HAL
open science

The Attributed Pi Calculus

Mathias John, Cédric Lhoussaine, Joachim Niehren, Adelinde Uhrmacher

To cite this version:

Mathias John, Cédric Lhoussaine, Joachim Niehren, Adelinde Uhrmacher. The Attributed Pi Calculus. Computational Methods in Systems Biology, 6th International Conference CMSB, Oct 2008, Rostock, Germany. pp.83-102. inria-00308970v4

HAL Id: inria-00308970

https://inria.hal.science/inria-00308970v4

Submitted on 2 Nov 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The Attributed Pi-Calculus with Priorities

Mathias John[1], Cédric Lhoussaine[24],
Joachim Niehren[34], and Adelinde M. Uhrmacher[1]

[1] University of Rostock
Institute of Computer Science, Modeling and Simulation Group
[2] University of Lille 1
[3] Inria, Lille
[4] BioComputing group, Lifl* & Iri**, Lille

Abstract. We present the attributed π-calculus for modeling concurrent systems with interaction constraints depending on the values of attributes of processes. The λ-calculus serves as a constraint language underlying the π-calculus. Interaction constraints subsume priorities, by which to express global aspects of populations. We present a non-deterministic and a stochastic semantics for the attributed π-calculus. We show how to encode the π-calculus with priorities and polyadic synchronization $\pi@$ and thus dynamic compartments, as well as the stochastic π-calculus with concurrent objects SPiCO.
We illustrate the usefulness of the attributed π-calculus for modeling biological systems at two particular examples: Euglena's spatial movement in phototaxis, and cooperative protein binding in gene regulation of bacteriophage lambda. Furthermore, population-based model is supported beside individual-based modeling. A stochastic simulation algorithm for the attributed π-calculus is derived from its stochastic semantics. We have implemented a simulator and present experimental results, that confirm the practical relevance of our approach.

1 Introduction

Biological systems are populations of molecules that evolve over time. Numeric modeling approaches based on differential equations assume that the evolution is continuous and deterministic. Stochastic approaches founded in continuous time Markov chains (CTMCs) consider the evolution of populations as discrete and non-deterministic. Transitions of a CTMC express state changes of the population. They can be obtained by modeling biochemical systems on population level or on individual level, for instance, by viewing chemical reactions as transformation rules for populations or respectively as rules for interactions between individuals. Deterministic models can be obtained from stochastic models by averaging over possible behaviors [1–3].

A plethora of formal concurrent programming languages for stochastic modeling of biological systems has been proposed, since the seminal work of Regev

[1] Laboratoire d'Informatique Fondamentale de Lille, Cnrs Umr8022.
[2] Interdisciplinary Research Institute, Cnrs Usr3078.

and Shapiro [4, 5] on the stochastic π-calculus. Semantically, programs in such languages define CTMCs. Operationally, they enable stochastic simulation without constructing the CTMC, so that only a (small) part of the (often huge) state space of the CTMC is inspected on the fly.

We distinguish rule-based and object-centered modeling languages. Note that this distinction is largely independent of a population-based or individual-based modeling style. Rule-based languages provide rewrite rules, by which to express chemical reactions and pathways being composed thereof. Multiset rewriting is the foundation of BIOCHAM [3] and Petri nets [6], while graph rewriting underlies the κ-calculus[7, 8] and Bigraphs [9].

Object-centered modeling languages describe concurrent systems as collections of interacting objects. Various interaction concepts were investigated. Channel communication underlies the stochastic π-calculus [4, 10–12], and its spatial relatives [13, 14], while the participation of a species in a chemical reaction is the approach of Bio-PEPA [15], and communication through a global constraint store the principle of stochastic concurrent constraint programming (SCCP) [16]. Different interaction concepts may lead to different modeling perspectives. The π-calculus and its relatives support the modeling of molecules as objects (individual-based), Bio-PEPA features the modeling of species as objects, whose amounts are changed by chemical rules (species-based). The idea of SCCP is to model chemical reactions as objects that change the population residing in the constraint store (population-based).

Various kinds of local and global constraints on interactions were proposed for the π-calculus. Polyadic synchronization [17] can express local constraints, which impose equality of tuples of channels. Priorities yield global constraints that were added to the π-calculus [18] in order to model global aspects of populations in individual-based modeling, such as global state updates following local interactions. Stochastic rates provide another kind of global constraints, that were proposed for the π-calculus [19, 20, 12, 21] in order to enable stochastic modeling and simulation. As shown in [18], dynamic compartments become expressible in the π-calculus with priorities and polyadic synchronization $\pi@$. There exists a variant called s$\pi@$ that is equipped with a stochastic simulation algorithm [22]. It can express dynamic compartments with variable volumes, but does neither provide a stochastic semantics in terms of CTMCs nor any support for general purpose attributes beside compartment volumes.

Basic Ideas. In this paper, we propose the attributed π-calculus as a unifying framework, in order to express local and global interaction constraints for the π-calculus. We obtain it by extending the π-calculus with attributed processes and interaction constraints, that may depend on the attribute values.

Attribute values of various types are useful in order to define various properties of biological processes. Two examples on different levels of abstraction illustrate the basic idea: first, a cell *Cell(coord,vol)* is attributed by coordinates $coord \in \mathbb{R}^3$ and a volume $vol \in \mathbb{R}_+$. Second, protein *Prot(comp)* is attributed by the cellular compartment $comp \in \{\texttt{nucleus}, \ldots\}$ in which it is located.

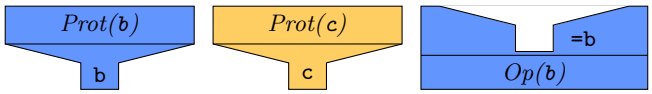


Fig. 1. Only the protein of sort b is permitted to bind to the operator of sort b, but not the protein of sort c. Equality of sorts is tested by the operator, once it sees the sort of the protein, by applying the test function $\lambda x.x{=}\mathtt{b}$.

Interaction constraints on attribute values may be imposed in order to define, whether or not two attributed processes are allowed to communicate. Consider e.g. proteins $Prot(x)$ of sort $x \in \{\mathtt{b}, \mathtt{c}\}$ and operators *Op(y)* of sort $y \in \{\mathtt{b}, \mathtt{c}\}$ able to bind a protein of the same sort. For the binding, equality of their sorts $x{=}y$ is required. In the attributed π-calculus this can be expressed by the following definitions, which are valid for all possible values of the attributes x and y:

$$Prot(x) \triangleq bind[x]!().0$$
$$Op(y) \triangleq bind[\lambda x.x = y]?().OpBound(y)$$

This says that before enabling a binding action, *Prot(x)* needs to provide its sort as specified by $bind[x]$. The operator receives the value of x and tests it for equality with its own sort, as imposed by $bind[\lambda x.x{=}y]$; the equality constraint is expressed by the Boolean valued function $\lambda x.x{=}y$. Consider for instance a system with two actors $Prot(\mathtt{b})$ and $Op(\mathtt{b})$:

$$Prot(\mathtt{b}) \mid Op(\mathtt{b}) \rightarrow OpBound(\mathtt{b})$$

The interaction constraint for these two actors is composed by function application $(\lambda x.x{=}\mathtt{b})\mathtt{b}$. It evaluates to the truth value of $\mathtt{b}{=}\mathtt{b}$ which is true. This enables the above binding action, by which the operator turns into its bound state. The slightly more complex system $Prot(\mathtt{b}) \mid Prot(\mathtt{c}) \mid Op(\mathtt{b})$ where only the first protein is permitted to bind to the operator is illustrated in Fig. 1.

In the above example, interaction constraints evaluate to Boolean values. If we permit constraints that evaluate to positive real numbers, we can express stochastic rates that depend on attribute values. Or else, if we generalize constraints such that they return values of an ordered set, we obtain a π-calculus where priorities may depend on attribute values.

Contributions. We introduce the attributed π-calculus $\pi(\mathcal{L})$ with priorities, in which $\mathcal{L}$ is a language providing attribute values and constraints. We propose to use a call-by-value λ-calculus as attribute language $\mathcal{L}$, in which we leave the choice of constants of $\mathcal{L}$ parametric. This enables the many reasonable choices, such as positive real numbers for stochastic rates or arbitrary ordered sets for priorities. Note that the role of the attribute language is analogous to the role of the constraint language in constraint logic programming $\textsc{clp}(\mathcal{L})$ [23] or concurrent constraint programming $\textsc{ccp}(\mathcal{L})$ [24], except that we now permit higher-order languages $\mathcal{L}$.

We present two operational semantics for $\pi(\mathcal{L})$, a non-deterministic semantics with priorities and a stochastic semantics in terms of CTMCs, under the assumptions that successful constraints evaluate to stochastic rates in $\mathbb{R}_+^\infty$. We show that the stochastic semantics is a proper refinement of the non-deterministic semantics, in that it permits the same reduction steps (Proposition 5). Our stochastic semantics does not impose any syntactical restrictions such as the biochemical form. This is relevant, since it simplifies all technical results in contrast to most previous approaches (including the conference version [25] of the present article). We also provide a type system, that extends that of the simply typed λ-calculus to the process level, and prove type safety (Theorem 1). Types express invariants that are useful to detect program errors for higher-order languages, and thus when modeling biological systems in such languages.

We illustrate the usefulness of the attributed π-calculus for modeling and simulation in systems biology. We first describe the light dependent movement of Euglena, a single celled organism living in inland water. This illustrates the treatment of spatial aspects with location dependent attributes. The second example regards cooperative binding, a phenomenon, which is often observed in gene regulatory systems [26, 27]. This is an example of an interaction between three actors that can be simplified by using attributes. We also discuss new opportunities for population based-modeling of chemical reactions in the attributed π-calculus. Here, populations are represented by attributed processes for multisets such as $P(4, 5, 3)$ and reactions as actors that transform populations. This modeling style was proposed for SCCP [16].

We present an encoding of the π-calculus with priorities in an ordered set into an attributed π-calculus with priorities as constants, and prove it correct for the non-deterministic semantics (Theorem 2). We show that the same translation can be used to encode the stochastic π-calculus into the attributed π-calculus with stochastic rates as constants and two levels of priorities and that the encoding is correct with respect to the stochastic semantics (Theorem 3). Types are preserved by the encoding.

We show how to encode the π-calculus with priorities and polyadic synchronization $\pi@$ into the attributed π-calculus with equality tests, and prove the correctness wrt. the non-deterministic semantics (Theorem 4). Thereby, we lift encodings of the spatial modeling languages BioAmbients [13] and Brane [14] to the attributed π-calculus (see [18, 28]). Note that there exists neither a stochastic semantics in terms of CTMCs for $\pi@$ nor a type system.

Finally, we present a stochastic simulation algorithm for the attributed π-calculus, that is inferred directly from the stochastic semantics. We have implemented this algorithm on top of the modeling and simulation framework JAMES II [29]. We discuss the performance of our simulator at run-time experiments with the Euglena model, and show that it yields comparable results with the SPiM simulator for the stochastic π-calculus [11]. This confirms the practical relevance of our approach.

Compared to the conference version [25], we added priorities to the non-deterministic semantics of the attributed π-calculus, and extended the encoding

of $\pi@$ in the attributed π-calculus, such that it accounts for priorities. The whole presentation has been changed, such that the attributed π-calculus becomes a proper generalization of the π-calculus with priorities and the stochastic π-calculus. Annotations have been moved from channels to communication prefixes, so that no global environments needed to be introduced. The changes have led to an important simplification of the stochastic semantics, which enables our correctness proofs. We also improved our implementation and extended the practical experiments.

Discussion. We argued that the attributed π-calculus supports both modeling styles, molecules as processes (individual-based) and reactions as processes (population-based). In the individual-based approach, priorities are needed to express global aspects of the population, while no priorities are not needed in the population-based style.

The stochastic semantics of the attributed π-calculus only slightly liberalizes the law of mass action, in that the propensity of an interaction remains the product of multiplicities of the interaction partners and the stochastic rate of the interaction. The only difference is that the stochastic rate may now be dependent on the values of the attributes of the interacting processes, so that other aspects than their masses may intervene. A weakness of the π-calculus approach presented here is that it does not support different kinetics such as Michaelis-Menten in individual-based modeling (while it does for population-based modeling).

Whereas an increased expressiveness of a modeling language typically will ease the development of models, it requires additional support for developing models, e.g. to ensure type consistency, and burdens model analysis and simulation. Here we followed a tradition in concurrent programming languages, to combine a process level and a sequential core language (expression level). Since only the λ-calculus with types of low (first or second) order are used in practice, we believe that our extension is justified. This holds in particular, when accepting the π-calculus as a starting point, since it is higher-order anyway.

