UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

PERFORMANCE EVALUATION OF MOBILE PROCESSES VIA ABSTRACT MACHINES

Chiara Nottegar, Corrado Priami and Pierpaolo Degano

2001

Technical Report # DIT-02-0020

Also: appeared in IEEE Transactions on Software Engineering, 27, 10 (2001), 867-889.

Performance Evaluation of Mobile Processes via Abstract Machines *

Chiara Nottegar, Corrado Priami
Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie 15, I-37134 Verona, Italy
priami@sci.univr.it
and
Pierpaolo Degano
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy
degano@di.unipi.it

Abstract

We use a structural operational semantics which drives us in inferring quantitative measures on system evolution. The transitions of the system are labelled and we assign rates to them by only looking at these labels. The rates reflect the possibly distributed architecture on which applications run. We then map transition systems to Markov chains, and performance evaluation is carried out using standard tools. As a working example, we compare the performance of a conventional uniprocessor with a prefetch pipeline machine. We also consider two case studies from the literature involving mobile computation to show that our framework is feasible.

Keywords. Calculi for Mobility, Enhanced Operational Semantics, Formal Methodology, Performance Evaluation, Stochastic Models.

1 Introduction

The design of distributed and concurrent systems requires the management of a huge amount of information (such as the structure of interconnection

*An extended abstract of this paper can be found in the proceedings of *FASE'99*. The last two authors have been partially supported by the MURST project TOSCA.

and synchronization, and the allocation and management of resources) along with the cooperation of a large group of people (designers, implementors, performance and quality analysts, etc.). A formal specification of the system being planned may help both to organize human interactions and avoid misunderstandings, and also to monitor the work of the implementors. The software modules produced must behave according to their specifications, against which they have to be checked. In the area of safety-critical control systems and secure systems (nuclear power stations, traffic alert and collision avoidance systems, railway signalling systems, medical instruments, etc.), where system failure may cause loss of human life, environmental damage or financial loss, the verification must be certified.

Often, specifications must also include performance constraints on system behaviour. A typical example is a distributed system for reserving plane seats. An implementation must be rejected when a reservation takes hours, even if it matches its behavioural specification.

We are mainly interested in specifying and evaluating distributed applications, especially those involving code migration, although our approach can be used in other fields as well. We shall describe systems through a calculus for mobility, endowed with an enhanced version of structural operational semantics (SOS) [55]. Our semantics [22] makes it possible to mechanically derive Markov chains, once the user has given additional information about the rates of system activities. More precisely, it suffices to have information about the activities performed by the components of a system in isolation, and on some features of its architecture. These values will then be composed to compute the rates of the activities of the whole system. The actual performance evaluation is then carried out using standard techniques and tools [66, 58, 61].

In practice, performance analysis is often delayed until a system is completely implemented. This delay may cause high extra-costs to the implementation, so one needs to measure a system as soon as possible and to closely integrate behavioural and performance analysis [33]. Since several tools are available to check the behaviour of systems specified within the SOS approach (which our proposal relies on) this integration becomes possible. Also, we can analyse the performance of systems from the very beginning of their design, as we use the very same specification language. What we are proposing is therefore a first step towards the development of a single, formal design methodology that supports its users in analysing (mobile) systems, following a (semi-) mechanizable procedure.

To assess our proposal, we applied it to a case study presented in the literature [4]. It considers a network management system, in which systems operate in highly distributed settings. We compare the implementations in the classical paradigms based on Remote Procedure Call and on Remote Evaluation (the latter exploits code migration). Our results coincide with

those of [4]. This makes us confident that our approach is sensible and applicable. Further support comes from a larger example, also taken from the literature on performance: the generalized polling system studied in [36] through stochastic process calculi and in [44, 39] through stochastic Petri nets.

We feel confident that our proposal can scale up. Indeed, many languages for reactive and mobile systems, used in industrial application as well (e.g. Facile [65], PICT [54], CML [53], Esterel [6] [1]), are built on top of a core process calculus like the one we use here, which is exploited as their intermediate language. This could also be done for Java-like languages (for a translation into a process calculus of the main features of Java see [34]). The formal connections between a real language and the core calculus might thus help the user to provide the information about performance mentioned above, in the form of annotations on the compiled code.

The paper is organized as follows. The next section briefly discusses process calculi and their use in performance analysis; also, it intuitively presents our approach. In Sect. 3 we recall the semantics of the HOπ, the process calculus we shall use, and the main concepts about continuous time Markov chains (CTMCs for short). In Sect. 4 we introduce a sample function that assigns rates to transitions. We show how to obtain the CTMC associated with a given process and how to extract performance measures from it, mainly about network traffic. As a simple running example, we compute and compare the performance of a conventional uniprocessor architecture and of a prefetch pipeline machine. We also outline how our proposal can be used for CPU load analysis. Section 5 presents the case study on a network management system, and we show that our results agree with those obtained via classical evaluation techniques. A generalized polling system is studied in Sect. 6.

2 Process Algebras and Quantitative Measures

One of the most interesting paradigms for concurrent and distributed systems is based on process calculi, or process algebras, such as CCS [45], TCSP [37], and the π-calculus [47], the *join-calculus* [27, 28], Plain-CHOCS [64], Dπ [59], the *ambient* calculus [12]. Particularly important for our study are the

[1] Facile has been used to implement the Calumet teleconferencing system [62, 63] (a support to cooperative work through real-time presentations of visual and audio information across WAN), and Einrichten [2] (an application that combines distribution, sophisticated graphics, and live video for collaborative interior design work). Esterel has been used as well in industrial application (see e.g., [17, 15, 40, 50]). Esterel [43] and Facile are already equipped with an enhanced operational semantics, and Facile has been used to check some qualitative properties of mobile agent systems [7, 24].

last ones, through which computations of mobile agents can be specified at various levels of abstraction.

These calculi describe a system in terms of the actions it can perform. The actions are put together via a few basic operators, such as sequential and parallel composition, nondeterministic choice and scope restriction. Sequentialization $P; Q$ of systems P and Q is often reduced to prefixing $a.Q$, i.e. P can only be the prefix a. A prefix is a single atomic action, standing for some system activity. Intuitively, the system $a.Q$ first performs a and then behaves like Q.

The behaviour of a system is usually represented through a directed graph called a *transition system*, specified using the structural operational semantics (SOS) approach [55]. The nodes are the *configurations* (or *states*) that the system can pass through, and are represented by expressions of the calculus (*processes*). The arcs represent the *transitions* from one configuration to another, i.e. the occurrence of activities. Transitions can be labelled by information on which activities they represent, yielding *labelled* transition systems. The key point of the SOS approach is that the transitions are deduced according to a set of inference rules driven by the syntax.

A transition system can be seen as the specification of an abstract machine. For example, in the sequential setting, the states consist of the program itself (with an instruction counter) and the store, plus some auxiliary data which represent the data structures on which the program works, e.g., the activation stack. A language can be specified by a hierarchy of abstract machines, described as transition systems, which are increasingly closer to the actual interpreter or compiler. The abstract machines can be formally related to each other, so that a higher level transition (in the "specification") is mapped onto a sequence σ of lower level ones (its "implementation"); see, e.g., the *VDM* approach [41] and in the process algebra field [20]). Technically, the sequence σ is tightly connected to a linearization of the deduction of the transition in hand. By composing the connecting relations along the hierarchy, a transition representing an activity a can be linked to the steps that (the specification of) the actual machine performs to execute a.

In the field of process algebras, a lot of attention has been devoted to the behavioural analysis of systems. The aim is to establish some equality relations (usually based on bisimulation [45]) between two descriptions of the same system to ensure that they behave the same. Using these equivalences, one can prove that an implementation behaves according to its specification.

Besides behavioural analysis, quantitative measures are also relevant while developing a system. The literature presents some extensions to enrich process algebras with quantitative information such as time [52] and probabilities [67, 31, 42]. A further step is made by stochastic process algebras [30, 36, 5, 11, 56, 9, 32, 18, 35] which associate probabilistic distributions

with prefixes. These become pairs of the form (a, F), with the intuition that a is performed and completed in a time drawn from the distribution F. Usually F is assumed to be an exponential distribution,[2] and prefixes have the form (a, r), where r is the parameter that characterizes such distributions. The exponential distribution enables us to recover a continuous time Markov chain from which performance measures are computed by using standard numerical techniques. The syntactic extension of stochastic process algebras has a potential drawback: either the rates are all symbolic (and different), or they have to reflect a current architecture. In the first case, the calculations needed to analyse the performance may become quite heavy; in the second the designers should know some features of the architecture right from the very beginning of project development.

We propose instead to leave the syntax of the calculus unchanged, yet retaining the peculiarities of stochastic process algebras. As mentioned above, our idea is to mechanically derive the probabilistic distributions from the labels of the transition system and from some additional information about the architecture of the target machine. In this way, we partially alleviate the problem mentioned above, and we can reuse existing tools for behavioural analysis.

Our starting point is an enhanced version of operational semantics called *proved* [22]. It is a parametric model used in [21, 23] to uniformly describe different qualitative aspects of processes. In our proved operational semantics the transition labels are enhanced so that they record the application of inference rules, namely the proofs. By inspecting enhanced labels we derive the rates of transitions, as the proof of a transition represents the low level routines performed by the run-time support to execute the transition itself. Since the rates are also affected by the target architecture, we consider its aspects relevant to performance evaluation as parameters. Pushing the connection proofs/run-time support further, we would like to delegate the instantiation of these parameters to the compiler of the language. Indeed, the compiler knows the architecture for which it has been written, and it can annotate (or at least suggest how to) the intermediate code with information about performance. Clearly, different target architectures originate different parameter instantiations. In this way, the same definition can describe the behaviour and the performance of a system running on different target machines or networks. In particular, we can compare two systems running on

[2] An exponential distribution with rate r is a function $F(t) = 1 - e^{-rt}$, where t is the time parameter. The value of $F(t)$ is smaller than 1 and $\lim_{t\to\infty} F(t) = 1$. The shape of $F(t)$ is a curve which monotonically grows from 0 approaching 1 for positive values of its argument t. The parameter r determines the slope of the curve. The greater r, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time x is $F(x) = 1 - e^{-rx}$, so r determines the time, Δt, needed to have a probability near to 1.

the same architecture, as well as two different architectures devised to run the same application.

To present our proposal in a pure setting, we adopt here the higher order π-calculus (HOπ) [60]. This is an extension of the π-calculus [47] to describe distributed and concurrent systems which allows values and port names to be transmitted in communications between the processes. The HOπ extends first-class values with processes allowing description of code mobility, as well. We recall it in the next section. Since our technique can be applied to any language with an operational semantics, no real limitation arises from this choice.

3 Background

This section recalls the syntax and the semantics of the HOπ and the main basic concepts about CTMCs.

3.1 Higher Order π-calculus

The HOπ (Higher-Order π-calculus) [60] extends the π-calculus of [47], a foundational model of concurrent communicating processes based on the notion of *name passing*. In the higher order version, names can represent values, links and also processes. Thus communications may cause processes to migrate. We slightly enrich the standard syntax to better express quantitative analysis. In particular, we use a set of prefixes τ_i indicating invisible actions rather than a single τ, in order to represent the fact that different internal actions may have different durations.

Definition 3.1 *Let $\mathcal{N}$ be a countably infinite set of* names*, ranged over by $a, b, \ldots, x, y, \ldots$, and let $\mathcal{S} = \{\tau_0, \tau_1, \tau_2, \ldots\}$ be a countably infinite set of* invisible actions *ranged over by τ_i, with $\mathcal{N} \cap \mathcal{S} = \emptyset$. We also assume a set of* agent identifiers*, each with its arity, ranged over by $A, A_1, \ldots$ and a set $\mathcal{V}$ of process variables ranged over by $X, Y, \ldots$. Then, the set $\mathcal{P}$ of* processes*, ranged over by $P, Q, R, \ldots$, is defined by the following syntax*

$$P ::= \mathbf{0} \mid \pi.P \mid P + P \mid P|P \mid (\nu x)P \mid [x = y]P \mid A(U_1, \ldots, U_n)$$

where U_i stands for a process variable or for a name. The prefix π may be either τ_i for silent *moves, or $x(U)$ for* input*, or $\overline{x}K$ for* output*, where K stands for a process or for a name. In the last cases x is called* subject*, while U and K are called* objects*.*

Hereafter, the trailing $\mathbf{0}$ will be omitted in the definition of processes.

3.1.1 Informal semantics

The prefix π is the first atomic action that the process $\pi.P$ can perform. A silent prefix τ_i denotes an action which is internal to the system. The input prefix $x(U)$ binds the occurrences of the process variable or name U in the prefixed process P. Intuitively, a name or a process will be received along the link named x and it will substitute the free occurrences of placeholder U in the process prefixed by the input.[3] The output prefix $\overline{x}K$ does not bind the name or process K which is sent along x.

Summation denotes nondeterministic choice. The process P_1+P_2 behaves either like P_1 or P_2.

The operator $|$ describes parallel composition of processes. The components of $P_1|P_2$ may act independently. Also, an output action of P_1 (P_2) at any output port $\overline{x}$ may synchronize with an input action of P_2 (P_1) at x to create a silent action denoting the communication. The value sent by P_1 replaces the relevant occurrences of the placeholder U in P_2 (see the comment above on input).

The restriction operator (νx) acts as a static declaration of (i.e. a binder for) the name x in the process P that it prefixes. In other words, x is a unique name in P which is different from all external names. The agent $(\nu x)P$ behaves like P, except that actions at ports $\overline{x}$ and x are prohibited. However communications along link x of components *within* P are allowed because the result is a silent action. Restriction affects the higher-order features of the calculus as well. Consider the process

$$P = x(U).(U|U)|(\nu\, a)\overline{x}Q \tag{1}$$

where $Q = \overline{a}b+a(y)$. A possible transition models the transmission of process Q along link x; Q replaces the two occurrences of the placeholder U. The neat result is the process displayed below, where two occurrences of Q are both enabled and may run in parallel. Furthermore, Q uses the private name a of the right-hand component of the parallel composition. Hence we must enlarge the scope of $(\nu\, a)$ after the communication to include the left-hand component of the top-level $|$. Actually, we reach the state

$$(\nu\, a)((Q|Q)|\mathbf{0})$$

(see the next sub-section for a formal derivation of the transition).

Matching $[x = y]P$ is an `if-then` operator: process P is activated only if $x = y$.

