewpoints to this problem are discussed, a *station oriented* point of view and a *event oriented* point of view. In the former Petri Nets are considered as parallel processes where each place corresponds to a sequential component. In the event oriented point of view the flow of tokens is modelled instead. Based on the latter one we describe conditions when and how SISO-cuts can be performed and show how to construct aggregated systems. The throughput of this class of models is computed via an iterative response time approximation algorithm.

The QMIPS Project: The QMIPS project is a collaborative research project supported by the CEC as ESPRIT-BRA project no 7269. It is being carried out by the following organisations: CWI (Amsterdam), EHEI (University of Paris V), Imperial College (London), INRIA (Sophia-Antipolis), University of Erlangen, University of Newcastle, University of Torino and University of Zaragoza.

Introduction to deliverable D15

The previous deliverable D14 as well as D12 showed a number of efficient solution methods for certain classes of stochastic Petri nets. The present deliverable is a joint report from UER and ZAR and represents a first attempt to build a bridge between these efficient solution techniques and the world of Stochastic Process Algebras (SPA).

We introduce a class of SPA-models called Decision Free Processes (DFP). These models correspond to 1-safe Marked Graph Petri nets. In this report we present a detailed discussion on various oportunities to represent such systems in SPA terms and show syntactic conditions under which a process fulfills this decision free property. For this class of processes we show how to obtain an approximate solution for the system throughput by an iterative decomposition and response time approximation algorithm.

The formulation of the approximation algorithm was perhaps the easiest part of this work. However, the examination of conditions that ensure the decision freeness as well as the formalization of decomposition schemes was the trickiest part. That is the reason why we restricted ourselfes for the time being to a very simple class of models. It should be mentioned, however, that the expressiveness of DFPs is still higher than that of task graph models, since task graphs are a subclass of 1-safe Marked Graphs.

1 Notation and definitions

We assume a fixed set of action names $Act := Com \cup \{\tau\}$, where τ is a distinguished symbol for internal, invisible actions and Com is the set of visible activities (*communication* actions). Each action can either be exponentially distributed or passive. Exponentially distributed actions — denoted by the action's name and the rate (a, λ) — happen instantaneously after a duration that is exponentially distributed with rate λ. Passive actions (usually denoted by $(a, 1)$) describe *receptive* behaviour, i.e. the behaviour of a component which is waiting for a partner before completing the action.

1.1 Syntax

The syntax of our Stochastic Process Algebra TIPP (*timed processes and performability evaluation*) is defined as follows [HR94]:

Definition 1.1 *The set $\mathcal{L}$ of valid system descriptions of TIPP is given by the following grammar, where $a \in Act$, $\lambda \in \mathbb{R}^+$; $S \subseteq Com$, $X \in Var$, Var is a set of process variables.*

$$P ::= Stop \ | \ (a, \lambda).P \ | \ P + P \ | \ P\|_S P \ | \ P \backslash S \ | \ rec\, X : P \ | \ X$$

In addition to the recursion operator *rec*, we will use *defining equations* in order to model recursion. We assume that there is a countable set of *constants*, whose meaning is given by a defining equation such as $A := P$, i.e. the constant A has the behaviour of the agent P.

Definition 1.2 *$Sort(P)$ denotes the set of action names corresponding to the actions which process P may engage in:*

$$\begin{array}{lll@{\qquad}lll} Sort(0) & = & \emptyset & Sort((a,\lambda).P) & = & \{a\} \cup Sort(P) \\ Sort(P \,\|_S Q) & = & Sort(P) \cup Sort(Q) & Sort(rec\ X : P) & = & Sort(P) \\ Sort(P\ +\ Q) & = & Sort(P) \cup Sort(Q) & Sort(P \backslash S) & = & Sort(P) \setminus S \end{array}$$

This is termed the action sort *of P.*

Note that the set of actions which P will actually engage in is a subset of $Sort(P)$ because the synchronization may disable the execution of some actions.

1.2 Semantics

The semantics of TIPP are given by a set of inference rules: they are presented here in Figure 1 and for a detailed introduction cf. [HR94]. Note that we don't provide separate semantic rules for processes with passive actions because they can be treated in the same way as active ones. Hence, it is a matter of interpretation whether an action is active or passive. For example, an action $(a, 1)$ might also describe an active action with rate 1, while an action $(a, 0.5)$ could describe a passive action with a weight 0.5 that will be multiplied with the rate of the active counterpart.

$$\langle . \rangle \quad \frac{}{(a,\lambda).P \xrightarrow{a,\lambda,\varepsilon} P} \qquad \langle \|_l \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P\|_S Q \xrightarrow{a,\lambda,\|_l.w} P'\|_S Q} \quad (a \notin S)$$

$$\langle +_l \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P+Q \xrightarrow{a,\lambda,+_l.w} P'} \qquad \langle \|_r \rangle \quad \frac{Q \xrightarrow{a,\lambda,w} Q'}{P\|_S Q \xrightarrow{a,\lambda,\|_r.w} P\|_S Q'} \quad (a \notin S)$$

$$\langle +_r \rangle \quad \frac{Q \xrightarrow{a,\lambda,w} Q'}{P+Q \xrightarrow{a,\lambda,+_r.w} Q'} \qquad \langle \| \rangle \quad \frac{P \xrightarrow{a,\lambda,v} P' \quad Q \xrightarrow{a,\mu,w} Q'}{P\|_S Q \xrightarrow{a,\lambda\mu,(v,w)} P'\|_S Q'} \quad (a \in S)$$

$$\langle \backslash_{yes} \rangle \quad \frac{P \xrightarrow{a,\lambda,w} P'}{P\backslash a \xrightarrow{\tau,\lambda,w} P'\backslash a} \qquad \langle \backslash_{no} \rangle \quad \frac{P \xrightarrow{b,\lambda,w} P'}{P\backslash a \xrightarrow{b,\lambda,w} P'\backslash a} \quad (a \neq b)$$

$$\langle rec \rangle \quad \frac{P\{(recX : P)/X\} \xrightarrow{a,\lambda,w} P'}{recX\ :\ P \xrightarrow{a,\lambda,w} P'}$$