Outline. In Section 2 we present the π-calculus with priorities and the stochastic π-calculus in a uniform manner. We start from an ordered set $(R, <)$ whose elements may be either priorities or stochastic rates. We provide a unified syntax for processes in both calculi, in which communication prefixes (rather than channels) are annotated by values of R. We then present two operational semantics for the same syntax, a non-deterministic semantics as for the π-calculus with priorities, and a stochastic semantics as for the stochastic π-calculus.

In Section 3, we present the attributed π-calculus with priorities. In the syntax, we permit λ-expressions instead of channels, priorities or stochastic rates. Again, we introduce two operational semantics, the one is non-deterministic and the other stochastic. We present a type system, which lifts the simple type system of the λ-calculus to attributed process, and prove type safety under the assumption that the attribute language is type safe.

Section 4 contains the examples for modeling biological systems in the attributed π-calculus, and a discussion about population- versus individual-based modeling. In Section 5, we encode the π-calculus with priorities and $\pi@$ into the attributed π-calculus with priorities, and prove correctness. We provide an encoding of SPiCO and discuss variants of the stochastic and attributed π-calculus. In Sections 6 and 7, we present the stochastic simulation algorithm and discuss the performance of our implementation.

Related work. The π-calculus with polyadic synchronization was first proposed in [17] but could be extended by more general data terms as in [30] without particular difficulties. An extended π-calculus with polyadic synchronization, data terms, and explicit substitutions motivated by security applications was proposed in [31]. Another instance of the idea of interaction constraints is present in BlenX[32, 33], where interactions are allowed only between prefixes sharing a compatible type annotation.

Further modeling approaches based on the stochastic π-calculus can be found in [34, 35]. All stochastic π models can be translated into the attributed π-calculus. This requires to encode the versions of the stochastic π-calculus used there into the attributed π-calculus. We show in Section 5.3, that we can express the version of the stochastic π-calculus where stochastic rates are annotated to channels.

The closest approach to ours is stochastic concurrent constraint programming (sCCP) [16]. There, one can indeed express attributed processes with stochastic rates depending on attribute values. This enables the modeling of "reactions as objects" and "populations as stores" (population-based), but not of "molecules as objects" (individual-based), since direct interactions between concurrent actors are not permitted. In sCCP they interact by indirection though a global constraint store. It should also be noticed that the constraints of the attributed π-calculus are higher-order while those of sCCP are first-order.

2 Pi-Calculus with Priorities

We start from the π-calculus with priorities, for which we provide a non-deterministic and a stochastic operational semantics. This latter leads us to a new version of the stochastic π-calculus, which in contrast to [11, 25] does not impose any syntactic restrictions (such as biochemical forms).

2.1 Design Decisions

We search for a unified treatment of the π-calculus with priorities and the stochastic π-calculus. Both, priorities and stochastic rates both express global properties, that concern the whole population of objects. For selecting a communication step with highest priority, one has to inspect all potential communication steps. Similarly, the probability of a communication step in a stochastic setting depends globally on all possible communication steps. In both cases, the

difficulty is thus to reason globally about all possible communication steps in a given population.

Our first design decision is to permit process definitions with recursion and parameters in the syntax (rather than replication). Definitions are very convenient for modeling and simulation in systems biology, and therefore supported by all current simulators for the stochastic π-calculus [12, 21, 10]. The difficulty in the presence of priorities is to discover all potential communication steps in a given process, since some of them may be hidden by definitions. In order to solve this problem, we propose to always apply process definitions exhaustively before selecting any communication step. Fortunately, the resulting operational semantics remains nicely simple, and can be generalized properly to the stochastic setting. The alternative approach elaborated in [18] consists in adding the replication operator to the syntax of the π-calculus, and to hide its application in the structural congruence of processes. Since applications of the structural congruence must be restricted when counting interaction opportunities as needed in the stochastic semantics, this approach cannot be applied here.

Our second choice is to annotate communication prefixes rather than channels by elements in an ordered set $(R, <)$ which may either contain priorities or stochastic rates. Note stochastic rates were annotated to channels in the conference version of the attributed π-calculus [25], and that priorities were not considered in the non-deterministic version there. The change toward prefix annotations simplifies our semantics considerably (and some proof obligations tremendously). Note also that we only annotate priorities to sender prefixes, rather than to sender and receiver prefixes, since otherwise one has to resolve conflicting priorities somehow [18]. See Section 5.3 for further discussions.

2.2 Syntax

Let $\texttt{Bool} = \{\texttt{true}, \texttt{false}\}$ be the set of Booleans, $\mathbb{N}$ the set of natural numbers starting from 1, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $\mathbb{R}_+$ the set of non-negative real numbers, and $\mathbb{R}_+^\infty = \mathbb{R}_+ \cup \{\infty\}$.

We start from a partially ordered set $(R, <)$ of priorities, an infinite set *Vars* of channel names $x, y \in$ *Vars*, and an infinite set of process names $A \in$ *Proc*, that have fixed arities in $\mathbb{N}_0$. Whenever we write a term $A(x_1, \ldots, x_n)$ we assume that n is the arity of A. We use tuple notation in many places, as for instance $\tilde{x}$ for tuples of channels. If $\tilde{x} = (x_1, \ldots, x_n)$ then we define the length of the tuple by $|\tilde{x}| = n$. Whenever we use terms $A(\tilde{x})$ we assume that the length of $\tilde{x}$ is equal to the arity of A. Substitutions replacing y by x are denoted by $[x/y]$. Substitutions $[\tilde{y}/\tilde{x}]$ apply to tuples of the same length $|\tilde{y}| = |\tilde{x}|$.

The syntax of the π-calculus with priorities is defined in Fig. 2. In addition to channel names $x \in$ *Vars* and priorities $r \in R$ there are 4 syntactic categories: prefixes π, processes P, sums M, and definitions D. A prefix is either a receiver $x?\tilde{y}$ or a sender $x{:}r!\tilde{z}$. We assume that all channel names in $\tilde{y}$ are pairwise distinct (since they are distinct formal parameters). A receiver is supposed to receive a tuple of values for $\tilde{y}$ on channel x, and a sender to send a tuple of values $\tilde{z}$ on channel x. The priority r of an interaction is determined by the

Prefixes	π	$::= x?\tilde{y}$	receiver
		$\mid x{:}r!\tilde{z}$	sender
Sums	M	$::= \pi.P$	prefixed process
		$\mid M_1 + M_2$	choice
Processes	P	$::= M$	sums
		$\mid A(\tilde{x})$	defined process
		$\mid P_1 \mid P_2$	parallel composition
		$\mid (\nu x)P$	channel creation
		$\mid \mathbf{0}$	idle process
Definitions	D	$::= A(\tilde{x}) \triangleq P$	parametric process definition

Fig. 2. Syntax of the π-calculus with channels $x, \tilde{x}, \tilde{y}, \tilde{z} \in \mathit{Vars}$ and priorities $r \in R$.

$fv(M_1 + M_2) = fv(M_1) + fv(M_2)$	$fv(\mathbf{0}) \;=\; \emptyset$
$fv(x?\tilde{y}.P) = \{x\} \cup (fv(P) \setminus \{\tilde{y}\})$	$fv(P_1 \mid P_2) = fv(P_1) \cup fv(P_2)$
$fv(x{:}p!\tilde{z}.P) = \{x\} \cup fv(\{\tilde{z}\}) \cup fv(P)$	$fv(A(\tilde{x})) = \{\tilde{x}\}$
$fv((\nu x)P) = fv(P) \setminus \{x\}$	$fv(A(\tilde{x}) \triangleq P) = fv(P) \setminus \{\tilde{x}\}$

Fig. 3. Free channel names.

sender. A term $\pi_1.P_1 + \ldots + \pi_n.P_n$ is a sum of guarded prefixes, that we denote by $\sum_{i=1}^{n} \pi_i.P_i$ equivalently. A process P may be either a defined process $A(\tilde{x})$, or a parallel composition $P_1|\ldots|P_n$ that we denote equivalently as $\prod_{i=0}^{n} P_i$, or a process $(\nu x)P$ creating a new channel x with scope P. If $\tilde{x} = (x_1, \ldots, x_n)$ then we write $(\nu\tilde{x})P$ instead of $(\nu x_1)\ldots(\nu x_n)P$. Note that our syntax provides empty products but not empty sums, i.e. if $n = 1$ then $\prod_{i=1}^{n} P_i = \mathbf{0}$ is the idle process, while $\sum_{i=1}^{n} P_i$ is undefined.

The free channel names $fv(P)$ are defined as usual in Fig. 3. The three variable binders are new binders $(\nu x).P$, formal parameters $\tilde{y}$ in input prefixes $x?\tilde{y}.P$, and formal parameters $\tilde{x}$ in definitions $A(\tilde{x}) \triangleq P$. We say that bound variables are renamed apart in P, if 1) no variable is bound twice in P, 2) no bound variable of P has a free occurrence in P, and 3) no bound variable of P has a free occurrence in some definition. We generally assume, that all processes P are renamed apart, before applying any interaction step to any subprocess of P.

The structural congruence on processes $\equiv$ remains the least congruence satisfying the axioms given in Fig. 4, i.e. consistent renaming of bound variables, associativity and commutativity of parallel composition and summation, the rule of the neutral element of $\mathbf{0}$ with respect to parallel composition, and scope intrusion and extrusion for ν-binders. Note that every process P is congruent to some process in *prenex form* $(\nu\tilde{x}) \prod_{i=1}^{n} P_i$, where all processes P_i are either sums M

$$
\begin{array}{rcl@{\qquad}rcl}
(P_1 \mid P_2) \mid P_3 & \equiv & P_1 \mid (P_2 \mid P_3) & P_1 \mid P_2 & \equiv & P_2 \mid P_1 \\
(M_1 + M_2) + M_3 & \equiv & M_1 + (M_2 + M_3) & M_1 + M_2 & \equiv & M_2 + M_1 \\
P \mid \mathbf{0} & \equiv & P & (\nu x)(P \mid Q) & \equiv & (\nu x)P \mid Q \text{ if } x \notin fv(Q) \\
P \equiv_\alpha Q & \Rightarrow & P \equiv Q & (\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P
\end{array}
$$

Fig. 4. Axioms of structural congruence

or defined processes $A(\tilde{y})$, such that all all bound variables are renamed apart, also from the free variables in the definitions of the defined processes.

As an example, we consider silent actions $\texttt{delay}{:}r.P$. A salient action becomes active with priority r without any communication partner and then behaves like P. In our syntax of the π-calculus, silent actions can be expressed by processes $(\nu\, delay)(delay?().P \mid delay{:}r!().\mathbf{0})$ where a dummy interaction partner sends the empty tuple on local channel *delay* with priority r and then disappears.

2.3 Operational Semantics

The operational semantics of the π-calculus with priorities is presented in Figure 6. It is defined by a reduction relation $\rightarrow$ that is based on three kinds of binary relations, reductions $\xrightarrow[nd]{r}$ with priority $r \in R$, reductions $\xrightarrow[nd]{err}$ leading to errors, and reductions $\xrightarrow[nd]{app}$ applying process definitions. The label *nd* distinguishes non-deterministic from stochastic reduction steps, *err* stands for error and *app* for application.

Fig. 5 contains the axioms of the binary relations on processes $\xrightarrow[nd]{\alpha}$ where $\alpha \in R \cup \{app, err\}$, which define local interaction opportunities on the level of individuals. A communication step (COM) applies to two parallel sums with matching prefixes, a sum with a receiver $x?\tilde{y}.P_1 + M_1$ and another with sender $x{:}r!\tilde{z}.P_2 + M_2$ for the same channel x and with the same number of arguments $|\tilde{y}| = |\tilde{z}|$. The sender hands over its arguments $\tilde{z}$ to the receiver and continues with P_2, while the receiver replaces its formal parameters $\tilde{y}$ by $\tilde{z}$ and continues with $P_1[\tilde{z}/\tilde{y}]$. All alternative choices in M_1 and M_2 are discarded. The whole step may be performed with priority r contributed by the sender. A communication error (E.COM) is raised, if two matching prefixes on the same channel x offer different arities $|\tilde{y}| \neq |\tilde{z}|$. Here we write $\perp$ for an arbitrary erroneous expression. A single application step (APP) replaces a defined process by its definition. We assume that there exists a unique definition for all defined processes.