An agent is a static definition of a parameterised process that becomes a process when its parameters are instantiated. Each agent identifier A

[3] This is the so-called *late* semantics, because the binding of U takes place at communication time, as opposed to *early* semantics [48] which we do not consider in this paper.

has a unique defining equation of the form $A(\tilde{U}) = P$ (hereafter, $\tilde{U}$ denotes $U_1, \ldots, U_n$, where the U_i are all distinct and are the only names occurring free in P; see the next sub-section for the definition of free names.) The equation $A(\tilde{U}) = P$ is akin to the definition of function: it is invoked by $A(\tilde{K})$ and triggers the substitution of the actual parameter K_i for the corresponding formal parameter U_i.

3.1.2 Operational semantics

For the (proved) operational semantics of the HOπ, we need to define the set of *actions*, which will be the basic ingredients of the transition labels. The actions are almost the same as prefixes, from which they are obtained. The set of silent actions will be simply $\mathcal{S}$. The other actions (i.e. input and output, sometimes called *visible*) form the set $\mathcal{A}$, with typical element α. More precisely, α can be $x(U)$ for input, $\overline{x}K$ for *free* output just as prefixes; furthermore it can be $\overline{x}(K)$ for *bound* output. This new action is originated by the interplay between output prefix and restriction, as shown by rule *Open* in Tab. 1. Roughly speaking, the effect of a bound output is vanishing a (νx) operator, as in $Q = (\nu x)\overline{y}x.P \xrightarrow{\overline{y}(x)} P$. The intuition behind this operation is to make the name x, which is private to Q, available to the external environment. Recall that the operator (νx) can be interpreted as a declaration: the bound output then enlarges the scope of the declaration. For this reason a bound output is sometimes referred to as *scope extrusion* in the literature.

Now, we define our *enhanced labels* in the style of [19, 8, 21]. The label of a transition records the inference rules used during its deduction, besides the action itself. This is the only difference between the proved semantics and the standard one [47]. Below, we also introduce a function ℓ which takes an enhanced label to the corresponding standard action label, thus recovering the original semantics of [47]. For a detailed presentation of the proved semantics of the π-calculus and of the HOπ, see [23].

Definition 3.2
Let $\mathcal{L} = \{\|_0, \|_1\}$ *with* $\chi \in \mathcal{L}^*$, $\mathcal{O} = \{+_0, +_1, =_m, (\nu x), \tilde{m}\} \ni o$ *(with* $m, m_i \in \mathbb{R}^+, \forall m_i \in \tilde{m}$*) and let* $\vartheta \in (\mathcal{L} \cup \mathcal{O})^*$. *Then the set* Θ *of* enhanced labels (*with typical element* θ) *is defined by the following syntax*

$$\theta ::= \vartheta\alpha \mid \vartheta\tau_i \mid \vartheta\langle\|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1\rangle$$

with α_0 (α_1 *resp.) is* $x(U)$ *iff* α_1 (α_0 *resp.) is either* $\overline{x}K$ *or* $\overline{x}(K)$.
The function ℓ *is defined as* $\ell(\vartheta\alpha) = \alpha, \quad \ell(\vartheta\tau_i) = \ell(\vartheta\langle\|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1\rangle) = \tau$.

We partition the set of tags in two only to simplify the presentation of our cost function in Sect. 4.1. Intuitively, the tags will be used to keep track

of the context in which actions occur; we shall give more intuition below, while commenting on the rules of the proved operational semantics. We only mention here that the tags in $\mathcal{L}$ are concerned with the parallel structure of processes, and are therefore linked to the parallel composition "|"; those in $\mathcal{O}$ concern, so to speak, the sequential structure of processes.

We recall a couple of auxiliary definitions. The free names *fn*, bound names *bn*, and names $n = fn \cup bn$ of a standard label $\ell(\theta)$ are as follows.

$\ell(\theta)$	*Kind*	$fn(\ell(\theta))$	$bn(\ell(\theta))$
τ_i	Silent	$\emptyset$	$\emptyset$
$x(U)$	Input	$\{x\}$	$\{U\}$
$\overline{x}K$	Free Output	$\{x, K\}$	$\emptyset$
$\overline{x}(K)$	Bound Output	$\{x\}$	$\{K\}$

Functions *fn*, *bn* and n are extended in the obvious way to enhanced labels and to processes by considering the input prefix $x(U)$ and the restriction $(\nu\, x)$ as binders. We sometimes write $(\nu x, y)P$ for $(\nu x)(\nu y)P$.

A *variant* of $P \xrightarrow{\theta} Q$ is a transition which only differs in that P and Q have been replaced by processes that are α-equivalent (i.e. that only differ in the choice of bound names), and $\ell(\theta)$ has been α-converted, where a name bound in $\ell(\theta)$ includes Q in its scope. For example the transitions $x(b)\,.\,\overline{b}z \xrightarrow{x(b)} \overline{b}z$, $x(y)\,.\,\overline{y}z \xrightarrow{x(b)} \overline{b}z$ and $x(y)\,.\,\overline{y}z \xrightarrow{x(y)} \overline{y}z$ are variants of each other. Hereafter, a transition stands for all its variants.

The (schemata of the) inference rules of the proved semantics of the HOπ are in Tab. 1. The table has the rules for the actual transitions $P \xrightarrow{\theta} Q$, and for an auxiliary transition relation $\xrightarrow{\theta}_I$. All the rules are interpreted as follows: if the transition(s) above the inference line (premises) can be deduced, then so is the transition below the line (conclusion).

We briefly describe the intuition of the rules in the auxiliary transition relation $\xrightarrow{\theta}_I$. The additional index I is a set of names that require special care, since they are bound.

The axiom *Act* is the rule from which we start the derivation of any transition. It essentially says how a prefix of the calculus originates a transition that in its label records the information on the kind of prefix (silent, input, output) fired. In rule Sum_0 (Sum_1) the intuition behind tag $+_0$ ($+_1$) is to record that the left (right) alternative of a choice has been selected. Rule Par_0 (Par_1) adds a tag $\|_0$ ($\|_1$) to the label of its conclusion, because the left (right) component is moving. Its side condition guarantees that if $\theta = \vartheta\overline{x}(K)$, i.e. it is a bound output of object K with bound names $bn(\theta)$, no clash will occur between $bn(\theta)$ and the names free in Q. Note that the names bound in K are recorded in the set I by rule *Open*, see below. Restriction on name x is

represented in labels by (νx). It signals that the restricted name is no longer available. We record the resolution of a matching through tag $=_m$, where the positive real number m is the size of the actual data compared. Pairs are used to signal interaction and to record the components which actually interacted (and the proofs of the relevant transitions). So, the rule Com_0 (and its symmetric Com_1) has a pair instead of the usual τ in its conclusion. Note that the placeholder U, which is the subject of the input action, is replaced by the subject of the output, which is an actual datum, both in the residual of the input and in the label of the communication. The invocation of a definition enriches the label of the conclusion of rule *Ide* with the tag $\tilde{m}$, the vector of the sizes of the actual parameters $\tilde{K}$.

We now comment on the rules that handle the set I in the auxiliary transition relation $\xrightarrow{\theta}_I$. As mentioned before, this set contains names extruded while communicating them, or communicating a process in which they occur. Rule *Open* assigns to I all the names to be extruded. Then, the rule $Close_0$ (symmetrically $Close_1$) empties I while using it to include the receiving process as well in the scope of the extruded names (note that here $I \subseteq fn(K)$, because of the side condition in rule *Open*).

Finally, the rule $HO\pi$ discards the index I and gives the actual transitions.

To clarify the interaction of extrusion and higher-order mobility, consider again the process P in (1). The derivation of the communication along channel x according to the rules in Tab. 1 is

$$\dfrac{\dfrac{x(U).(U|U) \xrightarrow{x(U)}_{\emptyset} U|U, \quad \dfrac{\overline{x}Q \xrightarrow{\overline{x}Q}_{\emptyset} \mathbf{0}}{(\nu\, a)\overline{x}Q \xrightarrow{\overline{x}(Q)}_{\{a\}} \mathbf{0}}}{x(U).(U|U)|(\nu\, a)\overline{x}Q \xrightarrow{\langle ||_0 x(Q), ||_1 \overline{x}Q\rangle}_{\emptyset} (\nu\, a)((Q|Q)|\mathbf{0})}}{x(U).(U|U)|(\nu\, a)\overline{x}Q \xrightarrow{\langle ||_0 x(Q), ||_1 \overline{x}Q\rangle} (\nu\, a)((Q|Q)|\mathbf{0})}$$

As mentioned above, the ϑ part of an enhanced label $\vartheta\alpha$ keeps track of the context in which the prefix originating the corresponding transition is plugged (and likewise for communications). In fact, transitions are deduced by induction on the syntax of processes that drives the selection of inference rules.

It is straightforward to obtain the original semantics from the proved one by applying function ℓ in Def. 3.2 to each enhanced label in Tab. 1.

3.1.3 Some auxiliary definitions

The rules in Tab. 1 define the transition system of the whole $HO\pi$. It is a graph in which processes form the nodes, and the arcs represent the

$Act:\ \pi.P \xrightarrow{\pi}_{\emptyset} P$

$Sum_0:\ \dfrac{P \xrightarrow{\theta}_I P'}{P+Q \xrightarrow{+_0\theta}_I P'}$ $\qquad$ $Sum_1:\ \dfrac{P \xrightarrow{\theta}_I P'}{Q+P \xrightarrow{+_1\theta}_I P'}$

$Par_0:\ \dfrac{P \xrightarrow{\theta}_I P'}{P|Q \xrightarrow{\|_0\theta}_I P'|Q},\ I \cap fn(Q)=\emptyset$ $\qquad$ $Par_1:\ \dfrac{P \xrightarrow{\theta}_I P'}{Q|P \xrightarrow{\|_1\theta}_I Q|P'},\ I \cap fn(Q)=\emptyset$

$Res:\ \dfrac{P \xrightarrow{\theta}_I P'}{(\nu x)P \xrightarrow{(\nu x)\theta}_I (\nu x)P'},\ x \notin n(\ell(\theta))$ $\qquad$ $Match:\ \dfrac{P \xrightarrow{\theta}_I P'}{[x=x]\,P \xrightarrow{=_m\theta}_I P'}$

$Com_0:\ \dfrac{P \xrightarrow{\vartheta \overline{x} K}_{\emptyset} P'\ ,\ Q \xrightarrow{\vartheta' x(U)}_{\emptyset} Q'}{P|Q \xrightarrow[\emptyset]{\langle \|_0 \vartheta \overline{x} K, \|_1 \vartheta' x(K)\rangle} P'|Q'\{K/U\}}$ $\qquad$ $Com_1:\ \dfrac{P \xrightarrow{\vartheta' x(U)}_{\emptyset} P'\ ,\ Q \xrightarrow{\vartheta \overline{x} K}_{\emptyset} Q'}{P|Q \xrightarrow[\emptyset]{\langle \|_0 \vartheta' x(K), \|_1 \vartheta \overline{x} K\rangle} P'\{K/U\}|Q'}$

$Ide:\ \dfrac{P\{\tilde{K}/\tilde{U}\} \xrightarrow{\theta}_I P'}{A(\tilde{K}) \xrightarrow{\tilde{m}\theta}_I P'},\ A(\tilde{U})=P$

$Open:\ \dfrac{P \xrightarrow{\vartheta \overline{x} K}_{\emptyset} P'}{(\nu I)P \xrightarrow{\vartheta \overline{x}(K)}_I P'},\ x \notin I \subseteq fn(K)$

$Close_0:\ \dfrac{P \xrightarrow{\vartheta \overline{x}(K)}_I P'\ ,\ Q \xrightarrow{\vartheta' x(U)}_{\emptyset} Q'}{P|Q \xrightarrow[\emptyset]{\langle \|_0 \vartheta \overline{x}(K), \|_1 \vartheta' x(K)\rangle} (\nu I)(P'|Q'\{K/U\})}\ ,\ I \cap fn(Q)=\emptyset$

$Close_1:\ \dfrac{P \xrightarrow{\vartheta' x(U)}_{\emptyset} P'\ ,\ Q \xrightarrow{\vartheta \overline{x}(K)}_I Q'}{P|Q \xrightarrow[\emptyset]{\langle \|_0 \vartheta' x(K), \|_1 \vartheta \overline{x}(K)\rangle} (\nu I)(P'\{K/U\}|Q')}\ ,\ I \cap fn(Q)=\emptyset$

$HO\pi:\ \dfrac{P \xrightarrow{\theta}_I P'}{P \xrightarrow{\theta} P'}$

Table 1: Proved transition system of the HOπ-calculus.

possible transitions between them.[4] In the following we will only consider the transition system associated with a process P, i.e. the fragment of the whole transition system reachable from P. A few auxiliary definitions and notations will be helpful.

Hereafter, we will write a transition $P \xrightarrow{\theta} Q$ only if it is deducible according to the inference rules in Tab. 1; furthermore we will write it simply as θ, when unambiguous.

First we make precise the notion of a derivative of P, i.e. a node reachable from P.

Definition 3.3 *Given a process P, P' is a* derivative *of P if there is a (possible empty) sequence of transitions with source P and target P', i.e.* $P = P_0 \xrightarrow{\theta_1} P_1 \dots P_{n-1} \xrightarrow{\theta_n} P_n = P'$, $n \geq 0$.
The derivative set *of a process P, written $d(P)$, contains all the derivatives of P.*

We can now formalize the notion of transition system associated with a process P.

Definition 3.4 *Given a process P, its* (proved) transition system *is a triple* $\langle d(P), \xrightarrow{\theta}_P, P\rangle$, *where* $\xrightarrow{\theta}_P = \{Q \xrightarrow{\theta} R \mid Q, R \in d(P)\}$.
When unambiguous, we will omit the index P.

As an example of $HO\pi$ specification, we now describe two architectures that run sequential programs. Our two specifications will be used later on to compute a simple performance measure.

Example. We consider a conventional uniprocessor and a prefetch pipeline machine. Assume that their logical components are specified as processes composed through the parallel composition operator. The interactions between the components in the fetch-execute cycle are represented by communications between the corresponding processes, whose behaviour consists of fetching an instruction and then executing it.

As far as the conventional architecture is concerned, the processor PP specified below interacts with a memory [5] M to fetch an instruction j that is sent along the channel $fetch$ to PP. Then M resumes, while PP receives the instruction, executes it and restarts. The specifications of M and PP are given by the following agent definitions

$$M = \overline{fetch}\, j.M \quad \text{and} \quad PP = fetch(x).x.PP$$

We make the channel *fetch* private to M and PP, and we eventually obtain the specification

$$Sys^c = (\nu\, fetch)(M \mid PP).$$

[4] Strictly speaking, it is not a graph because the nodes are countably infinite.

[5] We model the memory as an active component, actually as a one-place buffer.

Note that we are specifying here the behaviour of the conventional machine. Hence we can assume in our abstraction that M outputs the value needed by PP even though in a real machine it is PP that asks what instruction to fetch.