Figure 1: Operational Semantics of TIPP [HR94]

Definition 1.3 (Reachability Set) *The* reachability set $ds(P)$ *of a given process* P *is defined as:*

$$ds(P) = \{P' \in \mathcal{L} \mid P \rightarrow^* P'\}$$

where $P \rightarrow^* P'$ *denotes that the process state* P' *is derivable by applying a sequence of SOS rules to the process* P.

1.3 Static TIPP models

Because we want the component structure of the model to be static during evolution we only allow recursion over the lower level components, not over parallel combinations of components. This leads us to formally define the syntax of *static* TIPP expressions in terms of *sequential components* Q and *model components* P:

$$\begin{array}{lcl} P & ::= & Q \mid P\|_S P \\ Q & ::= & (\alpha, r).Q \mid Q+Q \mid rec\ X : Q \mid X \mid Stop \end{array}$$

Definition 1.4 *Let* $\mathcal{L}_{seq} \subset \mathcal{L}$ *be the set of all sequential processes. Then, the set of static processes with* n *components is defined as follows [HM95]:*

$$\mathcal{L}^n_{par} = \{P \in \mathcal{L} \mid P \equiv P_1 \|_{S_1} P_2 \|_{S_2} \ldots\ P_m \ \wedge\ P_i \in \mathcal{L}^{k_i}_{par} \ \wedge\ \sum_{i=1}^{m} k_i = n\}$$

for arbitrary $S_i \subseteq Com$ *where* $\mathcal{L}^1_{par} = \mathcal{L}_{seq}$.

It will sometimes be convenient to be able to define whether a given sequential component occurs within a static agent: the partial order, $\prec$, over components, which captures the notion of *being a subcomponent* [HM95]:

Definition 1.5 (Subcomponents)

1. $R \prec P$ *if* $R \in ds(P)$

2. $R \prec P + Q \quad \textit{if } R \prec P \vee R \prec Q$

3. $R \prec P \|_S Q \quad \textit{if } R \prec P \vee R \prec Q$

4. $R \prec A \qquad \textit{if } A := P \wedge R \prec P$

The *interface* of a sequential component within a static model is then defined to be the union of all the cooperation sets whose scope includes the component R.

Definition 1.6 (Interface) *For any sequential component R within a model component C (i.e. $R \prec C$) the* interface *of R within C, denoted $\mathcal{I}(C :: R)$, is the set of action types on which R is required to cooperate:*

1. $\mathcal{I}(R :: R) = \emptyset$

2. $$\mathcal{I}(P \|_L Q :: R) = \begin{cases} \mathcal{I}(P :: R) \cup \mathcal{I}(Q :: R) \cup L & \\ \qquad \textit{if } R \prec P \wedge R \prec Q & \\ \mathcal{I}(P :: R) \cup L & \\ \qquad \textit{if } R \prec P \wedge R \not\prec Q & \\ \mathcal{I}(Q :: R) \cup L & \\ \qquad \textit{if } R \prec Q \wedge R \not\prec P & \\ \emptyset \qquad \textit{otherwise.} & \end{cases}$$

Note that the interface of a component may be larger than the action sort of the component. Sometimes we will be interested in only the subset of the interface over which a component is active; this is termed the *active* interface.

Definition 1.7 (Active Interface) *The* active interface *of a sequential component R within a model C, denoted $\mathcal{I}_A(C :: R)$, is the set of actions within the interface of R which R can engage in:*

$$\mathcal{I}_A(C :: R) = \mathcal{I}(C :: R) \cap Sort(R)$$

Two sequential components within a model can cooperate by synchronizing on some actions or can act independently from each other. The set of actions on which two components interact is defined as:

Definition 1.8 (Common Interface) *For any sequential components Q, R within a model component C (i.e. $Q \prec C \ \wedge R \prec C$) the* common interface *of Q and R within C, denoted $\mathcal{I}(C :: Q, R)$, is the set of action types on which Q is required to cooperate with R:*

1. $\mathcal{I}(S :: Q, R) = \emptyset \quad \textit{if } S \in \mathcal{L}_{seq}$

2. $$\mathcal{I}(P_1 \|_L P_2 :: Q, R) = \begin{cases} \mathcal{I}(P_1 :: Q, R) & \textit{if } Q \not\prec P_2 \wedge R \not\prec P_2 \\ \mathcal{I}(P_2 :: Q, R) & \textit{if } Q \not\prec P_1 \wedge R \not\prec P_1 \\ \mathcal{I}(P_1 :: Q, R) \cup \mathcal{I}(P_2 :: Q, R) \cup L & \textit{otherwise.} \end{cases}$$

As with the interface, the common interface may also contain actions that are bound by other sequential components, as the following simple example demonstrates:

Example:

$$P := (P_a \parallel P_b) \parallel_{\{a\}} (P_a \parallel P_b) \qquad P_a := (a, \lambda).Stop \qquad P_b := (b, \lambda).Stop$$

According to the definition, the common interface of P_a and P_b is

$$\mathcal{I}(P :: P_a, P_b) = \{a\}$$

even though P_b can never participate on action a. That is why we have to build intersection with the action sorts of the affected components:

Definition 1.9 (Common Active Interface)

$$\mathcal{I}_A(C :: Q, R) = \mathcal{I}(C :: Q, R) \cap Sort(Q) \cap Sort(R)$$

2 Canonical representation of QNs/PNs

In order to develop a better understanding of how efficient analysis algorithms known from the queueing networks and Petri nets domain might be adopted to the formalism of stochastic process algebras we need to find proper syntactical restrictions that yield classes of processes which would correspond to the special classes of QNs/PNs that are amenable to efficient solution schemes. Here we collect some basic definitions of what could be a canonical representation of SMs, MGs, or FCNs in stochastic process algebras.