Fig. 6 provides the closure rules for these relations, and defines the global reduction relation $\rightarrow$ between processes, which depend globally on the set of all potential interactions in the population. Communication and error steps are closed under structural congruence (STRUCT), and permitted under parallel composition (PAR) and new binders (NEW). Rule (PRIOR) states that only communication steps with highest available priority may be selected by final reduction

Communication and application steps

$$(\textsc{com})\ \frac{|\tilde{y}| = |\tilde{z}|}{x?\tilde{y}.P_1 + M_1 \mid x{:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{z}/\tilde{y}] \mid P_2} \qquad (\textsc{app})\ \frac{A(\tilde{x}) \triangleq P}{A(\tilde{y}) \xrightarrow[nd]{app} P[\tilde{y}/\tilde{x}]}$$

Program errors

$$(\textsc{e.com})\ \frac{|\tilde{y}| \neq |\tilde{z}|}{x?\tilde{y}.P_1 + M_1 \mid x{:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{err} \bot}$$

Fig. 5. Axioms of operational semantics of π-calculus with priorities.

Structural rules where $\alpha \in \{err, app\} \cup R$

$$(\textsc{par})\ \frac{P_1 \xrightarrow[nd]{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow[nd]{\alpha} P_1' \mid P_2} \quad (\textsc{new})\ \frac{P \xrightarrow[nd]{\alpha} P'}{(\nu x)P \xrightarrow[nd]{\alpha} (\nu x)P'} \quad (\textsc{struc})\ \frac{P \equiv P_1 \quad P_1 \xrightarrow[nd]{\alpha} P_2 \quad P_2 \equiv P'}{P \xrightarrow[nd]{\alpha} P'}$$

Error-free convergence of application

$$(\textsc{conv})\ \frac{P \xrightarrow[nd]{app}{}^{*} P' \quad P' \equiv (\nu\tilde{x}) \prod_{i=1}^{n} M_i \quad \neg P' \xrightarrow[nd]{err} \bot}{P \Downarrow P'}$$

Reduction $(r \in R)$

$$(\textsc{prior})\ \frac{P \Downarrow P' \quad P' \xrightarrow[nd]{r} Q \quad \neg\exists r_1 \in R.\exists Q_1.\ r < r_1 \wedge P' \xrightarrow[nd]{r_1} Q_1}{P \rightarrow Q}$$

Fig. 6. Rules of operational semantics of π-calculus with priorities in $(R, <)$.

relation $\rightarrow$. The set of all communication prefixes becomes apparent only after having applied definitions exhaustively (CONV). Application may not terminate such as for $A()$ if defined by $A() \triangleq A()$. Such nonterminating definitions block all potential subsequent communication steps. Similarly communication errors $P \xrightarrow[nd]{err} \bot$ block all communication steps on P. Finally note, that we define the reflexive transitive closure $(\xrightarrow[nd]{app})^*$ used in rule (CONV) such that it contains the structural congruence $\equiv$.

Example 1. We consider the example of forwarders $Fwd(x, y)$ which receive some value on channel x and forward it to channel y. Forwarders can be used to let objects flow along lists, such as RNAP polymerases along DNA sequences. We assume two levels of priorities $low < high$ and give highest priority to forwarding actions.

$$Fwd(x, y) \triangleq x?(z).(y{:}high!(z).\mathbf{0} \mid Fwd(x, y))$$

We first use forwarders in order to define a list with two elements, which an object z traverses.

$$List_2() \triangleq x_1{:}low!(z).\mathbf{0} \mid Fwd(x_1, x_2) \mid Fwd(x_2, x_3)$$

Process $List_2()$ can be reduced as follows:

$$\begin{aligned} List_2() &\rightarrow Fwd(x_1, x_2) \mid x_2{:}high!(z).\mathbf{0} \mid Fwd(x_2, x_3) \\ &\rightarrow Fwd(x_1, x_2) \mid Fwd(x_2, x_3) \mid x_3{:}high!(z).\mathbf{0} \end{aligned}$$

Beside lists, we can construct rings or other cyclic data structures from forwarders:

$$Ring_3() \triangleq x_1{:}low!(z).\mathbf{0} \mid x_2{:}low!(z).\mathbf{0} \mid Fwd(x_1, x_2) \mid Fwd(x_2, x_3) \mid Fwd(x_3, x_1)$$

One of the two z objects is turning around in the ring forever, while the other can never enter the ring, since entering actions are given lower priority.

$$\begin{aligned} Ring_3() &\rightarrow x_1{:}low!(z).\mathbf{0} \mid Fwd(x_1, x_2) \mid Fwd(x_2, x_3) \mid x_3{:}high!(z).\mathbf{0} \mid Fwd(x_3, x_1) \\ &\rightarrow x_1{:}low!(z).\mathbf{0} \mid x_1{:}high!(z).\mathbf{0} \mid Fwd(x_1, x_2) \mid Fwd(x_2, x_3) \mid Fwd(x_3, x_1) \\ &\rightarrow x_1{:}low!(z).\mathbf{0} \mid Fwd(x_1, x_2) \mid x_2{:}high!(z).\mathbf{0} \mid Fwd(x_2, x_3) \mid Fwd(x_3, x_1) \\ &\rightarrow \dots \end{aligned}$$

2.4 Convergence

We show that the order of application steps does not matter, so that the result of exhaustive application of definitions is always unique. Otherwise, the approach presented here would be computationally unfeasible and thus useless in practice.

Let R be a binary relation between processes. We define the reflexive transitive closure of R up to structural congruence by $R^* = \cup_{i=0}^{\infty} R^i$ where R^0 is the structural congruence $\equiv$ and $R^{i+1} = R \circ R^i$ for all $i \in \mathbb{N}$, and write R^{-1} for the inverse relation. We call R confluent modulo structural congruence, if $(R^*)^{-1} \circ R^* \subseteq R^* \circ (R^*)^{-1}$.

Lemma 1. *The rewrite relation $\xrightarrow[nd]{app}$ is confluent modulo structural congruence; thereby irreducible processes are congruent to processes $(\nu\tilde{x}) \prod_{i=1}^{n} M_i$.*

Proof. The lemma relies on the fact, that we assume a unique definition for every defined process, and that the order of application of these definitions does not matter. We start with a standard analysis of the structural congruence.

Claim. Let $P = (\nu\tilde{x}) \prod_{i=1}^{n} P_i$ be a prenex normal form in which all bound variables are renamed apart, and such that all P_i are sums or defined processes. In this case, $P \xrightarrow[nd]{app} P'$ if and only if the following rule applies:

$$\frac{1 \leq j \leq n \quad P_j = A_j(\tilde{z}_j) \quad A_j(\tilde{y}_j) \triangleq Q_j \quad P' \equiv (\nu\tilde{x})(\prod_{i=1, i \neq j}^{n} P_i \mid Q_j[\tilde{z}_j/\tilde{y}_j])}{P \xrightarrow[nd]{app} P'}$$

Application defines a relation on equivalence classes of processes modulo structural congruence, such that $[P]_\equiv \xrightarrow[nd]{app} [P']_\equiv$ if $P \xrightarrow[nd]{app} P'$. The above claim shows that application terminates on equivalence classes of processes of the form $[(\nu\tilde{x})\prod_{i=1}^{n} M_i]_\equiv$, since we assume that there exists at least one definition for every defined process. We next show that application on equivalence classes is uniformly confluent [36], i.e., if $P \xrightarrow[nd]{app} P_1'$ and $P \xrightarrow[nd]{app} P_2'$ then $P_1' \equiv P_2'$ or there exists P'' such that $P_1' \xrightarrow[nd]{app} P''$ and $P_2' \xrightarrow[nd]{app} P''$. Uniform confluence implies strong confluence and thus confluence [37]. To see uniform confluence, we assume that $P \xrightarrow[nd]{app} P_1'$ and $P \xrightarrow[nd]{app} P_2'$ and let j_1 and j_2 be the positions of the respective reduction step (according to the above rule). If $j_1 = j_2$ then $P_1' \equiv P_2'$, since we assume that there exists at most one definition for every defined process. Otherwise if $j_1 \neq j_2$, then we can set P'' to $(\nu\tilde{x})(\prod_{i=1, i\notin\{j_1,j_2\}}^{n} P_i \mid Q_{j_1}[\tilde{z}_{j_1}/\tilde{y}_{j_1}] \mid Q_{j_2}[\tilde{z}_{j_2}/\tilde{y}_{j_2}])$. □

There exists processes P that do not converge to any P' since the application of process definitions does not terminate. Our semantics ensures that such processes cannot be reduced any further, even though they might not contain an immediate error $P \xrightarrow[nd]{err} \bot$. For instance, consider the process $A()$ with the following definition $A() \triangleq A()$ that is not well-founded. An implementation may either run into an infinite loop unfolding the definition of A repeatedly, or report the erroneous cycle.

Proposition 1 (Convergence). *For all process P there exists at most one class $[P']_\equiv$ such that $P \Downarrow P'$.*

Proof. This follows immediately from the confluence result in Lemma 1.

This proposition states that the idea of exhaustive application of definitions is consistent, in that the result does not depend of the application order. There are three possible kinds of results: convergence, arity mismatches, and non-termination of application.

Remark 1. If $P \equiv (\nu\tilde{x})\prod_{i=1}^{n} M_i$ and $\neg P \xrightarrow[nd]{err} \bot$ then $P \equiv P' \Leftrightarrow P \Downarrow P'$.

Proof. Suppose that $P \equiv (\nu\tilde{x})\prod_{i=1}^{n} M_i$ and $\neg P \xrightarrow[nd]{err} \bot$. If $P \equiv P'$ then $P \xrightarrow[nd]{app}{}^{*} P'$ by definition of reflexivity so that $P \Downarrow P'$. Conversely, suppose that $P \Downarrow P'$. By definition of convergence, this implies $P \xrightarrow[nd]{app}{}^{*} P'$, which yields $P \equiv P'$ since $[P]_\equiv$ is irreducible with respect to $\xrightarrow[nd]{app}$ by Lemma 1.

2.5 Stochastic Operational Semantics

We present a stochastic operational semantics for the π-calculus with priorities, under the assumption that stochastic rates in $\mathbb{R}_+^\infty$ are used as priorities with two levels, the lower level for numbers in $\mathbb{R}_+$ and the higher for ∞.

Labeled communication steps ($r \in \mathbb{R}_+^\infty$ and $\ell \in \mathbb{N}^4$)

$$(\text{COM}_\ell)\quad \frac{\begin{array}{c}\ell = (i_1, j_1, i_2, j_2) \quad i_1 \neq i_2 \\ \pi_{i_1}^{j_1} = x?\tilde{y} \quad \pi_{i_2}^{j_2} = x{:}r!\tilde{z} \quad |\tilde{y}| = |\tilde{z}|\end{array}}{(\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \;\underset{\ell}{\xrightarrow{r}}\; (\nu\tilde{x})(\prod_{i=1, i\neq i_1, i_2}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \mid P_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid P_{i_2}^{j_2})}$$

Fig. 7. Axioms of operational semantics of stochastic π-calculus.

Markov chain ($r, r' \in \mathbb{R}^+$)

$$(\text{SUM})\quad \frac{P \Downarrow P_1 \quad \sum_{\{(r',\ell) \mid P_1 \underset{\ell}{\xrightarrow{r'}} P_2 \equiv P'\}} r' = r \neq 0 \quad \neg\exists\ell\exists P''.P_1 \underset{\ell}{\xrightarrow{\infty}} P''}{P \xrightarrow{r} P'}$$

$$(\text{COUNT})\quad \frac{P \Downarrow P_1 \quad n = \sharp\{\ell \mid P_1 \underset{\ell}{\xrightarrow{\infty}} P_2 \equiv P'\} \neq 0}{P \xrightarrow{\infty(n)} P'}$$

Fig. 8. Rules of the stochastic semantics of the π-calculus with priorities.

In contrast to most previous approaches, the syntax of processes remains without change. This means in particular, that stochastic rates are annotated to output prefixes rather than to channel names as in [12, 21, 20], or to both input and output prefixes [19].

The stochastic semantics of a process P in the stochastic π-calculus is a continuous time Markov chain (CTMC). The states of such CTMCs are classes of processes $[P]_\equiv$. A priori, the state space may be infinite, even though only finitely many states may be reachable in many cases. The purpose of the transitions of a CTMC is to define the probability of a reduction step $P \to P'$. We will use transitions $P \xrightarrow{r} P'$ that are labeled by propensities $r \in \mathbb{R}_+^\infty$. If r is finite, then the probability of a reduction step from P to P' is r/s, where s is the sum of all propensities r' of transitions starting in P. If s is infinite, then the transition is impossible, since a transition exists with infinite rate which has priority.

The probability of a reduction step follows the *Chemical Law of Mass Action* according to which the propensity of a chemical reaction in a solution is proportional to the number of possible interactions of its reactants in the solution (when assuming a fixed volume). Given a source process P and a target process P', the propensity of $P \to P'$ depends on the number of ways in which P may reduce to P'. For instance, consider $P_1 = x?().\mathbf{0}$ and $P_2 = x{:}r!().\mathbf{0}$ for some rate $r \in \mathbb{R}_+$. If we fix $P = P_1 \mid P_1 \mid P_2$ and $P' = P_1$ then we have two possible interactions of rate r, so we have $P \xrightarrow{2r} P'$ where $2r$ is the propensity of this reaction.