The prefetch pipeline architecture has two modules in place of PP above: an instruction unit IU and an execution unit EU. They should behave as follows. The IU module fetches an instruction from M, passes it to EU and begins again. The EU module receives the instruction, executes it and then restarts.

This kind of architecture improves the performance of conventional processors because fetching instructions can overlap with the time of their execution, provided that communications between IE and EU are faster than those between M and PP. The specification of the system, consisting of IU, EU and M, follows

$$M = \overline{fetch}\, j.M, \quad IU = fetch(x).\overline{pass}\, x.IU \quad \text{and} \quad EU = pass(s).s.EU.$$

The specification of the prefetch pipeline machine is

$$Sys^p = (\nu\, fetch)(M \mid (\nu\, pass)(IU \mid EU)).$$

Figure 1 shows the proved transition systems of our two idealized architectures. For the sake of clarity, in the labels of the systems we omit the restriction of the channels $(\nu\ fetch)$ and $(\nu\ pass)$. □

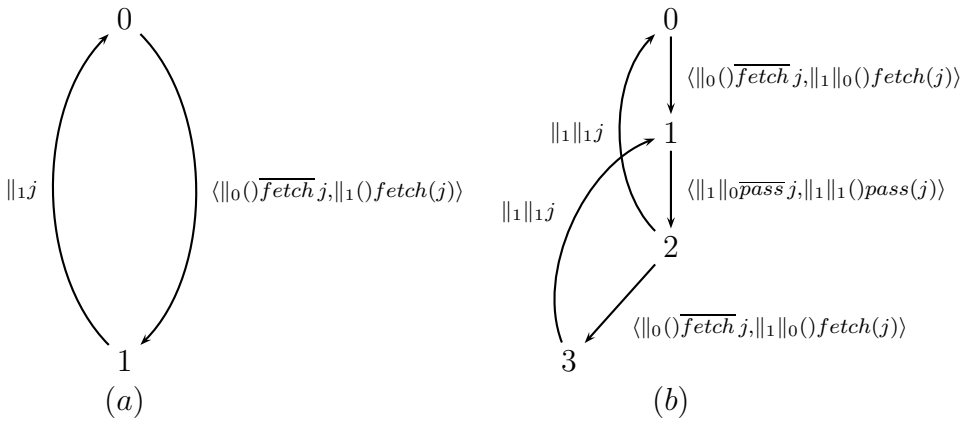

Figure 1: Proved transition system of the conventional (a) and the prefetch pipeline uniprocessor (b).

3.2 Continuous time Markov chains

In this section we briefly recall the relevant portions of the theory of stochastic processes and continuous time Markov chains (CTMC), see [3, 51] for more details.

Definition 3.5
A stochastic process *is a family of random variables* $\{X(t)\ s.t.\ t \in T\}$. *The index set* T *is usually called* time parameter *and* t *is called* time.
The process is continuous time *if* T *is a continuous set.*
The state space *of the process is the set of possible values that* $X(t)$ *can assume.*

Intuitively, $X(t)$ is the state of the process at time t.

Many systems in practice have the property that, once in a given state, the past states have no influence on their future. This is called the *memoryless* or *Markov property* and the stochastic processes satisfying it are called *Markov chains*, if their state space is discrete.

Definition 3.6 *The family of random variables* $\{X(t)\ s.t.\ t \geq 0\}$ *is a* continuous time Markov chain *(CTMC) if for any set of* $n+1$ *values* $t_1 < \ldots < t_{n+1}$ *in the index set, and any set* $\{x_1, \ldots, x_{n-1}, x_i, x_j\}$ [6] *of* $n+1$ *states, we have that the* one step transition probability *is*

$$\begin{aligned} &p(X(t_{n+1}) = x_j \,|\, \{X(t_n) = x_i,\ X(t_{n-1}) = x_{n-1}, \ldots, X(t_1) = x_1\}) \\ &\quad = p(X(t_{n+1}) = x_j \,|\, X(t_n) = x_i) = p_{ij}(n) \end{aligned}$$

where $p(A|B)$ *stands for the conditional probability of* A *given* B.
A Markov chain is time homogeneous *if* $p(X(t_{n+1}) = x_j | X(t_n) = x_i) = p_{ij}$ *does not depend on* n.

Hereafter, we shall only consider time homogeneous CTMCs. As a consequence of time homogeneity we do not consider systems where time-outs are implicitly modelled by implying failure of transitions, i.e. we require that the transitions which are enabled in a given state cannot be disabled by flow of time. Note that many communication networks, distributed systems and production lines can be modelled by CTMC (see [35]).

A continuous time Markov chain C can be conveniently represented as a directed graph whose nodes are the states of C, and whose arcs only connect the states that can be reached by each other.

The rates at which the process jumps from one state to another can be arranged in a square matrix $\mathbf{Q}$, called *generator matrix*. Apart from its diagonal, it is the adjacency matrix of the graph representation of the CTMC considered. The entries of $\mathbf{Q}$ are called *instantaneous transition rates* and

[6] We let here the n^{th} and $(n+1)^{th}$ elements of the set be x_i and x_j instead of the more natural x_n and x_{n+1}. This is to simplify the index notation for the one step transition probability, which we write as usual as p_{ij}, instead of $p_{n(n+1)}$.

are defined by

$$q_{ij} = \begin{cases} \lim_{\tau \to 0} \frac{p(X(t+\tau)=x_j \mid X(t)=x_i)}{\tau} & \text{if } i \neq j \\ -\sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij} & \text{if } i = j \end{cases} \tag{2}$$

The case $i = j$ in the definition (2) ensures that the rows of matrix $\mathbf{Q}$ sum to 0, because $-q_{ii}$ is the sum of the rates of the transitions outgoing from the state x_i. Also, $-q_{ii}$ coincides with the exponential distribution of the holding of state x_i, i.e. the time that the chain spends in the state x_i. It is called the *sojourn time of state* x_i. The relation between the rates in $\mathbf{Q}$ and the one-step transition probability p_{ij} from state x_i to state x_j is $p_{ij} = q_{ij}/-q_{ii}$.

Performance measures of systems make sense over long periods of execution. These measures are then usually derived by exploiting the stationary probability distribution of Markov chains which we recall below.

Definition 3.7 *Let* $\Pi^t(x_i) = p(X(t) = x_i)$ *be the probability that a CTMC is in the state* x_i *at time* t*, and let* $\Pi^0 = (\Pi^0(x_0), \ldots, \Pi^0(x_n))$ *be the* initial distribution *of states* $x_0, x_1, \ldots, x_n$*. Then a CTMC has a* stationary probability distribution $\Pi = (\Pi(x_0), \ldots, \Pi(x_n))$ *if*

$$\Pi\mathbf{Q} = 0 \text{ and } \sum_{i=0}^{n} \Pi(x_i) = 1.$$

Note that all the computations needed above, and the stochastic analysis we will make afterwards, can be performed by standard numerical techniques, exploiting the preferred numerical package available.

Finally, we characterize the Markov chains which always admit a stationary distribution.

Definition 3.8 *A state* x_i *is* positive recurrent *if a Markov chain that visits state* x_i *returns to* x_i *with probability 1 in a finite expected number of steps. A Markov chain is* positive recurrent *if all its states are positive recurrent.*

Note that if x_i is positive recurrent and the chain visits it at least once, then it does so infinitely often.

Definition 3.9 *A Markov chain is* irreducible *if for every state* x_i *there is a path in the chain from* x_i *to* x_i *itself, and also there are paths from* x_i *to every other state.* [7]

[7] The graph of an irreducible Markov chain does not need to be fully connected. In fact, we require connection between nodes via sequences of arcs and *not* via individual arcs. For example, the graph $\langle \{0,1,2,3\}, \{\langle 0,1\rangle, \langle 0,2\rangle, \langle 1,0\rangle, \langle 2,3\rangle, \langle 3,0\rangle\}\rangle$ is irreducible, although it is not fully connected. In terms of transition systems, we require that $\forall P_i, P_j \in d(P)$, P_i is a derivative of P_j.

Later on we will make use of the fact that irreducible Markov chains with finite state space are positive recurrent.

The following theorem [26] gives a sufficient condition for the existence and the uniqueness of the stationary distribution Π of a CTMC. It also says that Π can be computed as the limit of the transient distributions.[8]

Theorem 3.10 *A time homogeneous, irreducible, positive recurrent CTMC has a stationary probability distribution* Π. *Moreover*

$$\lim_{t\to\infty} p\big(X(t) = x_k | X(0) = x_0\big) = \Pi(x_k).$$

4 Stochastic semantics

In this section we show how to extract quantitative information from a proved transition system by transforming it into a CTMC. We do this in two steps: first we introduce a function that assigns costs to individual transitions derived from their labels, alone. Here, we use "cost" to mean any measure that affects quantitative properties of transitions (speed, availability, etc.). In particular, we will compute the costs of performing some low-level operations to devise the rate of the corresponding actual transition. In the next sub-sections we will perform operations on costs to tune a probabilistic distribution with respect to the expected speed of actions. Although we interpret costs as parameters of exponential distributions, our relabelling functions are not intended to manipulate random variables. The intuition is that cost functions define a *single* exponential distribution by subsequent refinements as soon as information on the run-time becomes explicit. Operationally we start with an optimistic selection of an exponential distribution and then, while scanning contexts, we jump to other distributions until the one suitable for the current transition is reached. The cost functions encode this jumping strategy. Once the exponential distributions of transitions have been computed, we make some numerical calculations, possibly by collapsing those arcs that share source and target. We also prove some properties of our construction that help to derive performance measures from the CTMC obtained. In the next two sections we will illustrate our proposal by applying it to some examples.

Hereafter, we will restrict ourselves to processes that generate *finite* state spaces, i.e. which have a finite set of derivatives. [9] Note that this does

[8] The distributions associated with the states of the chains at time t are called transient distributions. If the transient distributions at times t and $t+1$ coincide, we obtain the stationary distribution of the chain, because it will no longer change.

[9] A sufficient condition for a process to be such is that all the agent identifiers occurring in it have a restricted form of definition $A(\tilde{U}) = P$. Namely, A can occur in P only if prefixed by some action and cannot occur within a parallel context.

not mean that the behaviour of such processes is finite, because their transition systems may have loops. Indeed, particular forms of loops are essential to apply the steady state analysis that we carry out in our examples (see the discussion at the beginning of Sect. 4.2.2). Moreover, we will only consider processes that can perform inputs *within* communications and never in isolation from the environment, i.e. an input can only fire if there is a complementary output. [10] The two restrictions above ensure that our processes have a *finite* set of derivatives.

4.1 Cost function

Our first step consists in defining the function which associates a cost with each transition labelled by θ. As we have already seen, this cost represents the *rate* of the transition, i.e. the parameter which identifies the exponential distribution of the duration times of θ.

We assign costs to the labels of the transitions in the proved transition system and *not* to the prefixes, like in the classical approach, in order to give a more accurate description of system behaviour. Indeed our costs depend not only on the action $\alpha = \ell(\theta)$ originating the transition, but also on the operations that the run-time support of the target architecture performs for firing α; likewise if the action originating the transition is an internal action τ_i or a communication. We briefly explain why in this way we gain descriptive power through the following example. Consider the resolution of a choice, made by applying a rule *Sum*. It mimics some real operations on the target architecture, such as checking the ready list or implementing fairness policies. In practice, an action fired after a choice takes more time than the same action occurring deterministically. Also, the implementation of the run-time support makes the duration of choices vary. We wish to account for this performance degradation, so we charge with an extra delay the execution of an action occurring in a nondeterministic sum. The other operations of our calculus reflect other analogous routines of the run-time support that delay the execution of an action as well. Communications deserve a special treatment, discussed below. Summing up, we wish our cost function to take the routines of the run-time support into account.

Since the structural operational semantics of a language specifies its abstract machine, the context in which an action occurs in the program represents the operations that the target machine performs in order to fire that action. In our proved semantics, the part ϑ of an enhanced label $\vartheta\alpha$ represents the context in which the action α occurs. Accordingly, a suitable linearization of ϑ represents the execution of the above mentioned run-time

[10] It is sufficient to consider processes where every occurrence of an input prefix $x(U)$ occurs within a restriction (νx).

support routines on the target machine. Following this intuition, we let the cost of the transition $\vartheta\alpha$ depend on α and on (the context) ϑ.

To define a cost function, we start by considering the execution of the action α on a dedicated architecture that only has to perform α, and we estimate once and for all the corresponding duration with a fixed rate r. Then we model the performance degradation due to the run-time support. In order to do that, we introduce a scaling factor for r in correspondence with each routine called by the implementation of the transition θ under consideration. As a matter of fact, we do not fix the cost function and we only propose for it a possible set of parameters that reflect some features of a somewhat idealized architecture (the number of processors of a net, the bandwidth of a channel, etc.). Clearly our proposal is still very abstract. However it is sufficient to derive performance measures that agree with those presented in the literature [4, 13], as the next section shows.

We proceed now with our definitions. First we assign costs to visible actions $\alpha \in \mathcal{A}$ and to invisible actions $\tau_i \in \mathcal{S}$; communications will be considered afterwards. As discussed above, our cost function returns the parameter of the exponential distribution which describes the time needed to perform the very basic, low-level operations corresponding to one of these actions, regardless of the context in which it occurs. Formally, we use a function

$\$_\mu : \mathcal{A} \cup \mathcal{S} \to \mathbb{R}^+$

defined as

$$\begin{aligned} &\$_\mu(\tau_i) = \lambda_i \\ &\$_\mu(\overline{x}K) = f_{out}(bw(x), size(K)) \\ &\$_\mu(\overline{x}(K)) = f_{bo}(bw(x), size(K)) \\ &\$_\mu(x(K)) = f_{in}(bw(x), size(K)) \end{aligned} \tag{3}$$

The real numbers λ_i represent the cost of executing the routine corresponding to the i^{th} internal action τ_i. The functions f_{out}, f_{bo} and f_{in} define the costs of the routines which implement the send and receive primitives. Function f_{bo} differs from f_{out} because the fresh names need to be checked in the bound output, so its cost incorporates a call to a name generator. For example, if there are n names around and the delay due to calling the name generator is represented by the function $h(n)$, then we can have, e.g., $f_{bo} = f_{out} + h(n)$. Besides the implementation cost due to their own algorithms, the functions above depend on the bandwidth of the communication channel (represented by $bw(x)$) and the size of their objects ($size(K)$). As for the cost of an input action, we note that sending data along a channel is faster than receiving them, hence we assume $f_{in} \leq f_{out}$. We let, however, f_{in} be a parameter supplied by the designer in order not to limit the applicability of the framework. The intuition of input slower than output follows. The

reception of a message involves at least one check for the presence of the data at the site of the sending process, and then the message needs to be copied in some storage location of the reading process. As an example, consider the input of a piece of mobile code: the receiver must set up an execution environment to run the received code. For instance a Java agent which moves needs some class loading at the recipient site and possibly the activation of a new thread of control.