Before we do this, a general discussion about the point of view for a canonical representation is necessary. Since queueing networks and Petri nets are two quite different modelling techniques, we can develop two viewpoints on such a canonical representation. A third method appears to be feasible as a mixture of the first two representation methods:

- **Station oriented representation**

  Considering networks of FIFO queueing systems as a parallel composition of sequential processes modelling FIFO queues, we would first think of this point of view, since this seems to be the most straightforward way to translate queueing networks into SPA models.

- **Event oriented representation**

  This representation would arise automatically from a Petri net perspective. This can be understood as focusing on the customers/tokens of a system instead of the stations. So we would describe a system as the parallel composition of sequential processes, each representing one single customer. However, in order to translate a Petri net model that uses a single server semantics the introduction of immediate actions into the SPA models seems to be necessary, as soon as the net to be translated is not 1-safe.

- **Mixed representation**

  In this notational style one might use the event oriented specification method to describe the system workload which consists of a sequential process for each customer/token. This can be done in almost the same way as in the second representation method. The machine model is described as a network of passive FIFO-stations, as in our station oriented model. This representation method would come out without immediate actions as long as no multiprocessor servers have to be considered. Methodologically, this is only a way to avoid immediate actions by means of marking dependend firing rates. We will not consider this representation method in detail.

2.1 The methodology

One systematic approach to define classes of SPA-models that correspond to certain classes of QN/PN models is the following: Considering the particular classes of QN/PN models we are aiming to represent, we try to find in a first step a translation of these models to SPA models. Dependend on the point of view, it might turn out that the resulting SPA-model would contain too detailed information, a fact that means that we would have implicitly transformed the original net into an unfolded, possibly colored P/T net. In this case we have to find systematically the compact representation of this net and the result should then be equivalent – in an appropriate sense – with the original net (cf. Fig. 2).

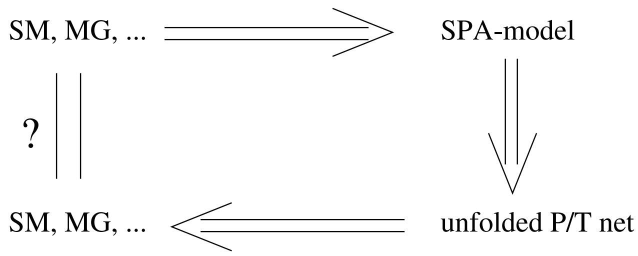


Figure 2: Methodology

In the sequel we follow this methodology for the classes of live and bounded State-Machines (SM), Marked-Graphs (MG), and Free-Choice-Nets (FCN). Unless otherwise stated, we will assume liveness and boundedness. Our aim is to conclude with necessary conditions for a SPA model to yield an equivalent SM, MG, or FCN representation.

2.2 Station oriented representation

As already mentioned, the easiest way to obtain a SPA-model from a SPN/QN is to represent each place/queue by a FIFO-process and superposing all these processes appropriately in parallel. Therefore, we need first the notion of a Birth-Death-Process (BDP).

Definition 2.1 *Be* $BDP_l^k : I\!N \times I\!N \times (Act \times I\!R)^{k+l} \rightarrow \mathcal{L}_{seq}$ *and*

$$a = ((a_1, w_1), (a_2, w_2), \ldots, (a_k, w_k)) \qquad d = ((d_1, \lambda_1), (d_2, \lambda_2), \ldots, (d_l, \lambda_l))$$

Then, we define the function BDP^k_l *as follows:* $BDP^k_l(n,m,a,d) \equiv Q_n$ *where*

$$Q_n := \begin{cases} \sum_{i=1}^{k}(a_i, w_i).Q_1 & \text{if } n = 0 \\ \sum_{i=1}^{k}(a_i, w_i).Q_{n+1} \;+\; \sum_{i=1}^{l}(d_i, \lambda_i).Q_{n-1} & \text{if } 0 < n < m \\ \sum_{i=1}^{l}(d_i, \lambda_i).Q_{m-1} & \text{if } n = m \end{cases}$$

Informally, the function BDP builds for a given set of arrival actions and departure actions a sequential process modelling a FIFO station with the specified buffer size accepting and delivering customers via the actions specified in the vectors a and d respectively.

Suppose the net $(\mathcal{N}, M_0)$ to be translated has n places $P_1, \ldots, P_n$. Then, the corresponding SPA representation of this net is given by:

$$P \equiv P_1 \parallel_{S_1} (P_2 \parallel_{S_2} (\ldots P_{n-1} \parallel_{S_{n-1}} P_n) \ldots))$$

where $P_i = BDP^k_l(n,m,a,d)$ and

$$\begin{aligned} k &= |\bullet P_i| \\ l &= |P_i \bullet| \\ n &= M_0(P_i) \\ m &= M_{max}(P_i) \quad \text{(maximum number of tokens)} \end{aligned}$$

The action names and rates a_i, d_i, w_i, λ_i are set according to the predecessor and successor transitions of P_i. The synchronization sets S_i are determined as follows:

$$\begin{aligned} S_{n-1} &= Sort(P_{n-1}) \cap Sort(P_n) \\ S_i &= Sort(P_i) \cap S_{i+1} \qquad (1 \leq i \leq n-2) \end{aligned}$$

It should be noted that the resulting SPA-model does not model exactly the same behaviour as a SPN, since we have replaced the RANDOM strategy that is inherent in every PN by a FIFO scheduling discipline.