In order to discriminate interactions leading to the same state, rule (COM$_\ell$) in Fig. 7 defines communication steps labeled by positions $\ell \in \mathbb{N}^4$, where the interaction occurs. Given a prenex normal form $P = (\nu\tilde{x}) \prod_{i=1}^{n} \sum \pi_i^j.P_i^j$, a tuple $\ell = (i_i, j_1, i_2, j_2)$ defines the pair of communication prefixes $\pi_{i_1}^{j_1}.P_{i_1}^{j_1}$ and $\pi_{i_2}^{j_2}.P_{i_2}^{j_2}$. As before, a communication step can only be applied to senders and receivers on the same channel. We write $P \xrightarrow[\ell]{r} P'$ if there exists a potential interaction at position ℓ, where r is the rate annotated to the sender.

The transitions of the CTMC are defined in Fig. 8. Transitions $[P]_\equiv \xrightarrow[nd]{r} [P']_\equiv$ with finite propensities $r \in \mathbb{R}_+$ are obtained by rule (SUM). First, convergence of P with respect to application is tested. If this test fails then no transition is possible. Otherwise, the unique equivalence class $[P_1]_\equiv$ is computed such that $P \Downarrow P_1$. Second, an arbitrary representative in prenex normal form P_1 of this congruence class is fixed. Third, all pairs (r', ℓ) of P are computed such that there exists $P_2 \equiv P'$ and a communication step $P_1 \xrightarrow[\ell]{r'} P_2$. Finally, all such rates r' are summed up into propensity r. Going back to our previous example, we have $P_1 \mid P_1 \mid P_2 \xrightarrow[(1,1,3,1)]{r} P_1$ and $P_1 \mid P_1 \mid P_2 \xrightarrow[(2,1,3,1)]{r} P_1$, so that $P_1 \mid P_1 \mid P_2 \xrightarrow{2r} P_1$ as expected.

Communication steps with infinite propensities are treated by rule (COUNT). These are given highest priority as stated already in rule (SUM). The probability of a reduction $P \xrightarrow{\infty(n)} P'$ is n/m where n is the number of interactions with rate ∞ leading from P to a process congruent to P', and m the overall number of interactions with rate ∞ starting from P. Given these probabilities, and provided that no infinite sequence of immediate transitions is reachable, one can build a reduction, without immediate transitions, that defines a proper CTMC and preserves the *probabilities of transitions* and *sojourn times* (see e.g. [10] for details).

For illustration, consider a system of two chemical reactions, $x : A, B \xrightarrow{0.5} A, C$ and the inverse $y : A, C \xrightarrow{5} A, B$ whose rate is 10-fold higher. In Fig. 9, we define species A, B, C as processes in the stochastic π-calculus that act according to these chemical reactions. A chemical solution with species A, B, C is a multiset of molecules, i.e. a multiset with these species. In the π-calculus, it can be expressed by a parallel compositions of defined processes, such as for instance $A^2 \mid B^2 \mid C^1$, where we write P^n instead of $\prod_{i=1}^{n} P$. The reachable part of the CTMC of this chemical system is shown in Fig. 9.

The stochastic semantics of the π-calculus with priorities does indeed properly refine the non-deterministic operational semantics.

Proposition 2. *If the set of priorities $(R, <)$ is equal to $(\mathbb{R}_+^\infty, <_2)$ where $<_2$ defines the usual two levels of priorities (i.e. $r <_2 \infty$ for all $r \in \mathbb{R}_+$), then for all processes P, Q:*

$$P \to Q \textit{ iff } (\exists r \in \mathbb{R}_+ : P \xrightarrow{r} Q \vee \exists n \in \mathbb{N} : \; P \xrightarrow{\infty(n)} Q)$$

Chemical reactions:

$$x : A, B \xrightarrow{0.5} A, C$$
$$y : A, C \xrightarrow{5} A, B$$

π-calculus definitions:

$$A \triangleq x{:}0.5!().A + y{:}5!().A$$
$$B \triangleq x?().C$$
$$C \triangleq y?().B$$

Transitions of CTMC fragment reachable from $A^2 \mid B^2 \mid C^1$:

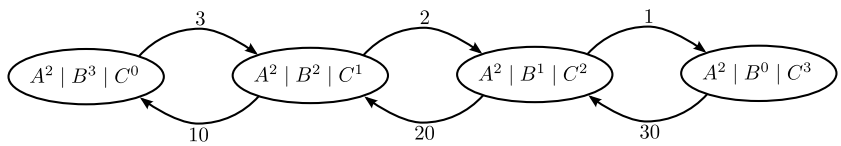


Fig. 9. Example of a CTMC generated by the stochastic π-calculus.

Proof. The implication from the right to the left is quite obvious, since $P \xrightarrow[\ell]{r} Q$ implies $P \to Q$. For the direction from the left to the right, we start with a claim that relates communication steps to labeled communication steps in this direction:

Claim. If $P_1 \xrightarrow[nd]{r} Q$ and $P_1 = (\nu x) \prod_{j=1}^{n} \sum_{i=1}^{m_j} \pi_i^j.P_i^j$ then there exists a label $\ell = (i_1, j_1, i_2, j_2)$ and a process Q' such that $Q' \equiv Q$ and $P_1 \xrightarrow[\ell]{r} Q'$.

This follows from a standard analysis of the structural congruence. Suppose now, that $P \to Q$ holds. In this case, the following rule must be applicable:

$$(\text{PRIOR})\ \frac{P \Downarrow P_1 \quad P_1 \xrightarrow[nd]{r} Q \quad \neg\exists r_1 \in R.\exists Q_1.\ r < r_1 \wedge P_1 \xrightarrow[nd]{r_1} Q_1}{P \to Q}$$

Without loss of generality, we can assume that P_1 is in prenex normal form, since relation $\xrightarrow[nd]{r}$ is closed under structural congruence by rule (STRUCT). The second hypothesis and the above claim show that $P_1 \xrightarrow[\ell]{r} Q'$ for some process Q' with $Q' \equiv Q$. The third hypothesis holds if and only if either $r = \infty$ or else $r \in \mathbb{R}_+$ and $\neg\exists Q_1.\ P_1 \xrightarrow[nd]{\infty} Q_1$.

- In the case $r = \infty$, we can create a transition with infinite propensity:

$$(\text{COUNT})\ \frac{P \Downarrow P_1 \quad n = \sharp\{\ell \mid P_1 \xrightarrow[\ell]{\infty} Q' \equiv Q\} \neq 0}{P \xrightarrow{\infty(n)} Q}$$

- In the case $r \in \mathbb{R}_+$, property $P_1 \xrightarrow[\ell]{r} Q'$ shows that $\sum_{\{(r',\ell) \mid P_1 \xrightarrow[\ell]{r'} Q' \equiv Q\}} r' \neq 0$.

  We can thus create a transition of the Markov chain with finite propensity:

$$(\text{SUM})\ \frac{P \Downarrow P_1 \quad \sum_{\{\ell \mid P_1 \xrightarrow[\ell]{r'} Q' \equiv Q\}} r' = r \neq 0 \quad \neg\exists\ell\exists Q_1.P_1 \xrightarrow[\ell]{\infty} Q_1}{P \xrightarrow{r} Q}$$

□

2.6 Type System

We present a type system for the π-calculus with priorities, that prevents from arity mismatches in communication attempts as defined by rule (E.COM). Non-immediate errors, like nonterminating applications of unguarded process definitions are not captured though. These can be detected by a simple cycle check.

Channels are the only values over our calculus. In order to exclude arity mismatches on channels, we introduce channel types, that fix the types of all arguments that can be communicated:

$$\text{types} \qquad \tau ::= ch(\tilde{\tau})$$

A channel of type $ch(\tilde{\tau})$ may only be used to receive and send tuples of values of type $\tilde{\tau}$. A defined process $A(\tilde{x})$ of process type $\tilde{\tau}$ must receive arguments of type $\tilde{\tau}$ for $\tilde{x}$. In our typed setting, we assume that channel creation is typed, by adding types into the syntax of new operators:

$$\text{typed processes} \qquad P ::= (\nu x{:}\tau)P \ \mid \ \ldots$$

A *type environment* Γ is a set of type assignments from channel names to types $x{:}\tau$ and from process names to tuples of types $A{:}\tilde{\tau}$. Thereby we fix the types of the arguments of parametrized process definitions.

Example 2. Consider the process $P = x{:}r!(z).z?(y).P_1 \mid x?(y).y{:}r!().P_2$. The arities of the sender and receiver for x coincide in that they both have 1 argument. After communication, however, P becomes $z?(y).P_1 \mid z{:}r!().P_2$ which has an arity mismatch on z. These kinds of situations are excluded for well-typed processes, so P cannot be well-typed. Indeed, the first subprocess of P is well-typed for type environments containing $x{:}ch(ch(\tau)), z{:}ch(\tau)$ for some type τ. The second subprocess of P, requires type environments containing $x{:}ch(ch())$. Both conditions together are unsatisfiable.

Example 3. Processes $List_2$ and $Ring_3$ are well-typed in environments, where channel z is given an arbitrary type, say $\tau = ch()$, while process Fwd must be assigned type $(ch(\tau), ch(\tau))$. Furthermore the three channels x_1, x_2, x_3 that connect the forwarders must be of type $ch(\tau)$ too. Valid type environments Γ for $Ring_3$ thus must contain the following assumptions:

$$z{:}\tau, Fwd{:}(ch(\tau), ch(\tau)), List_2{:}(), Ring_3{:}(), x_1{:}ch(\tau), x_2{:}ch(\tau), x_3{:}ch(\tau)$$

Type checking rules for processes are given in Fig. 10. They infer judgments $\Gamma \vdash P$ that state the consistency of a process P with a type environment Γ as usual. Inconsistencies may arise from arity mismatches of receivers (T.REC), sender (T.SEND), process applications (T.APP), and process definitions (T.DEF). Furthermore, there are three structural rules, and a rule for the null process $\mathbf{0}$.

$$(\text{T.REC})\ \frac{\Gamma \vdash x{:}ch(\tilde{\tau}) \quad \Gamma, \tilde{y}{:}\tilde{\tau} \vdash P}{\Gamma \vdash x?\tilde{y}.P} \qquad (\text{T.SEND})\ \frac{\Gamma \vdash x{:}ch(\tilde{\tau}) \quad \Gamma \vdash \tilde{z}{:}\tilde{\tau} \quad \Gamma \vdash P}{\Gamma \vdash x{:}r!\tilde{z}.P}$$

$$(\text{T.PAR})\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad (\text{T.SUM})\ \frac{\Gamma \vdash M \quad \Gamma \vdash M'}{\Gamma \vdash M + M'} \qquad (\text{T.NEW})\ \frac{\Gamma, x{:}\tau \vdash P}{\Gamma \vdash (\nu x{:}\tau)P}$$

$$(\text{T.APP})\ \frac{\Gamma \vdash A{:}\tilde{\tau} \quad \Gamma \vdash \tilde{x}{:}\tilde{\tau}}{\Gamma \vdash A(\tilde{x})} \qquad (\text{T.DEF})\ \frac{\Gamma \vdash A{:}\tilde{\tau} \quad \Gamma, \tilde{x}{:}\tilde{\tau} \vdash P}{\Gamma \vdash A(\tilde{x}) \triangleq P} \qquad (\text{T.NIL})\ \frac{}{\Gamma \vdash \mathbf{0}}$$

Fig. 10. Type system for π-calculus with priorities.

Proposition 3 (Type safety). *If $\Gamma \vdash P$ and $P \to Q$ then $\Gamma \vdash Q$.*

The proof works as usual. See the proof of Theorem 1 for a more general instance.

Corollary 1 (Error freeness). *If $\Gamma \vdash P$ and $P \to^* Q$ then $\neg Q \xrightarrow[nd]{err} \bot$.*

Proof. Assuming $\Gamma \vdash P$ and $P \to^n Q$, the proof is by induction on n. The inductive step follows from Proposition 3; it thus remains to prove the initial case that is $Q = P$. We proceed by contradiction assuming that there exists some process P_0 such that $\Gamma \vdash P_0$ and $P_0 \xrightarrow[nd]{err} \bot$.