According to the intuition that contexts slow down the speed of actions, we now determine a slowing factor for any construct of the language. The idea is to devise a general framework in which real situations can fit simply by supplying actual parameters (in our case functions defining the slowing factors). To simplify the presentation, we use here a minimal set of parameters that affect the cost of the routines implementing any operator of our language. In doing so, we abstract from actual architectures. A precise instantiation to physical networks can occur in the refinement steps from specification to implementations as long as actual parameters become available.

The cost of the operators is expressed by the function

$$\$_o : \mathcal{L} \cup \mathcal{O} \to (0, 1]$$

$$\begin{aligned}
&\$_o(\|_i) = f_{\|}(np),\ i = 0, 1\\
&\$_o(+_i) = k,\ i = 0, 1\\
&\$_o(=_m) = f_{=}(m)\\
&\$_o((\nu x)) = f_{\nu}(n(P))\\
&\$_o\big(m_1, \ldots, m_k\big) = f_{()}(m_1, \ldots, m_k, np).
\end{aligned}$$

Parallel composition is evaluated according to the number np of processors available. A particular case is $\$_o(\|) = 1$ which arises when there is an unbound number of processors. (Recall that we are not yet considering communications.) We assume that the cost of a choice between two alternatives is constant; note however that n nested choices delay the execution by $n \times k$. The cost of matching depends on the size m of the data to be compared. The cost of restriction depends at least on the number of names $n(P)$ of the process P because its resolution needs a search in a table of names. Finally, the activation of a new agent via a constant invocation has a cost depending on the size and the number of its actual parameters, as well as on the number of processors available.

We now consider communications. To determine their impact on the overall cost function, we follow their deduction scheme. Let the label of the transition in hand be $\vartheta\langle||_0\vartheta_0\alpha_0, ||_1\vartheta_1\alpha_1\rangle$. The two partners perform independently some low-level operations locally to their environment. These operations are recorded in ϑ_0 and ϑ_1, inductively built by the application of the rules that fill in the premises of rule Com. Each of the ϑ_i leads to a delay

in the rate of the corresponding α_i, which we compute through the auxiliary cost function $\$_o$. Then the pairing $\langle ||_0\vartheta_0\alpha_0, ||_1\vartheta_1\alpha_1\rangle$ occurs and corresponds to the actual communication. Finally, there are those operations, recorded in ϑ, that account for the common context of the two partners. Also, the slow down due to this common context is computed using $\$_o$. (Note that *Com* and *Close* rules can be applied at most once in a derivation of a given transition because pairs cannot occur as labels in their premises; see Tab. 1.)

We are left to compute the delay due to a communication, i.e. to the pairing discussed above. Since communication is synchronous and handshaking, we take the minimum of the costs of the operations performed by the participants independently (originated by ϑ_i) to make communications reflect the speed of the slower partner. [11]

Also, we wish to take into account the distance of the locations where the partners run. To do this, we use a function $f_{\langle\rangle} : \mathcal{L}^* \times \mathcal{L}^* \to (0, 1]$, and we encode locations as sequences $\chi_0, \chi_1 \in \{||_0, ||_1\}^*$. To see why, consider the binary abstract syntax tree of a process, when the parallel composition $|$ is the only syntactic operator. Then, a sequence χ can be seen as the access path from the root, i.e. from the whole process, to a leaf, i.e. to a subprocess. An *allocation table* is thus a mapping ρ from (abstract) locations, represented as strings over $\{||_0, ||_1\}^*$, to the set of physical locations available.

So, the two arguments of $f_{\langle\rangle}$, together with allocation tables, can be used to determine where the two communicating processes have been actually placed, and from this their distance. [12]

To apply function $f_{\langle\rangle}$, we need an auxiliary function that extracts the parallel tags from the part ϑ of an enhanced label. It is $\underline{\cdot} : (\mathcal{L} \cup \mathcal{O})^* \to \mathcal{L}^*$, inductively defined as (ϵ is the empty string and $o \in \mathcal{O}$)

$$\underline{\epsilon} = \epsilon, \quad \underline{\|_i\vartheta} = \|_i\underline{\vartheta}, \quad \underline{o\vartheta} = \underline{\vartheta}.$$

We now have all the ingredients to define the function that associates costs with enhanced labels. It is defined by induction on θ and by using the auxiliary functions $\$_\mu$ as basis, and then $\$_o$ and $f_{\langle\rangle}$.

Definition 4.1 *Let the cost function be*
$\$: \Theta \to \mathbb{R}^+$

[11] Recall that the lower the cost, the greater the time needed to complete an action and hence the slower the speed of the transition occurring.

[12] The distance of the sites where the communicating processes reside is only one of the aspects affecting the duration of the remote communication. Actually, a more realistic function $f_{\langle\rangle}$ should have more parameters: for instance the average number of hops that the message needs in the routing, or the kind of communication medium and hence latency, or the kind of network protocols adopted.

defined as

$$\$(\mu) = \$_\mu(\mu)$$
$$\$(o\theta) = \$_o(o) \times \$(\theta)$$
$$\$(\|_i\theta) = \$_o(\|_i) \times \$(\theta)$$
$$\$(\langle \|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1\rangle) = f_{\langle\rangle}(\underline{\|_0\vartheta_0}, \underline{\|_1\vartheta_1}) \times \min\{\$(\|_0\vartheta_0\alpha_0), \$(\|_1\vartheta_1\alpha_1)\}$$

Below we compute the costs of the transitions of our running example introduced at the end of subsection 3.1.

Example (cont'd). Consider the systems depicted in Fig. 1. For the sake of simplicity, in computing the performance of the fetch-execute cycle we assume that the execution of all instructions takes the same time. Furthermore, we assume that both architectures require the same time for fetching, as well as for executing instructions.
Since corresponding operations have the same names in Sys^c and Sys^p, we use a unique cost function for both architectures. So we assign rates to actions as follows

$$\begin{aligned} &\$_\mu(j) = a \\ &\$_\mu(\overline{fetch}\, j) = \$_\mu(fetch(j)) = b \\ &\$_\mu(\overline{pass}\, j) = \$_\mu(pass(j)) = c \end{aligned} \tag{4}$$

Also, assume that the machines we are comparing have to interpret a set of CISC instructions so that they take longer in the execution of an instruction than in its fetch. Hence, we let

$$b \leq a.$$

Nowadays, the run-time support of virtually all architectures is firmware, and it resolves calls to routines through a jump instruction. Assume then that the ones we are considering do the same, and neglect in what follows the cost of invoking identifiers M, PP, IU and EU. Therefore, we let

$$\$_o(0) = 1,$$

(the empty list of the parameter sizes has size 0). In fact, any other cost different from 1 only affects the value of the cost function; on the other hand our comparison of the performance of the traditional and the prefetch architectures is not affected.

In the labels of Fig. 1 the increment of the cost due to the parallel tags is null because each process has its own processing unit. In fact, although the prefetch architecture has two concurrent transitions in its behaviour ($||_1||_1 j$ and $\langle ||_0()\overline{fetch}j, ||_1||_0()fetch(j)\rangle$), it also has two processing units that can execute them independently and simultaneously. Thus we can set

$$\$_o(\|_i) = 1.$$

Finally, we neglect the delay due to the restriction of the channels because hardware connections are physical links which are private by definition and need no check on their owners

$$\$_o(\nu\ fetch) = \$_o(\nu\ pass) = 1.$$

Now we compute the cost of synchronizations. In the prefetch architecture Sys^p, the distance between the two units connected in pipeline (IU and EU) is not significant. Indeed both of them lie on the same platform. Instead, we consider the distance between the instruction unit IU and the memory M of the machine, and between M and PP in Sys^c. Thus, we assume that

$$f_{\langle\rangle}(\|_0, \|_1) = f_{\langle\rangle}(\|_0, \|_1\|_0) = d, \text{ and } f_{\langle\rangle}(\|_1\|_0, \|_1\|_1) = 1.$$

To give an example of the use of function \$, we compute the cost of the first transition in Fig. 1(a).

$$\begin{aligned}
& \$((\nu\ fetch)\langle\|_0()\overline{fetch}\, j, \|_1()fetch(j)\rangle) \\
= \ & \$_o((\nu\ fetch))\$(\langle\|_0()\overline{fetch}\, j, \|_1()fetch(j)\rangle) \\
= \ & f_{\langle\rangle}(\underline{\|_0()}, \underline{\|_1()})\min\{\$_o(\|_0)\$_o(0)\$_\mu(\overline{fetch}\, j), \$_o(\|_1)\$_o(0)\$_\mu(fetch(j))\} \\
= \ & f_{\langle\rangle}(\|_0, \|_1)b = d \times b.
\end{aligned}$$

Analogously, we obtain the costs of all the other transitions

$$\begin{aligned}
& \$(\|_1 j) = \$(\|_1\|_1 j) = a \\
& \$((\nu\ fetch,\ pass)\langle\|_0()\overline{fetch}\, j, \|_1\|_0()fetch(j)\rangle) = d \times b \\
& \$((\nu\ fetch,\ pass)\langle\|_1\|_0\overline{pass}\, j, \|_1\|_1()pass(j)\rangle) = c.
\end{aligned}$$

□

We now outline how the cost function defined above can be used for languages in other paradigms, such as the imperative, functional and object-oriented. One way is to encode these languages into a process calculus, e.g. an imperative language into CCS [45], or a functional or an object-oriented one into the π-calculus [46]. Alternatively, we can give the language in hand a proved semantics, and then follow our proposal. Since many imperative languages have an SOS semantics (including Java [14]), we only need to enhance the labels with tags for recording the application of the rules in the derivations. Of course, the definition of a sensible cost function entails accurately linking tags with the low-level operations of the run-time support of the language, as discussed in Sect. 2.

4.2 Stochastic analysis

Now we turn the transition system of a process into a CTMC for computing performance measures.

4.2.1 Some auxiliary results

We start by deriving some transition probabilities. In order to associate a parameter r with a transition, we transform the nondeterministic choices present in the specifications into probabilistic ones. This is because to measure performance we must estimate the relative frequency of a transition of a non deterministic choice. So we assign to each alternative a probability of occurrence. All the activities enabled attempt to proceed, but only the fastest one succeeds. Note that the fastest activity is different on subsequent attempts because durations are expressed by random variables drawn from an exponential distribution. Also, the probability of two activities ending simultaneously is 0, by definition of continuous probabilistic distributions.

Recall that the exponential distributions that we use enjoy the *memoryless property*. Roughly speaking, the occurrence of the next transition is independent of when the last transition occurred. In other words, how long the transition will wait before completing is independent of how long it has already waited. This means that any time a transition becomes enabled, it restarts its elapsing time as if it were the first time that it was enabled. Also, all transitions are assumed to be *time homogeneous*, meaning that the rate of a transition is independent of the time at which it occurs.

Hereafter, we define the rate at which a system changes from behaving like process P_i to behaving like P_j.

Definition 4.2 *The* transition rate *between two states P_i and P_j, written $q(P_i, P_j)$, is the rate at which the transitions between P_i and P_j occur*

$$q(P_i, P_j) = \sum_{P_i \xrightarrow{\theta_k} P_j} \$(\theta_k).$$

Note that more than one transition may be involved in the computation of $q(P_i, P_j)$. For example, the costs of both transitions of $a + a$ are summed to obtain $q(a+a, \mathbf{0})$. Clearly if P_j is not a one-step derivative of P_i, $q(P_i, P_j) = 0$.

Now we show how to attach probabilities to the transitions exiting from the same node, i.e. the probability that a process makes a transition θ leading to state P_j from state P_i. It is the ratio between the rate of θ and the sum of the rates of all the transitions leaving P_i. Formally,

Definition 4.3 *The* occurrence probability *of a transition from P_i to P_j is*

$$p_{ij} = \frac{\sum_{P_i \xrightarrow{\theta_k} P_j} \$(\theta_k)}{\sum_{P_i \xrightarrow{\theta_h} Q} \$(\theta_h)}.$$

Recall that in the transition system of a process P there is a node for each derivative of P and an arc for each possible transition. To form the CTMC, a state is associated with each node of the transition system, while the transitions between states are defined by the arcs, possibly coalescing those sharing source and target (as an example, consider again the process $a + a$ discussed above). Recall that we assumed our process to be such that the set of its derivatives is finite.

The quantity $q(P_i, P_j)$, or simply q_{ij} are the off-diagonal elements of the generator matrix of the CTMC, namely $\mathbf{Q}$ (see Eq. (2) in Sect. 3). Recall that CTMC can be seen as a directed graph and that its matrix $\mathbf{Q}$ (apart from its diagonal) represents its adjacency matrix. Hence, hereafter we will use indistinguishably $CTMC$ and its corresponding $\mathbf{Q}$ to denote a Markov chain. More formally, we have the following definition.

Definition 4.4
Given a process P and its transition system $\langle d(P), \stackrel{\theta}{\longrightarrow}_P, P\rangle$, let P_i, P_j belong to $d(P)$, the space state of P, with cardinality n. Then, the generator matrix $\mathbf{Q}$ of the continuous time Markov chain of P (called $CTMC(P)$) is a $[n \times n]$ square matrix with elements q_{ij} defined as

$$q_{ij} = \begin{cases} q(P_i, P_j) = \displaystyle\sum_{P_i \stackrel{\theta_k}{\longrightarrow} P_j} \$(\theta_k) & \text{if } i \neq j \\ \\ -\displaystyle\sum_{\substack{j=1 \\ j \neq i}}^{n} q_{ij} & \text{if } i = j \end{cases} \tag{5}$$

Note that two nodes of the graph representing $CTMC(P)$ are connected by at most one arc, while in the transition system of P two states can be connected by more than one transition, as mentioned above.

Equation (5) defines the instantaneous transition rate from P_i to P_j in terms of the transitions outgoing from P_i. Thus, we can define directly in SOS style the graph CTMC associated with a process. More precisely, we define a stratified transition system whose transition relation $\longrightarrow_M$ of the CTMC is defined in terms of the relation $\longrightarrow$ of the transition system.

Definition 4.5 *Let $\langle d(P), \stackrel{\theta}{\longrightarrow}_P, P\rangle$ be the transition system associated with a process P. Then we define the continuous time Markov chain $CTMC(P)$ of P as the labelled transition system $\langle d(P), \stackrel{r}{\longrightarrow}_M, P\rangle$ with $r \in \mathbb{R}^+$, whose transition relation $\longrightarrow_M$ is the minimal relation defined by the rule*

$$CTMC : \frac{P_i \stackrel{\theta_k}{\longrightarrow}_P P_j}{P_i \stackrel{q_{ij}}{\longrightarrow}_M P_j},$$

q_{ij} being defined according to Eq. (5).