2.3 Event oriented representation

In this subsection we try to present the event oriented point of view on the class of State Machines. These models correspond more or less to Gordon-Newell queueing networks. However, there is a small difference in the queueing discipline. Since in classical Petri nets there is no distinction between tokens waiting in one place, the scheduling strategy of a SM is strictly speaking not FIFO, but, dependent on the semantics of the net either IS or RANDOM.

2.3.1 1-safe nets

These models can very easily be represented using SPA, namely by representing each customer by a single sequential component that evolves through different states, each representing a station/place in the net. The states such a process evolves through would correspond to the places of a P-semiflow (*or P-invariant ???*). Consequently, the coverage by P-semiflows is one important condition to be met. Fortunately, for the class of live and bounded SM, MG, and FCN we are interested in, this condition is always fulfilled.

Example:

For the example nets illustrated in Fig. 3 equivalent SPA-models could be described in the following way:

- **SM (Fig. 3a)**

$$P_1 := (a, \lambda_a).(b, \lambda_b).P_1$$

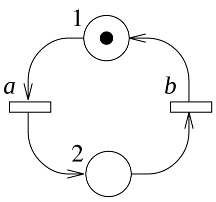
- **MG (Fig. 3b)**

$$\begin{aligned} P &:= P_1 \parallel_{\{b\}} P_2 \\ P_1 &:= (a, \lambda_a).(b, \lambda_b).P_1 \\ P_2 &:= (b, 1).(c, \lambda_c).P_2 \end{aligned}$$

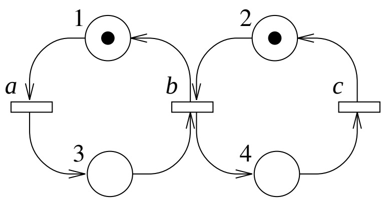
- **FCN (Fig. 3c)**

$$\begin{aligned} P &:= P_1 \parallel_{\{d\}} P_2 \\ P_1 &:= (a, \lambda_a).(c, \lambda_b).P_1 \; + \; (b, \lambda_b).(d, \lambda_d).P_1 \\ P_2 &:= (d, 1).(e, \lambda_e).P_2 \end{aligned}$$

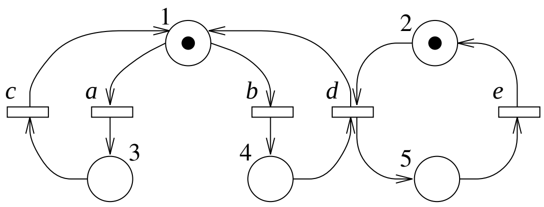
The translation of SMs is the most straightforward. SMs are covered by one P-semiflow, so the translation task is just finding an SPA-description of the entire FSM represented by the net. Translation of MGs and FCNs works similar. Here we have typically more than one P-semiflow, but each of them corresponds to a subnet that is a SM (in the case of MG these subnets are pure cycles). Superposition of all 'P-semiflows' in parallel synchronizing on the actions corresponding to the joint transitions yields an SPA-description. In general, a MG/FCN is not covered uniquely by P-semiflows, so there will be many different ways to obtain a SPA description for a given MG/FCN.



(a) State Machine



(b) Marked Graph



(c) Free Choice Net

Figure 3:

2.3.2 k-safe nets

Dealing with k-safe nets becomes a tough problem. The naive approach would be to just superpose an additional sequential process in parallel for every customer in the system. The example from Fig. 3a could then look like

$$System := (P_1 \,\|\, P_1) \,\|_{\{b\}} P_2$$

for an initial marking $M_0(P_1) = 2$ and $M_0(P_2) = 1$. This, however, would turn out to model a system with infinitely many servers. This is not a problem as long as the net to be translated has an infinite server semantics. Nets with single server semantics use a RANDOM preselection policy to choose a token from each input place of an enabled transition. To implement such a preselection policy with SPA we need a notion of immediate actions. With this extension it would be easy to describe k-bounded nets with single server semantics. We would have to add a *resource process* for each transition in the system. Additionally, every action has to be prefixed by an immediate action that tries to occupy a free resource. In our example this would look as follows:

$$\begin{array}{rcl}
System & := & ((P_1 \,\|\, P_1) \,\|_{\{\beta,b\}} P_2) \,\|_{\{\alpha,a,\beta,b\}} (R_a \,\|\, R_b \,\|\, R_c) \\
P_1 & := & (\alpha,\infty).(a,\lambda_a).(\beta,\infty).(b,\lambda_b).P_1 \\
P_2 & := & (\beta,\infty).(b,\lambda_b).(\gamma,\infty).(c,\lambda_c).P_2 \\
R_a & := & (\alpha,1).(a,1).R_a \\
R_b & := & (\beta,1).(b,1).R_b \\
R_c & := & (\gamma,1).(c,1).R_c
\end{array}$$

A more elegant solution to this problem would be to provide a special parallel operator that implements a preselection policy. These two solutions have been discussed intensively in [Her93]. A special *mapping operator* has been developed, that enables only one pair of synchronizing actions using a random strategy. This is realized by adding immediate actions in the semantic model. The user doesn't have to care about that.