As for the proof of Lemma 1, a standard analysis of the structural congruence shows the following claim: $P_0 \equiv (\nu\tilde{x}{:}\tilde{\tau}) \prod_{i=1}^{n} P_i$ be a prenex normal form in which all bound variables are renamed apart, and such that all P_i are sums or defined processes, and $\exists j, k. 1 \leq j < k \leq n$, $P_j = x_0?\tilde{y}.Q_1 + M_1$, $P_k = x_0{:}r!\tilde{z}.Q_2 + M_2$, and $|\tilde{y}| \neq |\tilde{z}|$.

From $\Gamma \vdash P_0$, we have $\Gamma \vdash (\nu\tilde{x}{:}\tilde{\tau}) \prod_{i=1}^{n} P_i$. This statement follows from a series of applications of rules (T.NEW) and (T.PAR) and from statements $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash P_i$ for all $i \in \{1, \ldots, n\}$. In particular, $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash x_0?\tilde{y}.Q_1 + M_1$, and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash x_0{:}r!\tilde{z}.Q_2 + M_2$. Therefore, by rules (T.REC) and (T.SEND), $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash x_0{:}ch(\tilde{\tau})$ and, $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \tilde{z}{:}\tilde{\tau}$ and, $\Gamma, \tilde{x}{:}\tilde{\tau}, \tilde{y}{:}\tilde{\tau} \vdash Q_1$, thus $|\tilde{z}| = |\tilde{\tau}|$ and $|\tilde{y}| = |\tilde{\tau}|$ which contradicts $|\tilde{y}| \neq |\tilde{z}|$. □

3 Attributed π-Calculus with Priorities

We introduce the attributed π-calculus with priorities $\pi(\mathcal{L})$, by extending the π-calculus with priorities with richer sets of values and expressions in some call-by-value λ-calculus $\mathcal{L}$ that we call the attribute language $\mathcal{L}$. We permit λ-expressions in generalized senders and receivers, in order to impose constraints on communication steps, subsuming priorities and stochastic rates. As before, we will present both a non-deterministic and a stochastic operational semantics (except that the set of successful values must be $\mathbb{R}_+^\infty$ with 2 levels of priorities in the stochastic case).

3.1 Idea of Communication Constraints

For illustration of communication constraints, we reconsider the example from the introduction. Proteins $\mathit{Prot}(x)$ can bind to operators $\mathit{Op}(y)$ only if they have equal sorts $x{=}y$. Expressing such constraints in object-centered languages such as the π-calculus is difficult, since it concerns the attribute values of two independent processes. Our solution proposed for the attributed π-calculus is to use functions such as $\lambda x.x{=}y$ on receiver side, and to apply them to the value of x on the sender side:

$$\begin{aligned}&\mathit{Prot}(x) \triangleq \mathit{bind}[x]!().0\\&\mathit{Op}(y) \triangleq \mathit{bind}[\lambda x.x{=}y]?().\mathit{OpBound}(y)\end{aligned}$$

These definitions allow reduction steps $\mathit{Prot}(\mathtt{b}) \mid \mathit{Op}(\mathtt{b}) \rightarrow \mathit{OpBound}(\mathtt{b})$ for arbitrary values $\mathtt{b}$, since the application $(\lambda x.x{=}\mathtt{b})\mathtt{b}$ evaluates to $\mathtt{true}$. In order to permit richer sets of values and constraints, such as for instance arithmetic values and constraints, we define use call-by-value λ-calculi as attribute languages $\mathcal{L}$. We keep the choice of constants parametric, in order to avoid reinventing independent calculi for the many useful choices in practice. We will define our semantics such that they are independent of the concrete choice of the attribute language.

3.2 Attribute Languages

An attribute language is a functional programming language that provides expressions by which to compute values. Expressions are built from constants – for numbers, functions, relations, or biological entities (such as 0, 1, $+$, $*$, $\geq$, $\mathtt{fst}$, $\mathtt{snd}$, $\mathtt{repressor}$, ...) – and from variables $x, y \in \mathit{Vars}$, which may play the name channels in particular. Whenever ambiguities might arise, we typeset constants in courier font (such as $\mathtt{fst}$ or $\mathtt{repressor}$) and variables in italics such as bind.

As an example, consider the expression $(\mathtt{snd}\ x) + (\mathtt{fst}\ y)$ in which $\mathtt{fst}$, $\mathtt{snd}$, and $+$ are constants, while x and y are variables. In the following process definition, this expression defines a stochastic rate or priority: $A(x, y, z) \triangleq z[(\mathtt{snd}\ x) + (\mathtt{fst}\ y)]!().P$.

An attribute language with variables in Vars is a tuple $\mathcal{L} = (\mathit{Consts}, \Downarrow, R, <)$, that contains a set of constants $c \in \mathit{Consts}$, a big-step evaluator $\Downarrow$ for λ-expressions with constants, pairs, and conditionals, and a partially ordered set $(R, <)$ of successful values, that enables communication steps. More precisely, the first component Consts is a finite set that fixes the constants of a call-by-value λ-calculus. The set of expressions $e \in \mathit{Exprs}$ of $\mathcal{L}$ is defined as the set of all λ expressions with constants in $c \in \mathit{Consts}$ and variables in $x \in \mathit{Vars}$. The set of values $v \in \mathit{Vals}$ of $\mathcal{L}$ is the subset of all values of this λ-calculus.

$$\begin{aligned}c \in \mathit{Consts} &::= \mathtt{false} \mid \mathtt{true} \mid \mathtt{fst} \mid \mathtt{snd} \mid \ldots\\ v \in \mathit{Vals} &::= x \mid c \mid \lambda x.e \mid \langle v_1, v_2\rangle\\ e \in \mathit{Exprs} &::= v \mid e_1 e_2 \mid \langle e_1, e_2\rangle \mid \mathtt{if}\ e\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2\end{aligned}$$

$$(\text{VAL})\ \frac{v \in \mathit{Vals}}{v \Downarrow v} \qquad (\text{FUN})\ \frac{e_1 \Downarrow \lambda x.e_1' \quad e_2 \Downarrow v' \quad e_1'[v'/x] \Downarrow v}{e_1 e_2 \Downarrow v}$$

$$(\text{PAIR})\ \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\langle e_1, e_2 \rangle \Downarrow \langle v_1, v_2 \rangle} \qquad (\text{SELECT})\ \frac{e \Downarrow \langle v_1, v_2 \rangle}{\texttt{fst}\ e \Downarrow v_1 \qquad \texttt{snd}\ e \Downarrow v_2}$$

$$(\text{COND}_1)\ \frac{e \Downarrow \texttt{true} \quad e_1 \Downarrow v_1}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v_1} \qquad (\text{COND}_2)\ \frac{e \Downarrow \texttt{false} \quad e_2 \Downarrow v_2}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v_2}$$

Fig. 11. Big-step evaluators of call-by-value λ-calculus with pairs and conditionals.

$$(\text{EQ}_1)\ \frac{e_1 \Downarrow v \quad e_2 \Downarrow v \quad v \in \mathit{Vars} \cup \mathit{Consts}}{e_1 {=} e_2 \Downarrow \texttt{true}} \qquad (+_{\mathbb{N}})\ \frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2 \quad n_1 +_{\mathbb{N}} n_2 = n}{e_1 + e_2 \Downarrow n}$$

$$(\text{EQ}_2)\ \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 \neq v_2 \in \mathit{Vars} \cup \mathit{Consts}}{e_1 {=} e_2 \Downarrow \texttt{false}}$$

Fig. 12. Additional rules of big-step evaluator of the attribute language $\lambda(\mathbb{N}_0, +, =)_{<_1}$.

As usual, λ-expressions provide abstractions $\lambda x.e$ and applications $e_1 e_2$ for specifying function definitions and function application. We assume that the set of constants *Consts* contains the Booleans `true` and `false`, and pair projections `fst` and `snd`. We also assume that there are expressions for pairs $\langle e_1, e_2 \rangle$ and Boolean conditionals `if` e `then` e_1 `else` e_2. In examples, we will freely write `if` e `then` e_1 instead of `if` e `then` e_1 `else false`.

The third component is a set $R \subseteq \mathit{Vals}$ of successful values enabling communication steps, such as priorities or stochastic rates. The fourth component $<$ is a partial order on successful values R that defines priorities. The last component of $\mathcal{L}$ is a big-step evaluator λ expressions, i.e. a partial function $\Downarrow : \mathit{dom}(\Downarrow) \to \mathit{Vals}$ from λ-expressions in a domain $\mathit{dom}(\Downarrow) \subseteq \mathit{Exprs}$ to value. It can be understood as a black box algorithm that evaluates all expressions to values or failure, in case of program errors or nontermination. Instead of $\Downarrow(e) = v$ we will write $e \Downarrow v$ and call v the value of e.

We assume that the big-step evaluator satisfies the usual rules of the call-by-value λ-calculus with conditionals in Fig. 11. Rule (VAL) states that all values evaluate to themselves. Rule (FUN) defines the usual meaning of call-by-value function application. It says that the value of an application $e_1 e_2$ is obtained by by evaluating e_1 to some function $\lambda x.e_1'$ and e_2 to some value v', and then returning the value of $e_1'[v'/x]$. Rule (PAIR) states that the value of a pair is the pair of values of its components (as usual in call-by-value languages). Rule (SELECT) states the usual meaning of pair selectors. Rule (COND_1) and (COND_2) defines the semantics of conditionals such that only the needed branch is evaluated. For richer attribute language with further constants (such as $+$, $*$, or call-by-value fixed point operator `fix`), we need to add further rules. This is possible, since we keep the big-step evaluator abstract.

As a first example, we consider the attribute language $\lambda(R)_<$ for some partially ordered set $(R, <)$ of priorities. These are the only successful values and ordered according to $<$. No new rules are needed for the big-step evaluator, since no function constants are added.

As a second example, we introduce the attribute language $\lambda(\mathbb{N}_0, +, =)_{<_1}$ where the call-by-value λ-calculus is extended by constants for natural numbers with 0 and addition $+$. The successful values are nonzero natural numbers and there is a single level of priorities, fixed by the empty partial order that we denote as $<_1$. Constant $=$ defines equality on all constants and variables, i.e. on Booleans, natural numbers, channel names, and the function constants. The required extensions of the big-step evaluator are given in Fig. 12. Note that addition and equality are treated as curried binary functions, since introduced by constant. We freely write $e + e'$ instead of $(+\ e)\ e'$ and respectively, $e{=}e'$ instead of $(=\ e)\ e'$.

As third example, we consider the attribute language $\lambda(\mathbb{R}_+^\infty)_{<_2}$, whose successful values are the stochastic rates in $\mathbb{R}_+^\infty$. There are two levels of priorities, lower priority for all positive real numbers in $\mathbb{R}_+$ and higher priority for ∞. We write $<_2$ for the obvious order that introduces these two levels of priorities.

Further extensions of the attribute language might be useful in various applications, such as n-tuples (beyond pairs), lists, or case statements, but cannot be obtained by adding new constants, since they require new forms of expressions and values. Even though we do not expect particular difficulties, we refrain from generalizing or extending the attribute language any further for the sake of simplicity.

Values of expressions may be undefined, i.e. for some expressions e there might not exist any value v with $e \Downarrow v$. This may have two possible reasons. The first are program errors, like division by 0, or type errors, like sending or receiving on non-channels. The second reason is non-termination, which may arise in an untyped setting or in rich attribute languages with fixed point operators.

In Section 3.7 we will present a type system for the attributed π-calculus which prevents us from type errors. If not adding any constants to the attribute language, it even excludes non-termination. For more general attribute languages, however, the type systems may neither exclude all program errors (e.g. division by 0) nor ensure termination (e.g. fixed point operators).

3.3 Syntax of Attributed Processes

Let $\mathcal{L}$ be an attribute language over some infinite set of variables $x \in \mathit{Vars}$, with expressions $e \in \mathit{Exprs}$ and values $v \in \mathit{Vals}$.

The syntax of the attributed π-calculus $\pi(\mathcal{L})$ is defined in Fig. 13. Compared to before, we use variables x of various types instead of channel names (which correspond to variables of channel type), permit expressions e in all non-binding positions where previously only channel names were allowed, extend priorities "$:r$" in senders to expressions "$[e]$", and introduce such expressions in the symmetric position in receivers. Receiver prefixes thus have the form $e_1[e_1']?\tilde{x}$ and sender prefixes the form $e_2[e_2']!\tilde{e}$. Prefixes in which e_1 resp. e_2 do not evaluate to

Prefixes	π ::=	$e_1[e_2]?\tilde{x}$	receiver
	|	$e_1[e_2]!\tilde{e}$	sender
Sums	M ::=	$\pi.P$	guarded process
	|	$M_1 + M_2$	choice
Processes	P ::=	M	sums
	|	$A(\tilde{e})$	defined process
	|	$P_1 \mid P_2$	parallel composition
	|	$(\nu x)P$	channel creation
	|	$\mathbf{0}$	empty solution
Definitions	D ::=	$A(\tilde{x}) \triangleq P$	parametric process definition

Fig. 13. Syntax of $\pi(\mathcal{L})$ where $x, \tilde{x} \in \mathit{Vars}$, and $e_1, e_2, \tilde{e} \in \mathit{Exprs}$.

channels are erroneous. The application $e_1'e_2'$ imposes a constraint on the ability to communicate, in addition to that e_1 and e_2 must evaluate to the same channel. Communication is permitted only if $e_1'e_2' \Downarrow v$ for some successful value $v \in R$. This value then fixes the priority or stochastic rates of the communication step.