We end this subsection by showing that the definitions above are correct, in the sense that they coincide (apart from the diagonal) and that $CTMC(P)$ is actually a continuous time Markov chain. The first fact holds trivially. To see that also the second holds, recall that all transition durations are exponentially distributed. Thus, the total transition rate between two states will be the sum of the rates of the transitions connecting the corresponding nodes in the transition system, as stated below.

Theorem 4.6 *For any finite process P, let $X(t)$ be its corresponding stochastic process. Then $X(t)$ is a continuous time Markov chain with state space $d(P)$, coincident with $CTMC(P)$.*
Furthermore, the sojourn time in a state $X(t_i) = P_j$ is an exponentially distributed random variable, the parameter of which is given by the sum of the rates of the transitions enabled in P_j.

Proof. For an arbitrary process P with an associated stochastic process $X(t)$, consider the *sojourn time* in an arbitrary state $X(t_i) = P_{j_i}$. Let $S_i(t)$ denote the sojourn time distribution, i.e. the probability that a sojourn in the state P_{j_i} has a duration less than or equal to t. Let the transitions enabled in P_{j_i} be $\{\theta_1, \theta_2, \ldots, \theta_n\}$. For each of them, say θ, define $S_{i\theta}(t)$ to be the *conditional sojourn time distribution*, i.e. $S_{i\theta}(t)$ is the probability that a sojourn in the state P_{j_i} has a duration less than or equal to t and ends by the completion of the transition θ. Note that the unconditional sojourn distribution is the sum of the conditional sojourn distributions

$$S_i(t) = \sum_{P_{j_i} \xrightarrow{\theta_k} P_k} S_{i\theta_k}(t).$$

Let the duration of each transition θ_k be exponentially distributed with parameter $r_k = \$(\theta_k)$. Thus the distribution function of the duration of the transition θ_k is $F_k(t) = 1 - e^{-r_k t}$, and has $f_k(t) = r_k e^{-r_k t}$ as its density function. [13] Therefore, for each transition θ_k we have

$$\begin{aligned}
S_{i\theta_k}(t) &= \int_0^t \left(\prod_{\substack{1 \le h \le n \\ h \neq k}} \Big(1 - F_h(x)\Big) \right) dF_k \\
&= \int_0^t \left(\prod_{\substack{1 \le h \le n \\ h \neq k}} \Big(1 - F_h(x)\Big) \right) f_k(x) dx \\
&= \int_0^t \left(\prod_{\substack{1 \le h \le n \\ h \neq k}} \Big(e^{-r_h x}\Big) \right) r_k e^{-r_k x} dx \\
&= r_k \int_0^t \prod_{1 \le h \le n} e^{-r_h x} dx \\
&= r_k \int_0^t e^{-\Sigma x} dx = \frac{r_k}{\Sigma}\Big(1 - e^{-\Sigma t}\Big)
\end{aligned}$$

where $\Sigma = \sum_{h=1}^{n} r_h$.

[13] Given a distribution F, its density function f is such that $F(\overline{x}) = \int_{-\infty}^{\overline{x}} f(t)dt$.

Hence

$$\begin{aligned} S_i(t) &= \sum_{\theta_k \in P_i \xrightarrow{\theta_h} Q} S_{i\theta_k}(t) = \sum_{k=1}^{n} S_{i\theta_k}(t) \\ &= \sum_{k=1}^{n} \frac{r_k}{\Sigma}\left(1 - e^{-\Sigma t}\right) = \frac{1 - e^{-\Sigma t}}{\Sigma} \sum_{k=1}^{n} r_k = 1 - e^{-\Sigma t}, \end{aligned}$$

expresses that the sojourn time in the state P_{j_i} is exponentially distributed with mean $1/\Sigma$, and is independent of the history leading to P_{j_i}. Therefore, the *expected* sojourn time is $\left(\sum_{P_i \xrightarrow{\theta_k} Q} S_{i\theta_k}(t) \right)^{-1}$. As a consequence, the stochastic process associated with the transition system of P is a CTMC with state space $d(P)$ because the time needed by the system to complete a transition with source P_{j_i} is independent of the time elapsed since the system reached P_{j_i}. □

4.2.2 Evaluating the performance

In order to evaluate the performance of a process P, we need to compute the (unique) stationary distribution for the CTMC we associate with it. To have a unique stationary distribution according to the hypothesis of Theorem 3.10, $CTMC(P)$ must be time homogeneous and irreducible. The first condition is clearly satisfied. The second condition is fulfilled if all the states of $CTMC(P)$ are positive recurrent. Since a derivative Q of P corresponds to a state of $CTMC(P)$, it suffices then that Q is reachable by any of its derivatives through a finite sequence of transitions, because we only consider finite state processes. So we will restrict our attention to processes in this form.

We measure the performance of a process P by associating a *reward structure* with it following [38, 36, 16]. Since our underlying performance model is a continuous time Markov chain, the reward structure is simply a function that associates a value with any derivative of P. For instance, when calculating the utilisation of a resource, we assign value 1 to any state in which the use of the resource is enabled (i.e. the state considered is the source of a transition that uses the resource). All the other states earn the value 0. Another classical measure of performance is the throughput of a system, i.e. the amount of useful work accomplished per unit time. For instance, the throughput of a processor is obtained by associating a reward with any state P that enables the execution of an instruction. Since the transitions outgoing from P and describing such an execution have rates r_h expressing the probability and the speed of execution, a reasonable choice for the reward is the sum of all the r_h.

Since rewards are associated with states while process algebras are transition-based, we define below how to compute rewards of states from rates of transitions. We follow the above discussion and assume as given a *transition reward* ρ_θ associated with any transition θ, in accordance with the measures of interest (utilisation, throughput, etc.).

Definition 4.7 *Given a function ρ_θ associating a transition reward with each transition θ in a transition system, the* reward *of a state P is*

$$\rho_P = \sum_{P \xrightarrow{\theta} Q} \rho_\theta.$$

Now, the reward structure of a process P is a vector of rewards with as many elements as the number of derivatives of P. From it and from the stationary distribution Π of $CTMC(P)$ we can compute performance measures. For instance, we obtain the utilisation of a resource by summing the values of Π multiplied by the corresponding reward structure. This amounts to considering the time spent in the states in which the usage of the resource is enabled. Likewise for computing throughputs. The above combination of the stationary distribution and rewards is usually called *total reward* and is defined below.

Definition 4.8 *Let Π be the stationary distribution of $CTMC(P)$. The* total reward *of a process P is computed as*

$$R(P) = \sum_{P_i \in d(P)} \rho_{P_i} \times \Pi(P_i).$$

Now we apply our method to our running example of the uniprocessor and the prefetch pipeline architectures.

Example (cont'd). Consider again the transition systems in Fig. 1 which are both finite and have cyclic initial states. Hence the associated CTMCs are irreducible and, by Theorem 3.10, they have stationary distributions. We derive the generator matrices $\mathbf{Q}^c$ and $\mathbf{Q}^p$ of $CTMC(Sys^c)$ and $CTMC(Sys^p)$. The stationary distributions of the Markov chains for Sys^c and Sys^p are the solutions of the following systems of linear equations

$$\begin{cases} (d \times b)x_0 - ax_1 = 0 \\ x_0 + x_1 = 1 \end{cases} \quad \text{and} \quad \begin{cases} -(d \times b)x_0 + ax_2 = 0 \\ (d \times b)x_0 - cx_1 + ax_3 = 0 \\ cx_1 - (a + (d \times b))x_2 = 0 \\ (d \times b)x_2 - ax_3 = 0 \\ x_0 + x_1 + x_2 + x_3 = 1. \end{cases}$$

The unique solution of the system on the left gives the stationary distribution of the Markov chain $CTMC(Sys^c)$:

$$\Pi^c = \left(\frac{a}{a + d \times b}, \frac{d \times b}{a + d \times b} \right)$$

while the unique solution of the system on the right is the stationary distribution of the Markov chain $CTMC(Sys^p)$:

$$\Pi^p = \left(\frac{a^2 \times c}{A} , \frac{a \times (d \times b)^2 + a^2 \times d \times b}{A} , \frac{a \times d \times b \times c}{A} , \frac{(d \times b)^2 \times c}{A} \right),$$

where $A = a^2 \times c + (d \times b) \times ((d \times b) + a) \times (a + c)$.

In order to compare the relative efficiency of the two architectures, we compare the throughput of Sys^c and Sys^p, i.e. the number of instructions executed per time unit. As mentioned above, only the transitions representing the execution of an instruction will receive a non-zero transition reward. Also, it is natural to set its value equal to the rate of the transition considered. So, the transition reward of the transition $||_1 j$ in Fig. 1(a) and that of the two transitions $||_1||_1 j$ in Fig. 1(b) is a. Instead all the other transitions receive 0 as their transition reward. Now, it is easy to compute the reward of a state, as the sum of the transitions rewards of the arcs outgoing from it. In the case of the conventional architecture, this amounts to

$$\rho_0 = 0 \text{ and } \rho_1 = a$$

and in the case of the prefetch one

$$\rho_0 = 0,\ \rho_1 = 0,\ \rho_2 = a \text{ and } \rho_3 = a$$

The throughputs of the execution of an instruction are therefore

$$T(Sys^c) = \rho_0 \Pi^c(0) + \rho_1 \Pi^c(1) = a\Pi^c(1),$$

and

$$T(Sys^p) = \sum_{i=0}^{3} \rho_i \Pi^p(i) = a\Pi^p(2) + a\Pi^p(3).$$

The prefetch architecture performs better if its throughput is higher than the one of the conventional machine. Thus we study the equation

$$\Pi^p(2) + \Pi^p(3) \geq \Pi^c(0). \tag{6}$$

Assume that a, b and c are defined as in equation (4) and they are all positive, and recall that a, $d \times b$ and c measure the time of executing and of fetching an instruction on channels $fetch$ and $pass$, respectively. Also, recall that the cost of an action is the parameter of the exponential distribution of its execution time: the greater the parameter, the faster the corresponding action. Then, the inequality (6) is reduced to

$$c \geq a + d \times b. \tag{7}$$

which indicates when the throughput, i.e. the performance, of the prefetch architecture, is higher than the one of the conventional machine. As expected, this happens when the speed at which its instruction unit sends a fetched instruction to its execution units is faster than fetching and executing an instruction in the conventional machine. □

Now, we summarize what a designer needs to know to use our proposal. Obviously, the relevant features of the architecture running the application under analysis are needed to instantiate the parameters of our model, e.g. the functions f_{out} and f_{bo} used in the definition of the cost function \$. Furthermore, the designer must provide the rates of the actions performed on dedicated computational units. This is all to build the generator matrix associated with the specification of the application. Finally, to define a reward structure for the measure of interest (e.g. throughput, utilisation) the designer has to assign rewards to transitions, following, for example, the suggestions in [16].

In the next sections we apply our method to two case studies taken from the literature. They involve a network management system and a polling system. We conclude this section with a brief discussion on how a transition affects the CPU load of a given node, thus showing that also other measures of performance can be derived in our framework.

In the presence of process migration and network applications CPU load analysis may help in balancing the load of the nodes; for multiprocessor based systems it helps in balancing the load of computational units. In fact, even a small amount of information leads to significant improvements in performance. Good indices for a node are, for example, the number of active processes on it or the length of the process queue [25, 49]. These indices can be easily retrieved from our model. Consider a transition $P \xrightarrow{\theta} P'$ (suppose it is not a communication; the tuning needed for communications is obvious). As mentioned in Sect. 4.1, the tags $||_i$ within θ single out the sequential component of P that actually fired the transition, and, via the allocation table ρ, they give us the physical node n that fires θ. [14] Consider now the set of transitions exiting from P, and let T be the set of those fired from n. Now, $|T|$, the cardinality of T, is the load index of n before firing θ. Likewise, get the set of transitions exiting from P', say T'. The ratio $|T'|/|T|$ measures the load variation, and expresses the way θ affects the load of the node n. A very simple example follows, which also shows that the enhanced labels and hence the proved operational semantics are an essential tool to discover the sequential components of systems that are active in a

[14] Note that the enhanced labels and hence our proved operational semantics are essential to enable us to discover which sequential components of systems are active in a given state, by only looking at the labels of transitions.

given state by only looking at the transition labels.

Example. Consider again the process in (1):

$$P = x(U).(U|U)|(\nu\, a)\overline{x}Q,$$

and allocate $x(U).(U|U)$ and $(\nu\, a)\overline{x}Q$ on two different physical nodes n and n'. Initially the load of both nodes is 1. To study how the communication of Q along the link x affects the load of n' we count the active sub-processes (the ones which may fire a transition) in n' after the communication, i.e. in the state $(\nu\, a)(\mathbf{0}|(Q|Q))$. The two possible transitions in T are labelled $||_1\langle||_0 +_0 \overline{a}b, ||_1 +_1 ab\rangle$ and $||_1\langle||_0 +_1 ab, ||_1 +_0 \overline{a}b\rangle$. Assume that the two copies of Q are both spawned on the same physical node n', so $\rho(||_1||_0) = \rho(||_1||_1) = n'$. From the labels of the two transitions we can then determine that there are two sub-processes active on n': the load index of n' is 2. Therefore, the initial communication doubles the load of n'. □

5 Network management

We consider here the network management application analysed in [4]. It uses the *Simple Network Management Protocol* based on the *Internet Protocol* for the maintenance of the network. This protocol is based on a client-server paradigm. The client *NMS* is a *network management station* that interacts with servers placed on the network devices to monitor and maintain the network. Each server handles a management information base (MIB) with the relevant parameters for the device. In order to perform all the computations related to the management, the *NMS* polls the servers periodically. The client interacts with the servers through remote procedure calls. The operations invoked are typically `get` and `set` for retrieving and updating values in the data bases of the servers. This kind of interaction between an *NMS* and the servers is called *micro-management* and it usually generates a huge amount of network traffic.

Migration of code can reduce the traffic by sending to the servers a piece of code that groups the calls altogether. This approach is known as *management by delegation*, and our specification will follow a remote evaluation paradigm [29].

We specify below the two implementations outlined above, and we keep the same assumptions as [4]. More precisely, if a chunk of data of size X is to be transmitted at the ISO/OSI application level, we represent the actual amount of data exchanged at the network layer as

$$\eta(X) \times X \quad \text{with} \quad \eta(X) = \alpha(X)/X + \beta(X),\ \eta(X) > 1$$

where $\alpha(X)$ accounts for the control information exchanged in a connection oriented protocol, and $\beta(X)$ accounts for the overhead introduced by message encapsulation.