3 Decomposition and aggregation of Decision Free Processes

Based on the experience we made on translating the way back, i.e. representing SPNs by SPA-models, we are now ready to define classes of processes that correspond to the interesting class of Petri nets for which highly efficient solution techniques exist. We start our studies with a very simple class of processes, Decision Free Processes (DFP), that correspond to 1-safe Marked Graphs (MG). Although this is a restricted class of models, it should be mentioned that Stochastic Graphs and PERT-networks represent already a subclass of 1-safe MG.

For this class of models we will show how to analyze them efficienty by techniques known from the queueing networks or Petri net context. Basically, there are two possible ways to do this. One possible approach would be to employ a SPN-semantics for SPA ([Rib94]) and subsequently leave the job to existing SPN-tools. The other possibility is to adopt the techniques to the SPA world. We find the latter approach more appealing mainly for two reasons:

1. Retain the oportunity ot exploit the structural information stored in the SPA-model. In particular, this means that in the SPA-model the user has specified already one possible covering by P-semiflows, so a possibly expensive search for a covering is not necessary.
2. In the more general case of k-bounded MGs the SPN-semantics would flaten the model to a 1-safe net that would loose its structural properties.

3.1 Decision Free Processes

Marked Graphs are Petri nets with the property that every place has only one input transition and one output transition. We introduce a class of processes that correspond to 1-safe Marked Graphs This means that it should always be possible to find a MG whose state space is isomorphic to the state space of the SPA model.

Definition 3.1 *A parallel process* $P \in \mathcal{L}^n_{par}$ *consisting of* n *sequential components* $P_1, \ldots, P_n$ *is a* DFP*, iff:*

1. $(\forall i)(P_i \in \mathcal{L}^{\setminus +}_{seq})$ *no choice*
2. $(\forall i)(\forall p, q \in ds(P_i))(p \xrightarrow{a,\cdot} \wedge q \xrightarrow{a,\cdot} \Rightarrow p \equiv q)$
3. $(\forall i < j < k)(\mathcal{I}_A(P :: P_i, P_j) \cap \mathcal{I}_A(P :: P_j, P_k) \subseteq \mathcal{I}_A(P :: P_i, P_k))$

Condition 1 of the above definition means that no choice is allowed in the model, since this would lead to a place with two outgoing arcs in an underlying PN. The other two conditions prevent the implicit modelling of choice situations. Fig. 4 shows a net that corresponds to a SPA-model that violates condition 2.

$$
\begin{aligned}
System &:= P_1 \parallel_{\{a\}} P_2 \\
P_1 &:= (b,\lambda_b).(a,\lambda_a).(a,\lambda_a).P_1 \\
P_2 &:= (c,\lambda_c).(a,1).P_2
\end{aligned}
$$

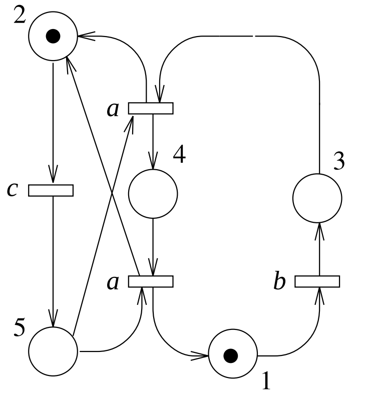


Figure 4: Violation of condition 2

Since the net representing P_1 has two transitions named a, the other subnet has to synchronize with both transitions and this yields a place with 2 outgoing arcs (place 5) as well as a place with 2 incoming arcs (place 2).

Fig. 5 shows that a similar situation occurs if condition 3 is not fulfilled. The situation here is somewhat dual to the last case, where it was forbidden to use the same name for a transition twice within a cycle. Here, we see that if a transition occurs more than once in different cycles, we introduce implicitly a choice as soon as we want to synchronize over such a transition.

$$
\begin{aligned}
System &:= (P_1 \parallel P_2) \parallel_{\{a\}} P_3 \\
P_1 &:= (a,\lambda_a).P_1 \\
P_2 &:= (a,1).P_2 \\
P_3 &:= (a,1).P_3
\end{aligned}
$$

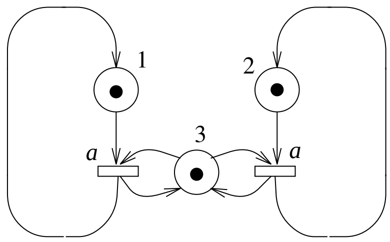


Figure 5: Violation of condition 3

3.2 A motivating example

Before we introduce a formal definition of Single Input Single Output cuts (SISO) for DFPs, let us consider how a SISO cut might be performed on a simple example and how the aggregated systems can be obtained. Consider the SPA-model together with its corresponding SPN representation shown in Fig. 6.

$$
\begin{aligned}
System &:= (P_1 \parallel_{\{a\}} P_2) \parallel_{\{c\}} P_3 \\
P_1 &:= (a, \lambda_a).(b, \lambda_b).P_1 \\
P_2 &:= (a, 1).(c, \lambda_c).P_2 \\
P_3 &:= (c, 1).(d, \lambda_d).P_3
\end{aligned}
$$

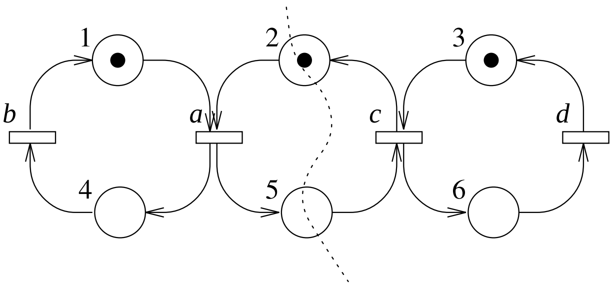


Figure 6: Example

As we can see clearly, the best way to decompose this system is to cut through the places 2 and 5. According to [JSS92] this represents a SISO cut and the aggregated nets resulting from this cut are shown in Fig. 7.