For illustration, we consider 3 instances of the attributed π-calculus with 3 different attribute languages. As a first example, we consider the calculus $\pi(\lambda(R)_<)$ whose attribute language is a λ-calculus with priorities. The second example is $\pi(\lambda(\mathbb{N}_0, +, =)_{<_1})$ which provides for natural numbers with addition and equality. The third examples is $\pi(\lambda(\mathbb{R}_+^\infty)_{<_2})$, where λ-expressions can be used in order to compute priorities on 2 levels.

For illustration, we consider process definitions in $\pi(\lambda(\mathbb{N}_0, +, =)_{<_1})$, which express schemes for chemical reactions *react*: $A(x), B(y) \xrightarrow{x+y} A(x+1), B(y)$ in which x and y can be instantiated by all possible natural numbers. Similar reaction schemes are used in [38], in order to model chemical reactions in biochemical systems[5].

$$A(x) \triangleq \mathit{react}[x]!().A(x{+}1) \qquad B(y) \triangleq \mathit{react}[\lambda x.x + y]?().B(y)$$

The process $A(2)|B(5)$ may communicate on channel z and become $A(3) \mid B(5)$, since the sum of the x-attribute of $A(2)$ and the value of the y-attribute of $B(5)$ is the successful value 7. More formally, we can compute this number by evaluating the interaction constraint $(\lambda x.x + 5)2$ via (FUN), (VAL), and $(+_{\mathbb{N}})$:

$$\frac{\dfrac{\lambda x.x + 5 \in \mathit{Vals}}{\lambda x.x + 5 \Downarrow \lambda x.x + 5} \quad \dfrac{2 \in \mathit{Vals}}{2 \Downarrow 2} \quad \dfrac{\ldots}{2 + 5 \Downarrow 7}}{(\lambda x.x + 5)2 \Downarrow 7}$$

Free variables $\mathit{fv}(P)$ are defined as before, except that we now need to account for free variables $\mathit{fv}(e)$ in λ-expressions e too, i.e., those occurring out of the

[5] In contrast to here, variables in [38] quantify over finite sets of values, while rules there permit more then 2 reactants in contrast to the π-calculus.

Communication and application steps

$$(\textsc{send})\ \frac{e_1 \Downarrow x \quad e_2 \Downarrow v_2 \quad \tilde{e} \Downarrow \tilde{v}}{e_1[e_2]!\tilde{e} \Downarrow x[v_2]!\tilde{v}} \qquad (\textsc{rec})\ \frac{e_1 \Downarrow x \quad e_2 \Downarrow v_2}{e_1[e_2]?\tilde{y} \Downarrow x[v_2]?\tilde{y}} \qquad (\textsc{tup})\ \frac{\wedge_{i=1}^{n} e_i \Downarrow v_i}{(e_i)_{i=1}^{n} \Downarrow (v_i)_{i=1}^{n}}$$

$$(\textsc{com})\ \frac{\pi_1 \Downarrow x[v_1]?\tilde{y} \quad \pi_2 \Downarrow x[v_2]!\tilde{v} \quad v_1 v_2 \Downarrow r \in R \quad |\tilde{v}| = |\tilde{y}|}{\pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{v}/\tilde{y}] \mid P_2} \qquad (\textsc{app})\ \frac{\tilde{e} \Downarrow \tilde{v} \quad A(\tilde{x}) \triangleq P}{A(\tilde{e}) \xrightarrow[nd]{app} P[\tilde{v}/\tilde{x}]}$$

Program errors

$$(\textsc{e.com})\ \frac{\pi_1 \Downarrow x[v_1]?\tilde{y} \quad \pi_2 \Downarrow x[v_2]!\tilde{v} \quad |\tilde{y}| \neq |\tilde{v}|}{\pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{err} \bot}$$

$$(\textsc{e.pref})\ \frac{\neg\exists \pi'.\pi \Downarrow \pi'}{\pi.P + M \xrightarrow[nd]{err} \bot} \qquad (\textsc{e.constr})\ \frac{\pi_1 \Downarrow x[v_1]?\tilde{y} \quad \pi_2 \Downarrow x[v_2]!\tilde{v} \quad \neg\exists v.v_1 v_2 \Downarrow v}{\pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{err} \bot}$$

Fig. 14. Non-deterministic operational semantics of $\pi(\mathcal{L})$ with priorities: all rules of the π-calculus with priorities in Fig. 6 remain valid, too.

scope of all λ binders in e.

$$fv(e_1[e_2]?\tilde{y}.P) = fv(e_1) \cup fv(e_2) \cup (fv(P) \setminus \{\tilde{y}\})$$
$$fv(e_1[e_2]!\tilde{e}.P) = fv(e_1) \cup fv(e_2) \cup fv(\tilde{e}) \cup fv(P) \qquad fv(A(\tilde{e})) = fv(\tilde{e})$$

Bound variables $bv(P)$ are defined as before, except that λ-binders in expressions $e \in \mathit{Exprs}$ are included too. The structural congruence on processes $\equiv$ remains unchanged, except that α-conversion becomes applicable to bound variables in λ-expressions.

3.4 Non-deterministic Operational Semantics

The non-deterministic operational semantics of the attributed π-calculus with priorities is given by the rules in Fig. 14 and the previous rules in Fig. 6. The new rules always evaluate expressions to values before applying communication or application steps (COM) and (APP). This is done by using the big-step evaluator of the attribute language according to axioms (SEND), (REC), and (TUP). Note that evaluation of expressions may get stuck – in contrast to the pure π-calculus with priorities. For instance, an application $A(\tilde{e})$ gets stuck if the evaluation of one of the expressions in $\tilde{e}$ does not succeed. In this case, application does not converge, so that communication gets blocked.

The communication rule (COM) permits receivers $x[v_1]?\tilde{y}.P_1$ and senders $x[v_2]!\tilde{v}.P_2$ to interact only if expression $v_1 v_2$ evaluates to a successful value $v_1 v_2 \Downarrow r \in R$. This value defines the priority level of the communication step. Communication steps perform substitutions $[\tilde{v}/\tilde{y}]$ replacing variables by values. The application of substitutions is well-defined for all processes, since our syntax permits values in all positions, where free variables may be used. Note however,

that substitution may raise program errors as specified by rule (E.PREF), where non-channel values arise in sender or receiver position. As before, we write $\perp$ for an arbitrary erroneous expression. Rule (E.CONSTR) specifies constraint errors, where the evaluation of communication constraints $v_1 v_2$ fails.

The closure rules in Fig. 6 remain unchanged. As before, all relations are closed under the structural rules, while (CONV) applies definitions exhaustively and continues to require error-freeness. The overall reduction relation $P \to P'$ is defined in rule (PRIOR) without change. All changes are imported from the changes in communication, application, and error steps.

Example 4. Let us consider a client server system in the attributed π-calculus with integers and strings and two levels of priorities $\pi(\lambda(\texttt{Int}, \texttt{String})_{<_2})$, such that there are two successful values $R = \{1, 2\}$ ordered by least ordering $<_2$ that satisfies $1 <_2 2$. We fix a value $never =_{df} 0$ that is not successful and name the two successful values as follows: $low =_{df} 1$ and $high =_{df} 2$. Furthermore, let function $price : \texttt{String} \to \texttt{Int}$ be defined by the following expression:

$$price =_{df} \lambda x.\texttt{if } x{=}\texttt{chicken then } 10 \texttt{ else if } x{=}\texttt{fish then } 14 \texttt{ else } 0$$

Servers are accessible on a public channel *connect* to all clients that know a password *key* of type `String`. The server applies function *price* to a string value received from the client and returns the value on a private channel, that was also provided by the client. We define servers and clients as follows:

$$\begin{aligned} &Server() \triangleq connect[\lambda k.\texttt{if } k{=}key \texttt{ then } low \texttt{ else } never]?(x, ret). \\ &\qquad\qquad\qquad\qquad\qquad\qquad (ret[high]!(price\ x).\mathbf{0}) \mid Server()) \\ &Client(s) \triangleq (\nu ret) connect[key]!(s, ret).ret[\lambda z.z]?(y).P \end{aligned}$$

We can then reduce a process with two clients and one server as follows:

$$\begin{aligned} &Server() \mid Client(\texttt{chicken}) \mid Client(\texttt{fish}) \\ \to\ &(\nu ret)(ret[high]!(price\ \texttt{chicken}).\mathbf{0} \mid Server() \mid ret[\lambda x.x]?(y).P) \mid Client(\texttt{fish}) \\ \equiv\ &Server() \mid Client(\texttt{fish}) \mid (\nu ret)(ret[high]!(price\ \texttt{chicken}).\mathbf{0} \mid ret[\lambda x.x]?(y).P) \\ \to\ &Server() \mid Client(\texttt{fish}) \mid (\nu ret)\ P[10/y] \end{aligned}$$

No unrelated client can access a client-server dialog, since private channels are used for communication. Note however, that the second communication action gets highest priority, so that client *Client*(`fish`) can not act before *Client*(`chicken`) obtained the price for the `chicken`.

3.5 Convergence

The next lemma extends on Lemma 1. It states that application of definitions is confluent, so that exhaustive application must lead to a unique outcome (including non-termination).

Lemma 2. *The rewrite relation* $\xrightarrow[nd]{app}$ *is confluent modulo structural congruence. Irreducible processes are congruent to processes of the form* $(\nu\tilde{x}) \prod_{i=1}^{n} P_i$ *such that all* P_i *are sums or match some defined process* $A_i(\tilde{e}_i)$ *with* $\neg\exists\tilde{v}.\tilde{e}_i \Downarrow \tilde{v}$.

Labeled communication steps ($\ell \in \mathbb{N}^4$, $r \in \mathbb{R}_+^\infty$)

$$(\text{COM}_\ell)\quad \frac{\begin{array}{c}\ell = (i_1, j_1, i_2, j_2) \quad i_1 \neq i_2 \\ \pi_{i_1}^{j_1} \Downarrow x[v_1]?\tilde{y} \quad \pi_{i_2}^{j_2} \Downarrow x[v_2]!\tilde{v} \quad v_1 v_2 \Downarrow r \in \mathbb{R}_+^\infty \quad |\tilde{y}| = |\tilde{v}|\end{array}}{(\nu\tilde{x}) \prod_{i=1}^n \sum_{j=1}^{m_i} \pi_i^j.P_i^j \xrightarrow[\ell]{r} (\nu\tilde{x})(\prod_{i=1, i\neq i_1, i_2}^n \sum_{j=1}^{m_i} \pi_i^j.P_i^j \mid P_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid P_{i_2}^{j_2})}$$

Fig. 15. Axioms of stochastic semantics of $\pi(\mathcal{L})$. The rules of the stochastic semantics for defining CTMCs are the same as those for the stochastic π-calculus in Fig. 8.

Proof. A standard analysis of the structural congruence yields the following:

Claim. Let $P = (\nu\tilde{x}) \prod_{i=1}^n P_i$ be a process in prenex normal form in which all bound variables are renamed apart, and such that all P_i are sums or defined processes. In this case, $P \xrightarrow[nd]{app} P'$ if and only if the following rule applies:

$$\frac{\begin{array}{lll} 1 \leq j \leq n & A_j(\tilde{y}_j) \triangleq Q_j & \\ \tilde{e}_j \Downarrow \tilde{v}_j & P_j = A_j(\tilde{e}_j) & P' \equiv (\nu\tilde{x})(\prod_{i=1, i\neq j}^n P_i \mid Q_j[\tilde{v}_j/\tilde{y}_j]) \end{array}}{P \xrightarrow[nd]{app} P'}$$

We consider the rewrite system on congruence classes of processes defined by $[P]_\equiv \xrightarrow[nd]{app} [P']_\equiv$ if $P \xrightarrow[nd]{app} P'$. The above claim shows that this rewrite system terminates on equivalence classes of processes of the form $(\nu\tilde{x}) \prod_{i=1}^n P_i$ where all P_i are either sums or irreducible defined processes $A(\tilde{e})$. We can prove the uniform confluence of this rewrite system by minor adaptation of the proof in Lemma 1. □

The following two properties of $\pi(\mathcal{L})$ are precisely the same that we obtained for the π-calculus with priorities (but now from Lemma 2 instead of Lemma 1).