Still following [4], we assume that the interactions between the *NMS* and each of the N network devices (servers) have the same format. So we only specify a single interaction, which consists of Q remote procedure calls to each device D. The extensions needed to cover the general case are straightforward: just put in parallel $N-1$ further copies of D. The operations that an *NMS* requires from network devices will be denoted as i_h, $h = 1, .., Q$, while the answers of the devices will be j_k, $k = 1, .., Q$.

5.1 Formal specifications

We formally describe the network management application in HOπ. We first use the client-server paradigm, and then the one based on remote evaluations. [15] We will only compute the network traffic of the interactions between *NMS* and D once a connection with a given device has been established. Actually, the selection of D imposes the same delays in the two paradigms; so, having specified the interactions of *NMS* with a single D is indeed harmless.

5.1.1 The client-server specification $NetMng^{CS}$

In the client-server paradigm (CS for short), the server usually offers some services to its clients, which may be located at different sites. A client requests a service by sending a message to the server, which performs the service requested. Usually, the service produces a result which is delivered back to the client.

In the specification of the client-server implementation of the network management system, we omit the actual update operations, because they are performed locally at a network unit. Therefore, its duration does not affect the network traffic. Our first HOπ process is

$$NetMng^{CS} = (\nu a)\ (NMS|D)$$

where

$$NMS = \overline{a}\, i_1.\, a\,(x_1).\, \overline{a}\, i_2.\, a\,(x_2).\, \cdots .\, \overline{a}\, i_Q.\, a\,(x_Q).\, NMS$$
$$D = a\,(y_1).\, \overline{a}\, j_1 . a\,(y_2).\, \overline{a}\, j_2 . \cdots .a\,(y_Q).\, \overline{a}\, j_Q .\, D$$

and the channel a is made private via the operator (νa).

[15] We consider here only two out of the four paradigms presented in [4], just to illustrate how to apply our proposal.

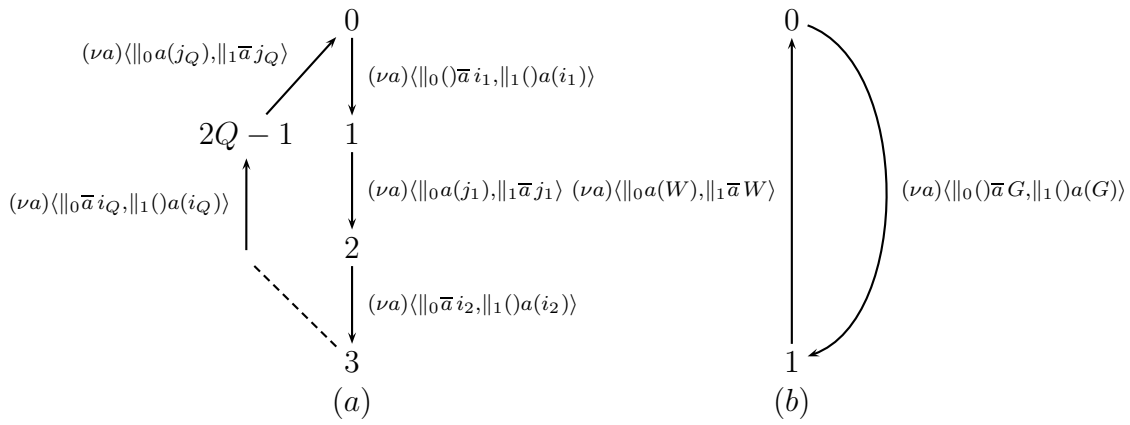

Figure 2: Proved transition systems of $NetMng^{CS}$ (a) and of $NetMng^{REV}$ (b).

The behaviour of $NetMng^{CS}$ is made up of $2Q$ sequential interactions between *NMS* and D. The typical interaction is as follows. *NMS* asks D for i_i and waits for an answer it receives by firing the input $a(x_i)$. The device D waits for a question via $a(y_i)$, and then sends the answer j_i. Now the system is ready to repeat the same interaction in the $(i+1)$-th step. The corresponding proved transition system is depicted in Fig. 2 (a).

5.1.2 The remote evaluation specification $NetMng^{REV}$

In the remote evaluation paradigm (REV for short), a process sends a piece of code to a remote site where it is executed. The process which receives the code eventually sends the results back to the first process.

In the REV paradigm, the network management station *NMS* sends a piece of code to a device D to query *locally* the data base. A single message suffices to send back the Q answers to *NMS*; this cluster of answers is denoted by W below. The specification is

$$NetMng^{REV} = (\nu a)\ (NMS|D)$$

where

$$NMS = \overline{a}\, G.\, a\,(x).\, NMS$$
$$D = a\,(X).\, \overline{a}\, W.\, D$$

The corresponding proved transition system is in Fig. 2 (b).

We conclude this subsection noting that the higher-order facility of the HOπ is only necessary for the formal description of the REV model. In the

case of the CS, the first-order π-calculus, or even CCS with value passing suffice, as no code is sent along the net. Of course, using the same specification language helps comparing the two paradigms.

5.2 Cost functions for $NetMng^{CS}$ and $NetMng^{REV}$

Following our method, we now define the cost functions for our two specifications of the network management applications. For the sake of simplicity we define a unique function \$ for both systems; we can do this because the names and their intuition are the same in the CS and in the REV specifications. Following [4], we assume that requests i_h, answers j_k and code G have an average size of I, R and C, respectively. The output costs are then [16]

$$\$_\mu(\overline{a}\, i_k) = \frac{s}{\eta_{CS}(I) \times I}, \qquad \$_\mu(\overline{a}\, r_k) = \frac{s}{\eta_{CS}(R) \times R},$$
$$\$_\mu(\overline{a}\, G) = \frac{s}{\eta_{REV}(C) \times C}, \qquad \$_\mu(\overline{a}\, W) = \frac{s}{\eta_{REV}(Q \times R) \times Q \times R}$$

where s is the cost of sending one bit along the channel a, and $\eta_{CS}(I)$, $\eta_{CS}(R)$, $\eta_{REV}(C)$ and $\eta_{REV}(Q \times R)$ are the overheads of a single answer, of a single request, of the piece of code and of the Q answers sent back all together, respectively. Also, we assume that the duration of an input depends on and is greater than or equal to that of the corresponding output

$$\$_\mu(a(x)) \leq \$_\mu(\overline{a}\, x).$$

As we will see, there is no need for an actual definition of $\$_\mu(a(x))$: the inequality above suffices in the following calculations. Restriction and identifier invocation slow down the systems, while the parallel composition does not, as there is a processing unit available for each process. Then, we let

$$\$_o((\nu a)) = 1/n$$

be the cost of the restriction of name a assuming that there are n names in the system, and

$$\$_o(0) = 1/p,\ p \geq 1 \quad \text{and} \quad \$_o(\|_i) = 1$$

be the costs of the invocation of identifiers with no parameters, and of parallel composition, respectively. Finally we assume

$$f_{\langle\rangle}(\|_0, \|_1) = 1/d,$$

where $d \geq 1$ takes into account the increment of duration of communications due to the distance between the network management station *NMS* and the selected device D.

[16] We are modelling here the costs of transitions by quantities at the denominator to simplify the comparison of our results with the ones in [4].

5.3 Stationary distribution

The proved transition systems of $NetMng^{CS}$ and of $NetMng^{REV}$ (in Fig. 2) have a finite number of states, and their initial states are cyclic: thus the CTMCs we associate with them are irreducible. To construct the Markov chains, we associate a cost with each transition through the function \$ defined above, according to Sect. 4.1. Then, we calculate their stationary distributions, Π^{CS} and Π^{REV}, respectively. Theorem 3.10 and the initial state being cyclic ensure their existence.

5.3.1 Stationary distribution for $NetMng^{CS}$

The cost of the first transition in $NetMng^{CS}$ is as follows. Hereafter, let $\theta_{i,j}$ be the transition from the state i to the state j.

$$\begin{aligned}
\$(\theta_{0,1}) &= \$((\nu a)\langle \|_0 () \,\overline{a}\, i_1, \|_1 () \, a(i_1)\rangle) \\
&= \$_o((\nu a)) \times f_{\langle\rangle}(\|_0, \|_1) \times \min\{\$(\|_0 () \,\overline{a}\, i_1) \,,\, \$(\|_1 () \, a(i_1))\} \\
&= \frac{1}{n \times d} \times \min\left\{ \frac{s}{p \times \eta_{CS}(I) \times I} \,,\, \frac{s}{k \times p \times \eta_{CS}(I) \times I} \right\} \\
&= \frac{s}{n \times d \times k \times p \times \eta_{CS}(I) \times I} \,.
\end{aligned}$$

This is also the cost of the other $Q - 1$ transitions performed by *NMS* to send a request to D, so

$$\$(\theta_{i,i+1}) = \frac{s}{n \times d \times k \times p \times \eta_{CS}(I) \times I}, \qquad i = 0, 2, 4, \ldots, 2Q - 2.$$

Likewise, for all the transitions in which an answer is communicated, we have

$$\$(\theta_{i,i+1}) = \frac{s}{n \times d \times k \times \eta_{CS}(R) \times R}, \qquad i = 1, 3, 5, \ldots, 2Q - 1.$$

It is now easy to derive $CTMC(NetMng^{CS})$, its generator matrix $\mathbf{Q}^{CS}$ and its stationary distribution Π^{CS}, which has the following components

$$\begin{aligned}
\Pi^{CS}(2i) &= \frac{p \times \eta_{CS}(I) \times I}{Q \times (p \times \eta_{CS}(I) \times I + \eta_{CS}(R) \times R)} \\
\Pi^{CS}(2i+1) &= \frac{\eta_{CS}(R) \times R}{Q \times (p \times \eta_{CS}(I) \times I + \eta_{CS}(R) \times R)} \qquad i = 0, \ldots, Q - 1.
\end{aligned}$$

5.3.2 Stationary distribution for $NetMng^{REV}$

We repeat the above for $NetMng^{REV}$. The rates associated with its transitions are

$$\$(\theta_{0,1}) = \$((\nu a)\langle \|_0 () \,\overline{a}\, G, \|_1 () \, a(G)\rangle) \;=\; \frac{s}{n \times d \times k \times p \times \eta_{REV}(C) \times C}$$

$$\$(\theta_{1,0}) = \$((\nu a)\langle \|_0 \, a(W), \|_1 \, \overline{a}\, W\rangle) \;=\; \frac{s}{n \times d \times k \times \eta_{REV}(Q \times R) \times Q \times R}\,.$$

These costs fully determine $CTMC(NetMng^{REV})$, $\mathbf{Q}^{REV}$ and Π^{REV}. In particular, we have

$$\Pi^{REV} = \left(\frac{p \times \eta_{REV}(C) \times C}{p \times \eta_{REV}(C) \times C + \eta_{REV}(Q \times R) \times Q \times R}, \right.$$
$$\left. \frac{\eta_{REV}(Q \times R) \times Q \times R}{p \times \eta_{REV}(C) \times C + \eta_{REV}(Q \times R) \times Q \times R} \right).$$

5.4 Comparison of $NetMng^{REV}$ and $NetMng^{CS}$

In this sub-section we compare the performance of $NetMng^{CS}$ and $NetMng^{REV}$ by relating their throughputs. As we have already seen, the throughput for a given activity is found by first associating a transition reward equal to the activity rate with each transition, and then by rewarding a state with the sum of the rewards of the transitions outgoing from it.

In both systems each transition is fired only once during an interaction browser-data manager. Also, the graphs of both CTMCs are cyclic and all the labels represent different activities. This amounts to saying that the throughputs of all the activities are the same, and we can freely choose one of them to compute the throughput of $NetMng^{CS}$ and $NetMng^{REV}$. Thus we associate a transition reward equal to its rate with the last communication and a null transition reward with all other communications. From them, we compute the reward structure of $NetMng^{CS}$:

$$\rho_{2Q-1} = \frac{s}{n \times d \times k \times \eta_{CS}(R) \times R} \text{ and } \rho_i = 0, i = 0, .., 2Q-2.$$

Similarly for $NetMng^{REV}$:

$$\rho_0 = 0 \qquad \text{and} \qquad \rho_1 = \frac{s}{n \times d \times k \times \eta_{REV}(Q \times R) \times Q \times R}.$$

The throughput of our systems is then

$$T(NetMng^{CS}) = \sum_{i=0}^{2Q-1} \rho_i \Pi^{CS}(i) = \rho_{2Q-1} \Pi^{CS}(2Q-1) =$$
$$\frac{s}{n \times d \times k \times Q \times (p \times \eta_{CS}(I) \times I + \eta_{CS}(R) \times R)}$$

and

$$T(NetMng^{REV}) = \rho_0 \Pi^{REV}(0) + \rho_1 \Pi^{REV}(1) = \frac{s}{n \times d \times k \times (p \times \eta_{REV}(C) \times C + \eta_{REV}(Q \times R) \times Q \times R)}$$

To discover when $NetMng^{REV}$ is better than $NetMng^{CS}$ we solve the following inequality

$$T(NetMng^{REV}) > T(NetMng^{CS})$$

from which

$$\eta_{REV} \times C < \frac{Q \times \eta_{CS}(R) \times R - \eta_{REV}(Q \times R) \times Q \times R}{p} + Q \times \eta_{CS}(I) \times I.$$

Usually a fixed overhead is associated with each packet exchanged over the network. Thus, the longer the message being segmented, the smaller the relative overhead because the fragmentation of packets is reduced. In other words it is likely that

$$Q \times \eta_{CS}(R) \times R \geq \eta_{REV}(Q \times R) \times Q \times .R$$

Hence, if several answers can be transmitted together in a single message, the overhead is going to become smaller, though it will depend on the protocols used to implement communications in CS and REV.

The analysis above suggests using a REV specification when the size of the filtering code sent along with the requests is smaller than the difference between the following two quantities. The first is the total size of all the requests. The second quantity is the overhead in the transmission of the answer divided by the duration p of the invocation of an identifier.

We finally note that, letting $p = 1$ our results coincide with those in [4], where the time consumption of "calling an agent" is not considered.

6 A revised polling system

In this section we present a larger example taken from the literature to show how our approach scales up. We consider a multi-server multi-queue system (MSMQ, for short) which is an extension of a classical polling system to include more than one server. This system has already been studied in the settings of stochastic process algebras [36] and of stochastic Petri nets [44, 39]. We consider here a little variant of [36], which stresses mobility issues. In our version the routing of servers is state-dependent instead of state-independent as in the original presentation. In particular, servers do not move randomly between nodes, but consider whether a node already has a server and whether it has something to perform. In this respect, our specification is not a proper

MSMQ system, but it enables us to cope with the general problem of remote servers dispatching agents to clients.