$$
\begin{aligned}
AP_1 &:= P_1 \parallel_{\{a\}} P_2' \\
P_1 &:= (a, \lambda_a).(b, \lambda_b).P_1 \\
P_2' &:= (a, 1).(\tau, \lambda_2).P_2' \\
AP_2 &:= P_2'' \parallel_{\{c\}} P_3 \\
P_2'' &:= (\tau, \lambda_1).(c, 1).P_2'' \\
P_3 &:= (c, \lambda_c).(d, \lambda_d).P_3
\end{aligned}
$$

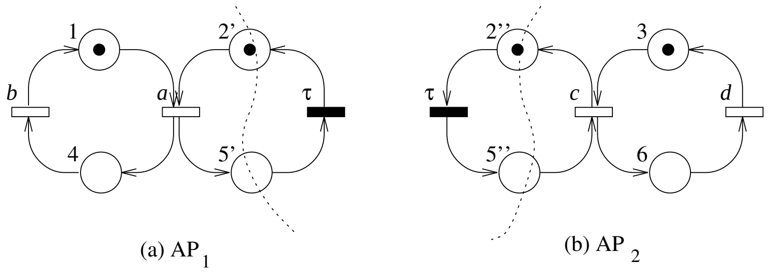


Figure 7: Aggregated processes

This simple example should motivate our definition of SISO cuts for DFPs. There must be two distinct partitions $\mathcal{P}_1, \mathcal{P}_2$ of the set of sequential components $\{P_1, P_2, P_3, \ldots\}$ with the property that there is no interaction across the partitions. Interaction across the partitions is realized by an *interface process* P_I that "visits" $\mathcal{P}_1, \mathcal{P}_2$ in consecutive order, i.e. it can interact with both sides, but it may also perform actions independently from $\mathcal{P}_1, \mathcal{P}_2$. In our example, $\mathcal{P}_1$ would be $\{P_1\}$, $\mathcal{P}_2 = \{P_3\}$, and $P_I = \{P_2\}$. A formalization of this first simple decomposition scheme is presented in the next subsection.

3.3 SISO cuts

Definition 3.2 *For a given sequential process* $P_i \in \mathcal{L}_{seq}$ *we define the relation* $<_{P_i} \subseteq Sort(P_i) \times Sort(P_i)$ *as*

$$
a <_{P_i} b \Leftrightarrow (\exists Q, Q', R, R' \in ds(P_i))(P_i \xrightarrow{\overline{ab}}{}^{*} Q \xrightarrow{a,\cdot} Q' \xrightarrow{\overline{ab}}{}^{*} R \xrightarrow{b,\cdot} R' \xrightarrow{\overline{ab}}{}^{*} P_i)
$$

where $p \xrightarrow{\overline{ab}}{}^{*} q$ *denotes an action sequence that does not contain the actions a,. The reflexive closure of* $<_{P_i}$ *is defined as* $a \leq_{P_i} \Leftrightarrow a <_{P_i} b \vee a = b$

Definition 3.3 *Be* $P \in \mathcal{L}_{par}^{n}$ *a DFP with* n *sequential components* $P_1, \ldots, P_n$*. A* SISO-cut *is formally characterized by a partitioning scheme*

$$\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_I\} \in 2^{\{P_1,\ldots,P_n\}}$$

where

1. $\mathcal{P}_I = \{P_I\}$ (interface process)
2. $(\forall p \in \mathcal{P}_1)(\forall q \in \mathcal{P}_2)(\mathcal{I}_A(P :: p, q) = \emptyset)$
3. $\bigcup_{p \in \mathcal{P}_1} \mathcal{I}_A(P :: P_I, p) \neq \emptyset$
4. $\bigcup_{q \in \mathcal{P}_2} \mathcal{I}_A(P :: P_I, q) \neq \emptyset$

and two interface actions $a, b \in Sort(P_I)$ *such that*

$$\begin{array}{l}(\forall h \in \mathcal{I}_A(P_I))\,(a \leq_{P_I} h \leq_{P_I} b \;\Rightarrow\; (\exists p \in \mathcal{P}_1)(\mathcal{I}_A(P :: p, P_I) \neq \emptyset)) \quad \wedge \\ (\forall h \in \mathcal{I}_A(P_I))\,(b \leq_{P_I} h \leq_{P_I} a \;\Rightarrow\; (\exists p \in \mathcal{P}_2)(\mathcal{I}_A(P :: p, P_I) \neq \emptyset))\end{array}$$

The construction of aggregated processes AP_1, AP_2 for a DFP with a SISO cut is now relatively straightforward. Without loss of generality, assume that $\mathcal{P}_1 = \{P_1, \ldots, P_{k-1}\}$, $\mathcal{P}_2 = \{P_{k+1}, \ldots, P_n\}$, and $\mathcal{P}_I = \{P_k\}$. Then, the aggregated process AP_1 can be obtained by removing all processes that belong to $\mathcal{P}_2$ and replacing P_I by a process Q_1 that performs repeatedly all actions between a and b with an adjusted rate for action b that will be determined later. AP_2 is constructed by analogy:

$$\begin{array}{rcl} AP_1 & := & P_1 \parallel_{S_1} P_2 \parallel_{S_2} \cdots \parallel_{S_{k-2}} P_{k-1} \parallel_{S_{k-1}} Q_1 \\ AP_2 & := & Q_2 \parallel_{S_k} P_{k+1} \parallel_{S_{k+1}} P_{k+2} \cdots \parallel_{S_{n-2}} P_{n-1} \parallel_{S_{n-1}} P_n \end{array}$$