Proposition 4 (Error-free convergence). *For every P there exists at most one class $[P']_\equiv$ such that $P \Downarrow P'$.*

Proof. This follows immediately from the confluence result in Lemma 2.

Remark 2. If $P \equiv (\nu\tilde{x}) \prod_{i=1}^n M_i$ and $\neg P \xrightarrow[nd]{err} \bot$ then $P \equiv P' \Leftrightarrow P \Downarrow P'$.

3.6 Stochastic Operational Semantics

We present a stochastic semantics for the attributed $\pi(\mathcal{L})$, under the condition that the set of successful values of the attribute language are the stochastic rates $R \subseteq \mathbb{R}_+^\infty$. As in the stochastic π-calculus, we assign highest priority to communication steps with infinite rates, and lowest priority to all others.

The axioms of the stochastic semantics of $\pi(\mathcal{L})$ is given in Fig. 15. The whole presentation of the article is done such that only very few changes are

needed with respect to the stochastic π-calculus. In particular, both calculi have the same closure rules, that were already presented in Fig. 8. This is a major improvement compared to the conference version of the present article [25].

The main difference concerns the new communication rule ($\textsc{com}_\ell$), where we have to evaluate all expressions, in order to compute the stochastic rate. All other difference are hidden in the convergence predicate, as defined in the non-deterministic operational semantics.

The stochastic version remains a proper refinement of the non-deterministic version of the attributed π-calculus with priorities.

Proposition 5. *If the successful values of $\mathcal{L}$ are $\in \mathbb{R}_+^\infty$ with the usual two levels of priority, then for all processes P, P':*

$$P \to P' \text{ iff } (\exists r \in \mathbb{R}_+.\ P \xrightarrow{r} P' \vee \exists n \in \mathbb{N}.\ P \xrightarrow{\infty(n)} P')$$

The proof is mostly the same as for Proposition 2, which relates the two operational semantics of the stochastic π-calculus. The only minor difference is in the treatment of basic interaction steps.

3.7 Type System

Higher-order attribute languages add much expressive power to the π-calculus, but at the price of introducing many new error situations, that systems biology users will be faced with during modeling and simulation. The most frequent errors are type errors. In this section, we present a type system by which to exclude type errors in attributed processes. Note that well-typedness does not exclude all kinds of errors, such as division by 0 for instance.

We present a type system for the $\pi(\mathcal{L})$, which integrates the simple type system for the λ-calculus $\mathcal{L}$ into the type system of the π-calculus from Section 2.6. We will show that the type system of $\pi(\mathcal{L})$ is safe if the type system of $\mathcal{L}$ is. Whether this holds depends on the precise definition of the big-step evaluator of $\mathcal{L}$ that we left open. The attribute languages $\lambda(R)_<$ from Fig. 11 and $\lambda(\mathbb{N}_0, +, =)_{<_1}$ in Fig. 12 are type safe. These two attribute languages are strongly normalizing (i.e., always terminating), as usual for the simply typed lambda calculus. General termination may fail, however, once we add new rules to the big-step evaluators for new constants with functional type, as for instance, in order to define the semantics of fixed-point combinators. In this case, the type safety of the attribute language must be checked again.

We assume a set of type constants such as `Int`, `Bool`, and `String` and define *Types* with these constants by the following grammar:

type constants	ι	$::= \texttt{Int} \mid \texttt{Bool} \mid \ldots$	
types	τ, σ	$::= \iota$	constants
		$\mid\ \tau \to \sigma$	function type
		$\mid\ [\tau] \Rightarrow \tilde{\sigma}$	channel type
		$\mid\ \tau \times \sigma$	pair type

Channel types $[\tau] \Rightarrow \tilde{\sigma}$ now type channel constraints by τ and channel arguments by $\tilde{\sigma}$. More precisely, a channel x of type $[\tau_1 \to \tau_2] \Rightarrow \tilde{\sigma}$ can be used as follows:

- in input prefixes $x[e]?y$, the type of expressions e must be $\tau_1 \to \tau_2$ and the types of $\tilde{y}$ must be $\tilde{\sigma}$.
- in output prefixes $x[e]!\tilde{e}'$, the type of expression e must by τ_1 and the types of $\tilde{e}'$ must be $\tilde{\sigma}$.

Type constants and the types for functions and pairs are standard. As before, we assume that *type environments* Γ and Δ are sets of type assignments for variables $x{:}\tau$ and process names $A{:}\tilde{\tau}$.

In the typed version of the attributed π-calculus, we need to assume type annotation in the syntax. In order to do so, we add types to all occurrences of constants in processes and to all channel creators.

$$\text{typed processes} \qquad P ::= c_\tau \mid (\nu x{:}\tau)P \mid \ldots$$

In examples, we will often ignore type annotations, if they are clear from the context. It is particularly useful to annotate functional types to constants, for instance, in order to type pair selectors $\mathtt{fst}_{\tau\times\sigma\to\tau}$ and $\mathtt{snd}_{\tau\times\sigma\to\sigma}$ or fixed point operators. Note also, that we can use pairs of different types, since we may use different annotations for the same constant.

Example 5. In the client server example, we used a λ-expressions *price* of type $\mathtt{Int} \to \mathtt{Int}$. In a typed version of this example, we have to annotate all constants by their types, i.e., $never =_{df} 0_{\mathtt{Int}}$, $low =_{df} 1_{\mathtt{Int}}$ and $high =_{df} 2_{\mathtt{Int}}$. Furthermore, we need to annotate the new binder in the definition of client by its type:

$$\begin{aligned} \mathit{Server}() &\triangleq \mathit{connect}[\lambda k.\mathtt{if}\ k{=}\mathit{key}_{\mathtt{Int}}\ \mathtt{then}\ \mathit{low}\ \mathtt{else}\ \mathit{never}]?(x, \mathit{ret}).\\ &\qquad\qquad (\mathit{ret}[\mathit{high}]!(\mathit{price}\ x).\mathbf{0}) \mid \mathit{Server}())\\ \mathit{Client}(s) &\triangleq (\nu \mathit{ret}{:}[\mathtt{Int} \to \mathtt{Int}] \Rightarrow (\mathtt{Int}))\mathit{connect}[\mathit{key}_{\mathtt{Int}}]!(s, \mathit{ret}).\mathit{ret}[\lambda z.z]?(y).P \end{aligned}$$

The definitions are well typed if in the following type environment:

$$\begin{aligned} &\mathit{connect} : [\mathtt{Int} \to \mathtt{Int}] \Rightarrow (\mathtt{String}, [\mathtt{Int} \to \mathtt{Int}] \Rightarrow (\mathtt{Int})),\\ &\mathit{Client}{:}(\mathtt{String}), \mathit{Server}{:}() \end{aligned}$$

Rules for typing expressions and processes are given in Fig. 16. Typing rules for expressions are standard from simply typed λ-calculus. Rule (T.AXIOMS) reveals for instance, that we treat equality and addition as curried binary functions. They receive their arguments in two steps, rather than at once. Typing communication prefixes (T.REC) and (T.SEND) derive directly from the above explanations of channel types. Rules for process application (T.APP) and definition (T.DEF) are similar to those for π-calculus with priorities in Fig. 10. Typing rule (T.NEW) now checks explicitly that a new channel name is created and nothing else; previously all values were channel names. Finally, typing rules (T.PAR) and (T.SUM) remain as in Fig. 10 and are not repeated here.

Typing rules for expressions

$$(\text{T.CONST})\ \frac{c \in \mathit{Consts} \quad c{:}\tau}{\Gamma \vdash c_\tau{:}\tau} \qquad (\text{T.AXIOMS})\ \frac{\tau, \sigma \text{ types}}{\begin{array}{ll} \texttt{fst} : \tau \times \sigma \to \tau & \texttt{false}{:}\texttt{Bool} \\ \texttt{snd} : \tau \times \sigma \to \sigma & \texttt{true}{:}\texttt{Bool} \\ = : \tau \to \sigma \to \texttt{Bool} & + : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \end{array}}$$

$$(\text{T.VAR})\ \frac{x{:}\tau \in \Gamma}{\Gamma \vdash x{:}\tau} \qquad (\text{T.PAIR})\ \frac{\Gamma \vdash e{:}\tau \qquad \Gamma \vdash e'{:}\sigma}{\Gamma \vdash \langle e, e' \rangle : \tau \times \sigma}$$

$$(\text{T.COND})\ \frac{\Gamma \vdash e{:}\texttt{Bool} \qquad \Gamma \vdash e_1{:}\tau \qquad \Gamma \vdash e_2{:}\tau}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}$$

$$(\text{T.FUNDEF})\ \frac{\Gamma, x{:}\tau \vdash e{:}\sigma}{\Gamma \vdash \lambda x.e : \tau \to \sigma} \qquad (\text{T.FUNAPP})\ \frac{\Gamma \vdash e : \tau \to \sigma \qquad \Gamma \vdash e'{:}\tau}{\Gamma \vdash e\ e' : \sigma}$$

Typing rules for processes

$$(\text{T.REC})\ \frac{\Gamma \vdash e_1 : [\tau] \Rightarrow \tilde{\sigma} \quad \Gamma \vdash e_2{:}\tau \quad \Gamma, \tilde{x}{:}\tilde{\sigma} \vdash P}{\Gamma \vdash e_1[e_2]?\tilde{x}.P} \qquad (\text{T.NEW})\ \frac{\Gamma, x{:}[\tau] \Rightarrow \tilde{\sigma} \vdash P}{\Gamma \vdash (\nu x{:}[\tau] \Rightarrow \tilde{\sigma})P}$$

$$(\text{T.SEND})\ \frac{\Gamma \vdash e_1 : [\tau_1 \to \tau_2] \Rightarrow \tilde{\sigma} \qquad \Gamma \vdash e_2{:}\tau_1 \qquad \Gamma \vdash \tilde{e}_3{:}\tilde{\sigma} \qquad \Gamma \vdash P}{\Gamma \vdash e_1[e_2]!\tilde{e}_3.P}$$

$$(\text{T.APP})\ \frac{\Gamma \vdash A{:}\tilde{\tau} \qquad \Gamma \vdash \tilde{e}{:}\tilde{\tau}}{\Gamma \vdash A(\tilde{e})} \qquad (\text{T.DEF})\ \frac{\Gamma \vdash A{:}\tilde{\tau} \qquad \Gamma, \tilde{x}{:}\tilde{\tau} \vdash P}{\Gamma \vdash A(\tilde{x}) \triangleq P}$$

Fig. 16. Type system

Proposition 6 (Type safety for expressions). *The attribute language* $\lambda(\mathbb{N}_0, +, =)$ *in Fig. 11 is type safe, i.e., if* $\Gamma \vdash e{:}\tau$ *and* $e \Downarrow v$ *then* $\Gamma \vdash v{:}\tau$.

The proof is standard and proceeds by induction on the proof of $\Gamma \vdash e{:}\tau$ and follows from a substitution lemma stating that: if $\Gamma, x{:}\tau \vdash e{:}\sigma$ and $\Gamma \vdash v{:}\tau$ then $\Gamma \vdash e[v/x] : \sigma$.

Proposition 7 (Normalization). *In the attribute language* $\lambda(\mathbb{N}_0, +, =)$ *every typable expression evaluates to some value, i.e., if* $\Gamma \vdash e{:}\tau$*, then there exists* v *such that* $e \Downarrow v$.

Typings of terms in $\lambda(\mathbb{N}_0, +, =)$ that make use of rule (T.CONST) always do so with a constant type (`Int`) for τ. Therefore, $\lambda(\mathbb{N}_0, +, =)$ is a simply-typed λ-calculus (e.g. fixed point operators are not typable) that is known to have the normalization property. A proof of this result by Tait's methods [39] can be found in many text books (e.g. [40]).

Lemma 3. *The following properties holds for the typing rules of processes:*

1. *(strengthening) if* $\Gamma, x{:}\tau \vdash P$ *and* $x \notin fv(P)$ *then* $\Gamma \vdash P$,
2. *(weakening) if* $\Gamma \vdash P$ *and* $x \notin fv(P)$ *then* $\Gamma, x{:}\tau \vdash P$,
3. *(substitution) if* $\Gamma, x{:}\tau \vdash P$ *and* $\Gamma \vdash v{:}\tau$ *then* $\Gamma \vdash P[v/x]$,
4. *if* $\Gamma \vdash P$ *and* $P \equiv Q$ *then* $\Gamma \vdash Q$.

Strengthening and weakening also hold for the typing of definitions.