We consider a system made up of two independent servers that are routed between two nodes, each with a single place buffer. Two nodes suffice for generating the smallest configuration on which we can comfortably illustrate the features of our model; having more nodes would only make the example longer. We assume that a customer occupies a place in the node until service is completed. We rely on $HO\pi$ to model the routing of servers to nodes via process migration. The specification of the polling system follows (assume $i = 1, 2$).

$$
\begin{aligned}
&P = (\nu\, x_i, r_i, s_i, p_i)((N_1 \mid N_2)|(S \mid S)) \\
&N_i = x_i(U).U \mid Node_i \\
&Node_i = in_i.s_i.Node_i + p_i.Node_i \\
&S = \overline{x}_1\langle S_1^1\rangle.r_1.S + \overline{x}_2\langle S_2^1\rangle.r_2.S \\
&S_i^1 = \overline{s}_i.\overline{r}_i.x_i(U).U + \overline{p}_i.\overline{r}_i.x_i(U).U.
\end{aligned}
$$

Servers migrate to the nodes along the links x_i. Once a server has sent its agent S_i^1 onto a node N_i, it waits for a signal $\overline{r}_i$ from S_i^1 to begin again. The agent of the server queries the node for customers waiting for service. If there is one waiting, the agent serves the customer, performing action s_i in interaction with the node N_i. Otherwise, via a pass action p_i, S_i^1 restores the initial state of N_i. In both cases, S_i^1 eventually resumes the server S, via an action $\overline{r}_i$. The arrival of a customer on node N_i is modelled by firing the action in_i, a synchronization with the operating environment. The one place buffer of N_i is implemented by blocking any other in_i action until the service of the current customer has ended.

Hillston [36] models the polling system with a first order calculus and associates stochastic information with prefixes in the syntax. An implicit assumption is that the routing of servers takes the same time for any node in the system. We could easily relax this assumption to distinguish the time spent for moving to one node or to another, by assigning different weights to the relative distances.

The transitions of the process P are in the Appendix; there are 210 of them and they form more than 45 distinct loops. The higher-order feature of our specification reduces the number of states to 68 from 210 of the specification in [36].

As an example of analysis we computed the usage of the link s_1 (which gives the throughput of the services on the node N_1) and we considered different performance characteristics of N_1. To model this aspect, we stipulate that all the local channels of N_1 ($x_1, s_1, \ldots$ and $x_2, s_2, \ldots$) have the same throughput. In particular we considered three cases for the throughput of the channels: 352.98, 7.74 and 6.66 Mbps. We got these values by profiling

three different machines at the Department of Computer Science of the University of Pisa by using Netperf [1]. The corresponding usage of link s_1 is 300.089, 9.88717 and 9.64 *communications/sec* which turns out to fit with our experimental data.

The transition system for the polling system and the throughputs above have been computed by using a prototypal tool described in [10]. Its kernel implements the enhanced operational semantics of π-calculus and assists the user in the definition of cost functions according to different architectures (uniprocessors, multiprocessors, network, cluster of networks). Once the cost functions have been defined, the tool computes the generator matrix of a CTMC, from the proved transition system of a specification. The user then chooses the measure of interest, and the tool generates a reward structure and interfaces *Mathematica* for numerical calculations.

7 Conclusions and further work

We have presented a framework in which the performance analysis of systems is driven by the semantics of their specifications. We used an enhanced structural operational semantics, called proved, whose transitions are labelled by (encodings of) their proofs. Taking advantage of enhanced labels, we associate rates with transitions mechanically: we only relabel transitions with probabilistic information. This is done symbolically, by looking at the enhanced labels, alone. Actual values are obtained as soon as the user provides some additional information about the architecture on which the system under analysis runs.

Since enhanced labels can be tightly connected to the routines called by the run-time support to implement each operator of the language, the information about architectural features can be supplied by a compiler. Indeed, calculi like the one considered here constitute the core of languages for reactive and mobile systems, e.g. Facile [65], PICT [54], CML [53], Esterel [6]. In some cases, HOπ is used as an intermediate language of them, i.e. as the code produced by some steps of compilation (see the encoding of Java in the π-calculus [34]). The compiled code can therefore be annotated to support quantitative analysis following our proposal, thus helping to mechanize performance analysis and to do it at compile time.

In this paper we have assumed that any activity is exponentially distributed, but general distributions are also possible (see [57]), as they depend on enhanced labels, alone.

Once rates have been assigned to transitions, it is easy to derive the CTMC associated with a transition system of a process. From its stationary distribution, if any, we evaluate the performance of the process in hand.

We have programmed a prototypal tool in ML [10], which generates

proved transition systems and their stochastic relabelling, supporting the user in the choice among a few cost functions for some typical architectures. It then computes CTMCs and some performance measures, e.g. throughput, via *Mathematica*. The tool has been used to compute the values of the examples on which we exemplified our approach.

Our first example considers the network management system analysed by Baldi and Picco in [4], and our results agree with theirs, under the same assumptions. We also analysed a slight variant of a polling system, studied in [36, 44, 39]. Although in this case a comparison cannot be as precise as in the previous one, the results we got seem quite sensible. This makes us confident that our approach is indeed applicable and makes sense. Of course, much work is still needed to test our ideas. We plan to apply them on real size applications, in order to prove whether our proposal is scalable and to gain more experience in the definition of sensible cost functions.

It is worth noticing that our approach follows the same pattern presented in [23] to derive behavioural information from our enhanced labels. Also, behavioural properties can be checked by using a tool integrated with ours. Therefore, we propose our operational semantics as a uniform framework to carry out integrated behavioural and quantitative analysis.

We claim that formal methods can help the designers in the development of applications by driving and assisting them in error-prone steps such as the translation of a specification into a model suitable for quantitative analysis. We think that our proposal is a little step in this direction.

Acknowledgements. The authors wish to thank the anonymous referees for their precise and helpful remarks, and Jane Hillston for her illuminating comments and careful suggestions on the example of a polling system.

References

[1] *Netperf: A network performance benchmark, Revision 2.1*, 1996. Information Networks Division, Hewlett-Packard.

[2] K. Ahlers, D. E. Breen, C. Crampton, E. Rose, M. Tucheryan, R. Whitaker, and D. Greer. An augmented vision system for industrial applications. In *SPIE Photonics for Industrial Applications Conference Proceedings*, October 1994.

[3] A.A. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press, 1978.

[4] M. Baldi and G.P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proceedings of ICSE'98*. ACM Press, 1998.

[5] M. Bernardo, L. Donatiello, and R. Gorrieri. A formal approach to the integration of performance aspects in the modelling and analysis of concurrent systems. *Information and Computation*, 144:83–154, 1998.

[6] G. Berry and L. Cosserat. The synchronous programming language ESTEREL and its mathematical semantics. In *Seminar on Concurrency, LNCS 197*. Springer-Verlag, 1984.

[7] R. Borgia, P. Degano, C. Priami, L. Leth, and B. Thomsen. Understanding mobile agents via a non-interleaving semantics for facile. In R. Cousot and D.A. Schmidt, editors, *Proceedings of SAS'96, LNCS 1145*, pages 98–112. Springer-Verlag, 1996.

[8] G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, XI(4):433–452, 1988.

[9] E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality based process algebra. *The Computer Journal*, 38(6):552–565, 1995.

[10] L. Brodo, P. Degano, and C. Priami. A tool for quantitative analysis of π-calculus processes. In *In Proceedings of PAPM'00*. Carleton Scientific, 2000.

[11] P. Buchholz. On a markovian process algebra. Technical report, Informatik IV, University of Dortmund, 1994.

[12] L. Cardelli and A. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98*, LNCS 1378, pages 140–155. Springer-Verlag, 1998.

[13] A. Carzaniga, G.P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of ICSE'97*, pages 23–32. ACM Press, 1997.

[14] P. Cenciarelli, A. Knapp, B. Reus, and M. Wirsing. An event-based structural operational semantics of multi-threaded Java. In *In Formal Syntax and Semantics of Java, LNCS 1523*, 1998.

[15] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, and A. Sangiovanni-Vincentelli. Hardware-software codesign of embedded systems. *IEEE Micro*, 14:26–36, 1994.

[16] G. Clark. Formalising the specification of rewards with PEPA. In *Proceedings of PAPM'96*, pages 139–160. CLUT Torino, 1996.

[17] E. Coste-Maniere and B. Faverjon. A programming and simulation tool for robotics workcells. In *In Proceedings International Conference on Automation, Robotics, and Computer Vision*, 1990.

[18] L. de Alfaro. Stochastic transition systems. In *Proceedings of CONCUR'98, LNCS*. Springer-Verlag, 1998.

[19] P. Degano, R. De Nicola, and U. Montanari. Partial ordering derivations for CCS. In *Proceedings of FCT'85, LNCS 199*, pages 520–533. Springer-Verlag, 1985.

[20] P. Degano and R. Gorrieri. A causal semantics of action refinement. *Information and Computation*, 122(1):97–121, 1995.

[21] P. Degano and C. Priami. Proved trees. In *Proceedings of ICALP'92, LNCS 623*, pages 629–640. Springer-Verlag, 1992.

[22] P. Degano and C. Priami. Enhanced operational semantics. *ACM Computing Surveys*, 28(2):352–354, 1996.

[23] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, (216):237–270, 1999.

[24] P. Degano, C. Priami, L. Leth, and B. Thomsen. Causality for debugging mobile agents. *Acta Informatica*, 1999.

[25] D. Eager, E. Lazowska, and J. Zahorjan. Dynamic load sharing in homogeneous distributed systems. *IEEE TSE*, 12(5):662–675, 1986.

[26] W. Feller. *An Introduction to Probability Theory and its Applications*. Wiley, 1970.

[27] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proceedings of 23rd ACM Symposium on Principles of Programming Languages*, 1996.

[28] C. Fournet, G. Gonthier, J-J. Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *Proceedings of CONCUR'96, LNCS 1119*, pages 406–421. Springer-Verlag, 1996.

[29] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[30] N. Götz, U. Herzog, and M. Rettelbach. TIPP- a language for timed processes and performance evaluation. Technical Report 4/92, IMMD VII, University of Erlangen-Nurnberg, 1992.

[31] H. Hansson and B. Jonsson. A calculus of communicating systems with time and probability. In *Proceedings of IEEE RTSS'90*, Orlando, Florida, 1990.

[32] P.G. Harrison and B. Strulo. Stochastic process algebra for discrete event simulation. In *Quantitative Methods in Parallel Systems*, pages 18–37, 1995.

[33] C. Harvey. Performance engineering as an integral part of system design. *BT Technology Journal*, 4(3):143–147, 1986.

[34] O. M. Herescu and C. Palamidessi. Probabilistic asynchronous π-calculus. In *Proceedings of 3rd International Conference FOSSACS 2000*, pages 146–10. Springer-Verlag, LNCS 1784, 2000.

[35] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras - between LOTOS and Markov chains. *Computer Networks and ISDN Systems*, 30(9-10):901–924, 1998.

[36] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[37] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[38] R. Howard. *Dynamic Probabilistic Systems:Semi-Markov and Decision Systems*, volume II. Wiley, 1971.

[39] O.C. Ibe and K.S. Trivedi. Stochastic petri net models of polling systems. *IEEE Journal on Selected Areas of Communication*, 8(9), 1990.

[40] L. Jategaonkar Jagadeesan, C. Puchol, and J.E. Von Olnhausen. A formal approach to reactive systems software: A telecommunications application in Esterel. In *In Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques*, 1995.

[41] C.B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, 1990.

[42] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In *Proceedings of CONCUR'92*, volume 630 of *LNCS*. Springer-Verlag, 1992.

[43] A. Maggiolo-Schettini and S. Tini. Applying techniques of asynchronous concurrency to synchronous languages. *Fundamenta Informaticae*, 40:221–250, 1999.

[44] M. Ajmone Marsan, S. Donatelli, F. Neri, and U. Rubino. On the construction of abstract GSPNs:an exercise in modelling. In *Proceedings of 4^{th} PNPM*, 1991.

[45] R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.

[46] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[47] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.

[48] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[49] D.S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. Technical report, HP Labs, 1998.

[50] G. Murakami and R. Sethi. Terminal call processing in Esterel. In *In Proceedings of IFIP 92 World Computer Congress*, 1992.

[51] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory*. Springer-Verlag, 1995.

[52] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Real Time: Theory in Practice, LNCS 600*, pages 526–548. Springer-Verlag, 1991.

[53] F. Nielson and H. R. Nielson. From CML to its process algebra. *Theoretical Computer Science*, 155:179–219, 1996.

[54] B.C. Pierce and D.N. Turner. PICT: A programming language based on the pi-calculus. In *In Proof, Language and Interaction: Essays in honour of Robin Milner*. MIT Press, 1999.

[55] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, 1981.

[56] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.

[57] C. Priami. Stochastic π-calculus with general distributions. In *Proceedings of PAPM'96*, pages 41–57. CLUT Torino, 1996.

[58] A. Reibman, R. Smith, and K. Trivedi. Markov and Markov reward model transient analysis: an overview of numerical approaches. *European Journal of Operations Research*, 40:257–267, 1989.

[59] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings of POPL'98*, pages 378–390, 1998.

[60] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[61] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[62] J.-P. Talpin. The Calumet Experiment in Facile - A Model for Group Communication and Interaction Control in Cooperative Applications. Technical Report ECRC-94-26, European Computer-Industry Research Centre, 1994.

[63] J.-P. Talpin, P. Marchal, and K.: Ahlers. Calumet - A Reference Manual. Technical Report ECRC-94-30, European Computer-Industry Research Centre, 1994.

[64] B. Thomsen. Plain CHOCS: a second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[65] B. Thomsen, L. Leth, S. Prasad, T.-M. Kuo, A. Kramer, F. Knabe, and A. Giacalone. Facile Antigua Release Programming Guide. Technical Report ECRC-93-20, European Computer-Industry Research Centre, 1993.

[66] K.S. Trivedi. *Probability and Statistics with Reliability, Queeing and Computer Science Applications*. Edgewood Cliffs, NY, 1982.

[67] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 1995.