For the construction of the new interface processes Q_i we need the following function:

Definition 3.4 *The function* $cut : \mathcal{L}_{seq}^{\setminus +} \times Act \times Act \rightarrow \mathcal{L}_{seq}^{\setminus +}$ *is defined as*

$$\begin{array}{rcl} cut((x,\lambda).P, a, b) & = & \left\{ \begin{array}{ll} cut(P,a,b) & \textit{if} \quad a <_P x <_P b \\ (\tau,\lambda).cut(P,a,b) & \textit{if} \quad x = b \\ (x,\lambda).cut(P,a,b) & \textit{otherwise} \end{array} \right. \\ cut(recX : P, a, b) & = & recX : cut(P,a,b) \\ cut(X,a,b) & = & X \\ cut(Stop,a,b) & = & Stop \end{array}$$

and removes from a sequential component all actions that are performed between actions a *and* b*.*

The new interface processes are obtained by

$$Q_1 \;=\; cut(P_k, b, a) \qquad \text{and} \qquad Q_2 \;=\; cut(P_k, a, b)$$

3.4 Aggregation and Response Time Approximation

With the formalization of SISO-cuts through DFPs we are able to perform the basic steps that are necessary in order to approximate the throughput of such processes. As above, let AP_1, AP_2 denote the aggregate processes and a, b be their interface actions, whose durations haven't been determined yet. For this we need another processes BS, the *basic skeleton system*, which is defined as

$$BS := (a, \lambda).(b, \mu).BS$$

The last section showed how to obtain the aggregated processes AP_i in detail. The basic skeleton just performs repeatedly the interface actions a, b.

The solution algorithm that we implement is an adaption of the iterative RTA algorithm for MGs that is explained in [Jun92]:

1. **Set initial values:**
$$\lambda \;=\; 1$$
2. **Solve aggregated process 2:** Compute the throughput $\mathcal{X}$ of AP_2 by solving its underlying CTMC.
3. **Response time approximation:** Using the basic skeleton process we determine an approximation for the rate of action a as follows: We require that BS has the same throughput $\mathcal{X}$ as AP_2 and compute the new rate of action b. The throughput of BS is given by
$$\mathcal{X} = \frac{\lambda\mu}{\lambda + \mu}$$
Hence, the rate μ of action b is
$$\mu = -\frac{\lambda\mathcal{X}}{\mathcal{X} - \lambda}$$
4. **Solve aggregated process 1:** Compute the throughput $\mathcal{X}$ of AP_1 by solving its underlying CTMC.
5. **Response time approximation:**
$$\lambda = -\frac{\mu\mathcal{X}}{\mathcal{X} - \mu}$$

Steps 2-5 are repeated until convergence of $\mathcal{X}$ is reached. The result X of this algorithm is the approximate throughput of our original process (in a DFP all actions have the same throughput).

3.5 Process-algebraic characterization of SISO cuts

In comparison to QNs/SPNs it is more difficult to find an informal interpretation of the aggregation procedure. This is due to the totally different nature of SPA models where we don't have an explicit notion of stations and customers. Instead, we have to find a process-algebraic interpretation of aggregation based on SISO cuts.

Aggregation of a MG Petri net can be regarded as a sort of abstraction. A certain part of the model is substituted by a single transition. Thus, from a technical point of view the aggregation procedure in process-algebraic terms can be regarded as an abstraction of certain actions which can be carried out by means of the hiding operator. The actions to be hidden are performed by a part of the system that interacts with its environment only through a special interface process.

Formally, the aggregation problem can be formulated as follows: For a given DFP $P = (P_{ab} \parallel_A P_I) \parallel_B P_{ba}$ find new interfaces Q_1, Q_2 which fulfill

$$\begin{array}{rcll} (P_{ab} \parallel_A P_I \parallel_B P_{ba}) \backslash C_1 & \approx & P_{ab} \parallel_A Q_1 & \text{and} \\ (P_{ab} \parallel_A P_I \parallel_B P_{ba}) \backslash C_2 & \approx & Q_2 \parallel_B P_{ba} & \text{where} \end{array}$$

$$\begin{array}{rcl} C_1 & = & Sort(P_{ba}) \cup \{h \in Sort(P_I) | b <_{P_I} h <_{P_I} a\} \\ C_2 & = & Sort(P_{ab}) \cup \{h \in Sort(P_I) | a <_{P_I} h <_{P_I} b\} \end{array}$$

where $\approx$ denotes a behavioural equivalence with respect to the functional behaviour.

The last section showed one efficient way to obtain the aggregate processes. Here we want to prove that the resulting aggregate processes are functionally equivalent with respect to the remaining activities. To ease the notation we denote in the sequel by H_1, H_2 the functions that compute the aggregate processes AP_1, AP_2 for a given DFP $P \equiv P_{ab} \parallel_A P_I \parallel_B P_{ba}$:

$$\begin{array}{rcll} H_1(P) & := & P_{ab} \parallel_A Q_1 & \text{where} \quad Q_1 = cut(P_I, b, a) \\ H_2(P) & := & P_{ba} \parallel_B Q_2 & \text{where} \quad Q_2 = cut(P_I, a, b) \end{array}$$

where $P \in \mathcal{L}^n_{par}$, $P_{ab} \in \mathcal{L}^{k-1}_{par}$ and $P_{ba} \in \mathcal{L}^{n-k}_{par}$.

Proposition 3.1 *Be $P \equiv P_{ab} \parallel_A P_I \parallel_B P_{ba} \in \mathcal{L}^n_{par}$ a DFP with SISO-cut $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_I$ and interface actions a, b. Then, the resulting aggregate processes are weak bisimilar to the original process with respect to the activities performed by the aggreagate processes:*

$$H_1(P) \approx P \backslash C_1 \qquad \text{and} \qquad H_2(P) \approx P \backslash C_2$$

where

$$\begin{array}{rcl} C_1 & = & Sort(P_{ba}) \cup \{h \in Sort(P_I) | b <_{P_I} h <_{P_I} a\} \\ C_2 & = & Sort(P_{ab}) \cup \{h \in Sort(P_I) | a <_{P_I} h <_{P_I} b\} \end{array}$$

Proof: We show that $H_1(P) \approx P \backslash C_1$. The second equivalence can be shown quite similarly.

$$P \backslash C_1 = (P_{ab} \parallel_A P_I \parallel_B P_{ba}) \backslash C_1 = P_{ab} \parallel_A (P_I \parallel_B P_{ba}) \backslash C_1$$

because $C_1 \cap Sort(P_{ab}) = \emptyset$. Since $H_1(P) = P_{ab} \parallel_A Q_1$ we just have to show that

$$Q_1 = cut(P_I, b, a) \approx P_I \parallel_B P_{ba} \backslash C_1$$

Be $\mathcal{B} := \{(p,q) \mid p \in ds(P_I \,\|_B P_{ba} \backslash C_1) \;\wedge\; q \in ds(Q_1) \wedge (\forall x \in Com)(p \xrightarrow{x,\cdot} p' \Leftrightarrow q \xrightarrow{x,\cdot} q')\}$

By definition, $(P_I \,\|_B P_{ba} \backslash C_1, Q_1) \in \mathcal{B}$. We show now that $\mathcal{B}$ is a weak bisimulation.

For $(E,F) \in \mathcal{B}$ we have to show that:

(i) if $E \xrightarrow{u,\cdot} E'$ then, for some F', $F \stackrel{\hat{u}}{\Longrightarrow} F'$ and $(E',F') \in \mathcal{B}$

(ii) if $F \xrightarrow{u,\cdot} F'$ then, for some E', $E \stackrel{\hat{u}}{\Longrightarrow} E'$ and $(E',F') \in \mathcal{B}$

where the relation $E \stackrel{\hat{u}}{\Longrightarrow} E'$ denotes the following sequence of transitions:

$$E \xrightarrow{\tau,\cdot}{}^{*} \,.\, \xrightarrow{u,\cdot} \,.\, \xrightarrow{\tau,\cdot}{}^{*} E'$$

(i) $E \equiv (R \,\|_B S) \backslash C_1 \quad (R \in ds(P_I))$. Since $Sort(S) \subseteq C_1$, the only visible actions that can be observed are performed by R. However, R has exactly the same behaviour as Q_1 until the action b is reached, which is hidden of course. Thus, we can say

$$E \xrightarrow{u,\cdot} E' \quad \Rightarrow \quad F \xrightarrow{u,\cdot} F' \;\wedge\; (E',F') \in \mathcal{B}$$

until no visible action is enabled anymore, i.e. $R' \xrightarrow{b,\cdot} R''$. In that case all successor states are in relation to the last state of Q_1 before a gets enabled again.

(ii) This direction can be shown in analogy to (i).

Consequently, $\mathcal{B}$ is a weak bisimulation and we can deduce

$$\begin{aligned}
(P_I \,\|_B P_{ba} \backslash C_1, Q_1) \in \mathcal{B} &\Rightarrow P_I \,\|_B P_{ba} \backslash C_1 \approx Q_1 \\
&\Rightarrow P_{ab} \,\|_A (P_I \,\|_B P_{ba} \backslash C_1) \;\approx\; P_{ab} \,\|_A Q_1 \\
&\Rightarrow P \backslash C_1 \approx H_1(P)
\end{aligned}$$

□

4 Future work

We demonstrated one possibility to employ decomposition techniques taken from stochastic Petri nets for the analysis of classes of SPA models. The models we considered are called Decision Free Processes and correspond to 1-safe Marked Graphs. We presented characterizations of Single Input Single Output cuts (SISO) as a way to decompose DFPs into two aggregates that can be solven in isolation. The overall throughput of the system is approximated using an iterative response time approximation algorithm.

Next thing to do is to generalize the above results. One direction is to investigate more general decomposition schemes, like canonical SISO cuts, SIMO cuts, and MIMO cuts. The other direction for future work is the generalization towards more expressive classes of models.

References

[Her93] H. Hermanns. Semantik für Prozeßsprachen zur Leistungsbewertung. Master's thesis, Universität Erlangen–Nürnberg, IMMD VII, November 1993.

[HM95] J. Hillston and V. Mertsiotakis. A Simple Time Scale Decomposition Technique for Stochastic Process Algebras. *The Computer Journal*, special issue, 1995. Proc. of the 3rd Workshop on Process Algebras and Performance Modelling.

[HR94] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 71–88, Regensberg/Erlangen, July 1994.

[JSS92] H. Jungnitz, B. Sanchez, and M. Silva. Approximate Throughput Computation of Stochastic Marked Graphs. *Journal of Parallel and Distributed Computing*, 15(3):282–295, July 1992.

[Jun92] H. Jungnitz. *Approximation Methods for Stochastic Petri Nets*. PhD thesis, Rensselaer Polytechnic Institute, May 1992.

[Rib94] M. Ribaudo. Understanding Stochastic Process Algebras via their GSPN Semantics. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, Regensberg/Erlangen, July 1994.