A Transition system of the polling specification

The following is transition system for the multi-server multi-queue system P specified in Sect. 6. For the sake of readability, in its transitions we omit restrictions and we use the following auxiliary processes, where $i = 1, 2$.

$$N_i^1 = x_i(U).U \mid s_i.Node_i \qquad N_i^2 = S_i^2 \mid Node_i$$
$$N_i^3 = S_i^1 \mid s_i.Node_i \qquad N_i^4 = S_i^2 \mid Node_i$$
$$N_i^5 = S_i^2 \mid s_i.Node_i \qquad S_i^2 = \overline{r}_i.x_i(U).U$$

$$P \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} ((N_1^2|N_2)|(r_1.S|S)) = P_1$$
$$P \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} ((N_1|N_2^2)|(r_2.S|S)) = P_2$$
$$P \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} ((N_1^2|N_2)|(S|r_1.S)) = P_3$$
$$P \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_1^1\rangle\rangle} ((N_1|N_2^2)|(S|r_2.S)) = P_4$$
$$P \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^1|N_2)|(S|S)) = P_5$$
$$P \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1|N_2^1)|S|S)) = P_6$$
$$P_1 \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^3|N_2)|(r_1.S|S)) = P_7$$
$$P_1 \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} ((N_1^2|N_2^2)|(r_1.S|r_2.S)) = P_8$$
$$P_1 \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^2|N_2^1)|(r_1.S|S)) = P_9$$
$$P_1 \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} ((N_1^4|N_2)|(r_1.S|S)) = P_{18}$$
$$P_2 \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^1|N_2^2)|(r_2.S|S)) = P_{10}$$
$$P_2 \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\langle S_1^1\rangle\rangle} ((N_1^2|N_2^2)|(r_2.S|r_1.S)) = P_{11}$$
$$P_2 \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1|N_2^3)|(r_2.S|S)) = P_{12}$$
$$P_2 \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1|N_2^4)|(r_2.S|S)) = P_{19}$$
$$P_3 \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^3|N_2)|(S|r_1.S)) = P_{13}$$
$$P_3 \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{11}$$
$$P_3 \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^2|N_2^1)|(S|r_1.S)) = P_{14}$$
$$P_3 \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} ((N_1^4|N_2)|(S|r_1.S)) = P_{20}$$
$$P_4 \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^1|N_2^2)|(S|r_2.S)) = P_{15}$$
$$P_4 \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\langle S_1^1\rangle\rangle} P_8$$

$P_4 \xrightarrow{||_0||_1||_1+_0in_2} ((N_1|N_2^3)|(S|r_2.S)) = P_{16}$

$P_4 \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1|N_2^4)|(S|r_2.S)) = P_{21}$

$P_5 \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_7$

$P_5 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{10}$

$P_5 \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{13}$

$P_5 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_1+_1\overline{x}_2\langle S_1^1\rangle\rangle} P_{15}$

$P_5 \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^1|N_2^1)|(S|S)) = P_{17}$

$P_6 \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_9$

$P_6 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{12}$

$P_6 \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{14}$

$P_6 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_1+_1\overline{x}_2\langle S_1^1\rangle\rangle} P_{16}$

$P_6 \xrightarrow{||_0||_0||_1+_0in_1} P_{17}$

$P_7 \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{18}$

$P_7 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} ((N_1^3|N_2^2)|(r_1.S|r_2.S)) = P_{22}$

$P_7 \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^3|N_2^1)|(r_1.S|S)) = P_{23}$

$P_8 \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^2|N_2^3)|(r_1.S|r_2.S)) = P_{26}$

$P_8 \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1^2|N_2^4)|(r_1.S|r_2.S)) = P_{25}$

$P_8 \xrightarrow{||_0||_0||_1+_0in_1} P_{22}$

$P_8 \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} ((N_1^4|N_2^2)|(r_1.S|r_2.S)) = P_{24}$

$P_9 \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} ((N_1^4|N_2^1)|(r_1.S|S)) = P_{27}$

$P_9 \xrightarrow{||_0||_0||_1+_0in_1} P_{23}$

$P_9 \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_26P_{10} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^1|N_2^3)|(r_2.S|S)) = P_{30}$

$P_{10} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} (N_1^3|N_2^2)|(r_2.S|r_1.S)) = P_{28}$

$P_{10} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1^1|N_2^4)|(r_2.S|S)) = P_{29}$

$P_{11} \xrightarrow{||_0||_0||_1+_0in_1} P_{28}$

$P_{11} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} ((N_1^4|N_2^2)|(r_2.S|r_1.S)) = P_{31}$

$P_{11} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^2|N_2^3)|(r_2.S|r_1.S)) = P_{33}$

$P_{11} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1^2|N_2^4)|(r_2.S|r_1.S)) = P_{32}$

$P_{12} \xrightarrow{||_0||_0||_1+_0in_1} P_{30}$

$P_{12} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{33}$

$P_{12} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{19}$

$P_{13} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{20}$

$P_{13} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{28}$

$P_{13} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^3|N_2^1)|(S|r_1.S)) = P_{34}$

$P_{14} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} ((N_1^4|N_2^1)|S|r_1.S)) = P_{35}$

$P_{14} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{33}$

$P_{14} \xrightarrow{||_0||_0||_1+_0in_1} P_{34}$

$P_{15} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{22}$
$P_{15} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1^1|N_2^4)|(S|r_2.S)) = P_{36}$
$P_{15} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^1|N_2^3)|(S|r_2.S)) = P_{37}$
$P_{16} \xrightarrow{||_0||_0||_1+_0 in_1} P_{37}$
$P_{16} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{26}$
$P_{16} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1 s_2\rangle} P_{21}$
$P_{17} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{23}$
$P_{17} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{30}$
$P_{17} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{34}$
$P_{17} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_1^1\rangle\rangle} P_{37}$
$P_{18} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0 r_1\rangle} P$
$P_{18} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{24}$
$P_{18} \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^5|N_2)|(r_1.S|S)) = P_{38}$
$P_{18} \xrightarrow{||_0||_1||_1+_0 in_2} P_{27}$
$P_{19} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0 r_2\rangle} P$
$P_{19} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{32}$
$P_{19} \xrightarrow{||_0||_0||_1+_0 in_1} P_{29}$
$P_{19} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1|N_2^5)|(r_2.S|S)) = P_{39}$
$P_{20} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1 r_1\rangle} P$
$P_{20} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{31}$
$P_{20} \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^5|N_2)|(S|r_1.S)) = P_{40}$
$P_{20} \xrightarrow{||_0||_1||_1+_0 in_2} P_{35}$
$P_{21} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1 r_2\rangle} P$
$P_{21} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{25}$
$P_{21} \xrightarrow{||_0||_0||_1+_0 in_1} P_{36}$
$P_{21} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1|N_2^5)|(S|r_2.S)) = P_{41}$
$P_{22} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1 s_1\rangle} P_{24}$
$P_{22} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1^3|N_2^4)|(r_1.S|r_2.S)) = P_{42}$
$P_{22} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^3|N_2^3)|(r_1.S|r_2.S)) = P_{43}$
$P_{23} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1 s_1\rangle} P_{27}$
$P_{23} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{43}$
$P_{24} \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^5|N_2^2)|(r_1.S|r_2.S)) = P_{44}$
$P_{24} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0 r_1\rangle} P_4$
$P_{24} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^4|N_2^3)|(r_1.S|r_2.S)) = P_{46}$
$P_{24} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1^4|N_2^4)|(r_1.S|r_2.S)) = P_{45}$
$P_{25} \xrightarrow{||_0||_0||_1+_0 in_1} P_{42}$
$P_{25} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1 r_2\rangle} P_1$
$P_{25} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^2|N_2^5)|(r_1.S|r_2.S)) = P_{47}$
$P_{25} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} P_{45}$
$P_{26} \xrightarrow{||_0||_0||_1+_0 in_1} P_{43}$
$P_{26} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} P_{46}$
$P_{26} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1 s_2\rangle} P_{25}$
$P_{27} \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^5|N_2^1)|(r_1.S|S)) = P_{48}$
$P_{27} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{46}$
$P_{27} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0 r_1\rangle} P_6$
$P_{28} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^3|N_2^3)|(r_2.S|r_1.S)) = P_{50}$
$P_{28} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1^3|N_2^4)|(r_2.S|r_1.S)) = P_{49}$
$P_{28} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1 s_1\rangle} P_{31}$
$P_{29} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^1|N_2^5)|(r_2.S|S)) = P_{51}$
$P_{29} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{49}$
$P_{29} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0 r_2\rangle} P_5$
$P_{30} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1 s_2\rangle} P_{29}$
$P_{30} \xrightarrow{\langle||_0||_0||_0 x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{50}$
$P_{31} \xrightarrow{||_0||_0||_1+_0 in_1} ((N_1^5|N_2^2)|(r_2.S|r_1.S)) = P_{52}$
$P_{31} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1 r_1\rangle} P_2$
$P_{31} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^4|N_2^3)|(r_2.S|r_1.S)) = P_{54}$
$P_{31} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1 p_2\rangle} ((N_1^4|N_2^4)|(r_2.S|r_1.S)) = P_{53}$
$P_{32} \xrightarrow{||_0||_0||_1+_0 in_1} P_{49}$
$P_{32} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} P_{53}$
$P_{32} \xrightarrow{||_0||_1||_1+_0 in_2} ((N_1^2|N_2^5)|(r_2.S|r_1.S)) = P_{55}$
$P_{32} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0 r_2\rangle} P_3$
$P_{33} \xrightarrow{||_0||_0||_1+_0 in_1} P_{50}$
$P_{33} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1 p_1\rangle} P_{54}$
$P_{33} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1 s_2\rangle} P_{32}$
$P_{34} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1 s_1\rangle} P_{35}$
$P_{34} \xrightarrow{\langle||_0||_1||_0 x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{50}$

$P_{35} \xrightarrow{||_0||_0||_1+_0in_1} ((N_1^5|N_2^1)|(S|r_1.S)) = P_{56}$ $P_{35} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{54}$

$P_{35} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_6$

$P_{36} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^1|N_2^5)|(S|r_2.S)) = P_{57}$ $P_{36} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{42}$

$P_{36} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_5$

$P_{37} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{36}$ $P_{37} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{43}$

$P_{38} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_5$ $P_{38} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{44}$

$P_{38} \xrightarrow{||_0||_1||_1+_0in_2} P_{48}$

$P_{39} \xrightarrow{||_0||_0||_1+_0in_1} P_{51}$ $P_{39} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{55}$

$P_{39} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_6$

$P_{40} \xrightarrow{||_0||_1||_1+_0in_2} P_{56}$ $P_{40} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{52}$

$P_{40} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_5$

$P_{41} \xrightarrow{||_0||_0||_1+_0in_1} P_{57}$ $P_{41} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{47}$

$P_{41} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_6$

$P_{42} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^3|N_2^5)|(r_1.S|r_2.S)) = P_{58}$ $P_{42} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{45}$

$P_{42} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_7$

$P_{43} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{46}$ $P_{43} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{42}$

$P_{44} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{15}$ $P_{44} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1^5|N_2^4)|(r_1.S|r_2.S)) = P_{59}$

$P_{44} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^5|N_2^3)|(r_1.S|r_2.S)) = P_{60}$

$P_{45} \xrightarrow{||_0||_0||_1+_0in_1} P_{59}$ $P_{45} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^4|N_2^5)|(r_1.S|r_2.S)) = P_{61}$

$P_{45} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{21}$ $P_{45} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{18}$

$P_{46} \xrightarrow{||_0||_0||_1+_0in_1} P_{60}$ $P_{46} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{37}$

$P_{46} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{45}$

$P_{47} \xrightarrow{||_0||_0||_1+_0in_1} P_{58}$ $P_{47} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} P_{61}$

$P_{47} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_9$

$P_{48} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{17}$ $P_{48} \xrightarrow{\langle||_0||_1||_0x_2(U),||_1||_1+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{60}$

$P_{49} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^3|N_2^5)|(r_2.S|r_1.S)) = P_{62}$ $P_{49} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{53}$

$P_{49} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{13}$

$P_{50} \xrightarrow{||_0||_0\langle||_0\overline{s}_1,||_1s_1\rangle} P_{54}$ $P_{50} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{49}$

$P_{51} \xrightarrow{\langle||_0||_0||_0x_1(U),||_1||_1+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{62}$ $P_{51} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{17}$

$P_{52} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^5|N_2^3)|(r_2.S|r_1.S)) = P_{63}$ $P_{52} \xrightarrow{||_0||_1\langle||_0+_1\overline{p}_2,||_1+_1p_2\rangle} ((N_1^5|N_2^4)|(r_2.S|r_1.S)) = P_{67}$

$P_{52} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{10}$

$P_{53} \xrightarrow{||_0||_0||_1+_0in_1} P_{62}$ $P_{53} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{19}$

$P_{53} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^4|N_2^5)|(r_2.S|r_1.S)) = P_{64}$ $P_{53} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{20}$

$P_{54} \xrightarrow{||_0||_0||_1+_0in_1} P_{63}$ $P_{54} \xrightarrow{\langle||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{12}$

$P_{54} \xrightarrow{||_0||_1\langle||_0\overline{s}_2,||_1s_2\rangle} P_{53}$

$P_{55} \xrightarrow{||_0||_0||_1+_0in_1} P_{62}$ $P_{55} \xrightarrow{||_0||_0\langle||_0+_1\overline{p}_1,||_1+_1p_1\rangle} P_{64}$

$P_{55} \xrightarrow{\langle||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{14}$

$P_{56} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{17}$

$P_{56} \xrightarrow{\langle ||_0||_1||_0x_2(U),||_1||_0+_1\overline{x}_2\langle S_2^1\rangle\rangle} P_{63}$

$P_{57} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{17}$

$P_{57} \xrightarrow{\langle ||_0||_0||_0x_1(U),||_1||_0+_0\overline{x}_1\langle S_1^1\rangle\rangle} P_{58}$

$P_{58} \xrightarrow{||_0||_0\langle ||_0\overline{s}_1,||_1s_1\rangle} P_{61}$

$P_{58} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{23}$

$P_{59} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{36}$

$P_{59} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^5|N_2^5)|(r_1.S|r_2.S)) = P_{65}$

$P_{59} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{38}$

$P_{60} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{37}$

$P_{60} \xrightarrow{||_0||_1\langle ||_0\overline{s}_2,||_1s_2\rangle} P_{59}$

$P_{61} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{41}$

$P_{61} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{27}$

$P_{62} \xrightarrow{||_0||_0\langle ||_0\overline{s}_1,||_1s_1\rangle} P_{64}$

$P_{62} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{34}$

$P_{63} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{30}$

$P_{63} \xrightarrow{||_0||_1\langle ||_0\overline{s}_2,||_1s_2\rangle} P_{62}$

$P_{64} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{39}$

$P_{64} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{35}$

$P_{65} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_0r_1\rangle} P_{57}$

$P_{65} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_1r_2\rangle} P_{48}$

$P_{66} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{51}$

$P_{66} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{56}$

$P_{67} \xrightarrow{||_0||_1||_1+_0in_2} ((N_1^5|N_2^5)|(r_2.S|r_1.S)) = P_{66}$

$P_{67} \xrightarrow{\langle ||_0||_1||_1\overline{r}_2,||_1||_0r_2\rangle} P_{40}$

$P_{67} \xrightarrow{\langle ||_0||_0||_0\overline{r}_1,||_1||_1r_1\rangle} P_{29}$