Proof. The proofs of the three first properties are straightforward inductions on the derivation of $\Gamma, x{:}\tau \vdash P$ (strengthening and substitution) and of $\Gamma \vdash P$ (weakening). They easily extend to (and depend on) the same properties for expressions. The proof of the last property is by induction of the definition of the structural congruence. The only interesting case is for scope extrusion, that is, assuming $x \notin fv(Q)$, $\Gamma \vdash (\nu x{:}\tau)(P \mid Q) \Leftrightarrow \Gamma \vdash (\nu x{:}\tau)P \mid Q$.
($\Rightarrow$) By rules (T.NEW) and (T.PAR), we have $\Gamma, x{:}\tau \vdash P$ and $\Gamma, x{:}\tau \vdash Q$. Since $x \notin fv(Q)$, by strengthening, $\Gamma \vdash Q$ and, by rule (T.NEW), $\Gamma \vdash (\nu x{:}\tau)P$. Finally, by rule (T.PAR) $\Gamma \vdash (\nu x{:}\tau)P \mid Q$.
($\Leftarrow$) By rules (T.PAR) and then (T.NEW), we have $\Gamma, x{:}\tau \vdash P$ and $\Gamma \vdash Q$. By weakening, we have $\Gamma, x{:}\tau \vdash Q$ and, by (T.PAR) and (T.NEW) we conclude that $\Gamma \vdash (\nu x{:}\tau)(P \mid Q)$. □

Lemma 4. *Let* P *be a process with definitions* $\mathcal{D}$ *in the attributed* π*-calculus with a type safe attribute language, and* Δ *a type environment such that* $\Delta \vdash D$ *for all* $D \in \mathcal{D}$. *If* $\Gamma \vdash P$ *with* $\Delta \subseteq \Gamma$ *and* $P \Downarrow Q$ *then* $\Gamma \vdash Q$.

Proof. By reduction rule (CONV), there exists $n \geq 0$ such that $P(\xrightarrow[nd]{app})^n Q$. Thus, we reduce to the proof, by induction on n, that if $\Gamma \vdash P$ and $P(\xrightarrow[nd]{app})^n Q$ then $\Gamma \vdash Q$. The case $n = 0$ is straightforward, so we only need to prove the case $n = 1$ by induction on the derivation of $P \xrightarrow[nd]{app} Q$. The induction cases (PAR) and (NEW) are straightforward and (STRUCT) follows from Lemma 3(4). In the (APP) case, we have $A(\tilde{e}) \xrightarrow[nd]{app} P[\tilde{v}/\tilde{e}]$ with $\tilde{e} \Downarrow \tilde{v}$ and $A(\tilde{x}) \triangleq P$. Since $\Delta \vdash A(\tilde{x}) \triangleq P$, by weakening Lemma 3(2), $\Gamma \vdash A(\tilde{x}) \triangleq P$ and, by rule (T.DEF), $\Gamma, \tilde{x}{:}\tilde{\sigma} \vdash P$ (†) and $\Gamma \vdash A{:}\tilde{\sigma}$. Moreover, by hypothesis, $\Gamma \vdash A(\tilde{e})$, thus $\Gamma \vdash \tilde{e}{:}\tilde{\sigma}$. Since $\tilde{e} \Downarrow \tilde{v}$, the type safety of attribute language yields $\Gamma \vdash \tilde{v}{:}\tilde{\sigma}$. Property (†) and substitution Lemma 3(3) yield $\Gamma \vdash P[\tilde{v}/\tilde{x}]$. □

Theorem 1 (Type safety for processes). *If* $\mathcal{L}$ *is a type safe attribute language then* $\pi(\mathcal{L})$ *is type safe in that if* $\Gamma \vdash P$ *and* $P \to Q$ *then* $\Gamma \vdash Q$.

Proof. More precisely, we assume that P is a process with definitions in $\mathcal{D}$ and that Γ is a type environment with $\Gamma \vdash P$ and $\Gamma \vdash D$ for any $D \in \mathcal{D}$. By reduction rule (PRIOR), there exists P' such that $P \Downarrow P'$ and $P' \xrightarrow[nd]{r} Q$ where $r \in R$. By Lemma 4, it is thus sufficient to prove the theorem for reduction $\xrightarrow[nd]{r}$ by induction on the derivation. The inductive cases (PAR), (STRUCT) and (NEW) are straightforward. In the (COM) case, we have $P = e_1[e_2]?\tilde{y}.P_1 + M_1 \mid e_1'[e_2']?\tilde{e}.P_2 + M_2$, $Q = P_1[\tilde{v}/\tilde{y}] \mid P_2$, such that $e_1 \Downarrow x$, $e_1' \Downarrow x$, $e_2 \Downarrow v_2$, $e_2' \Downarrow v_2'$,

$v_2 v_2' \Downarrow r$ and $\tilde{e} \Downarrow \tilde{v}$. By rules (T.PAR), (T.SUM), (T.REC) and (T.SEND), we have $\Gamma \vdash e_1 : [\tau] \Rightarrow \tilde{\sigma}$ and $\Gamma \vdash e_1 : [\tau_1 \to \tau_2] \Rightarrow \tilde{\sigma'}$. Since $e_1 \Downarrow x$ and $e_1' \Downarrow x$, type safety ensures that x has the same type as e_1 and e_1', thus $\tau = \tau_1 \to \tau_2$, $\sigma' = \tau$ and $\Gamma \vdash x : [\tau_1 \to \tau_2] \Rightarrow \tilde{\sigma}$. Moreover, since $\Gamma \vdash \tilde{e}{:}\tilde{\sigma}$, type safety yields $\Gamma \vdash \tilde{v}{:}\tilde{\sigma}$. In addition, we have $\Gamma, \tilde{y}{:}\tilde{\sigma} \vdash P_1$ and, by the substitution Lemma 3(3), $\Gamma \vdash P_1[\tilde{v}/\tilde{y}]$. Finally, from $\Gamma \vdash P_2$ and rule (T.PAR), $\Gamma \vdash P_1[\tilde{v}/\tilde{y}] \mid P_2$. □

Corollary 2 (Error freeness). *If $\mathcal{L}$ is an attribute language that is both type safe and normalizing (see Propositions 6 and 7) then $\pi(\mathcal{L})$ is error free in that if $\Gamma \vdash P$ and $P \to^* Q$ then $\neg Q \xrightarrow[nd]{err} \bot$.*

The proof is elaborated in Appendix B.

4 Modeling Techniques and Biological Examples

We illustrate the usefulness of the attributed π-calculus for modeling biological systems, and extract useful modeling techniques. As examples we consider spatial aspect of Euglena's phototaxis and cooperative enhancement for gene regulation at the lambda switch. Furthermore, we discuss population-based modeling in the attributed π-calculus, in contrast to modeling population aspects in individual-based modeling.

4.1 Spatial Aspects: Euglena's Phototaxis

We start with simple spatial aspects by considering location dependent motion at the example of Euglena's phototaxis [41]. The general relevance of spatial aspects in molecular biology is discussed e.g. in [42]. For the modeling of systems with dynamic compartment structures, we refer to Section 5.2.

Euglena is a single cell organism that lives in inland water and performs photosynthesis. Depending on the brightness, it swims up and down in order to reach a zone with just the right amount of light, [43]. In our model, the probability that an Euglena moves upwards is constant, since it always tries to reach regions with more light. However, in order to avoid too intense light, Euglena moves downwards whenever it receives light for photosynthesis, i.e. with a probability proportional to the light intensity of its current position. We assume that light photons travel top-down and that light intensity degrades exponentially with respect to the depth (repeated filtering).

Given a light source with initial intensity $I \in \mathbb{R}^+$ at depth 0, and a transparency factor for filtering $\sigma \in]0, 1]$, this means that the light intensity at depth $d \in \mathbb{R}^+$ equals $\sigma^d * I$. Our model comprises two light sources with initial intensities $\mathtt{I}_1$ and $\mathtt{I}_2$, such that the overall amount of light yields $\mathtt{I} = \mathtt{I}_1 + \mathtt{I}_2$. Furthermore, we assume a constant rate $\mathtt{u} \in \mathbb{R}$ for upward motion.

We consider discrete depth levels $\{0, \ldots, \mathtt{m}\}$ where level 0 denotes the surface and level $\mathtt{m} \in \mathbb{N}_0$ the ground. Euglenas may move up and down in steps of exactly 1 level. Continuous depth levels and movement steps could be modeled similarly,

Parameters

```
n ∈ ℕ0      // initial number of Euglenas per water level
m ∈ ℕ0      // deepest water level
I1, I2 ∈ ℝ+ // intensity rates of light sources 1 and 2
σ ∈ [0,1]   // transparency of water
u ∈ ℝ+      // Euglena's upwards speed
```

Process definitions

```
Euglena(d) ≜ up[λ_. if d ≥ 1 then u]?().Euglena(d − 1)
           + down[λi. if d ≤ m − 1 then σ^d ∗ i]?().Euglena(d + 1)
Light(i) ≜ down[i]!().Light(i)
Dummy() ≜ up[_]!().Dummy()
```

Example solution

$\prod_{d=0}^{\mathtt{m}} \prod_{i=1}^{\mathtt{n}} \mathsf{Euglena}(d) \mid \mathsf{Light}(\mathtt{I}_1) \mid \mathsf{Light}(\mathtt{I}_2) \mid \mathsf{Dummy}()$

Fig. 17. A discrete model of Euglena's light-dependent motion with two light sources.

but would increase simulation costs. For the initial system we assume, that $\mathtt{n} \in \mathbb{N}_0$ Euglenas are on every level, summing up to totally $\mathtt{n} * (\mathtt{m}+1)$ Euglenas in the water. Our model comprises two light sources, which are located at the water surface. Their respective intensities are given by real numbers $\mathtt{I}_1, \mathtt{I}_2 \in \mathbb{R}^+$. The probability of an interaction with a light source is proportional to its intensity. The values $\mathtt{u}, \sigma, \mathtt{m}, \mathtt{n}, \mathtt{I}_1, \mathtt{I}_2$ are model parameters.

We define our model in $\pi(\lambda(\mathbb{R}, +, -, *, /, pow, \leq)_{<_1})$, the attributed π-calculus with constants for real numbers and the usual arithmetic operations. For convenience, we write x^y instead of $(pow\ x)\ y$. The successful values are the positive real numbers, that all have the same level of priority. The big-step evaluator for these operators can be defined as usual (in analogy to natural numbers, see Section 3.2). We consider, non-zero positive real numbers to be the only successful values.

Our model is given in Fig. 17. An Euglena at depth level d may interact with a light source of intensity i and go down by one level:

$$down : \mathsf{Euglena}(d), \mathsf{Light}(i) \xrightarrow{\sigma^d * i} \mathsf{Euglena}(d+1), \mathsf{Light}(i)$$

Such interactions happen on channel *down* with rate $\sigma^d * i$ under the condition that $d \leq m-1$. An Euglena can also move up with constant rate $\mathtt{u}$ by interacting with a dummy interaction partner on channel *up*:

$$up : \mathsf{Euglena}(d) \xrightarrow{u} \mathsf{Euglena}(d-1)$$

If Euglena is at the surface, i.e. the constraint $d \geq 1$ is not satisfied, it cannot move upwards any further.

Based on the master equation, the amounts of Euglenas on each depth level in equilibrium can be computed. For illustration, we fix $m = 4$. The equilibrium

happens if the system is free of change, such that its derivation is 0. Let $l_0, \ldots, l_4$ be the amounts of Euglena per level. The master equation yields:

$$\begin{pmatrix} -\sigma^0 * \mathtt{I} & u & 0 & 0 & 0 \\ \sigma^0 * \mathtt{I} & -(\sigma^1 * \mathtt{I} + u) & u & 0 & 0 \\ 0 & \sigma^1 * \mathtt{I} & -(\sigma^2 * \mathtt{I} + u) & u & 0 \\ 0 & 0 & \sigma^2 * \mathtt{I} & -(\sigma^3 * \mathtt{I} + u) & u \\ 0 & 0 & 0 & \sigma^3 * \mathtt{I} & -u \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} l_0 \\ l_1 \\ l_2 \\ l_3 \\ l_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5n \end{pmatrix}$$

The first equation $-\sigma^0 * \mathtt{I} * l_0 + u * l_1 = 0 = l_0'$ states that the change in the amount of Euglena at level 0 is obtained by summing up a loss of $\sigma^0 * \mathtt{I} * l_0$ due to Euglena's downward motion to level 1, and a gain of $u * l_1$ due to Euglenas upwards motion from level 1. The last equation $\sum_{i=0}^{4} l_i = 5n$ denotes that the overall amount of Euglena is constant and equals the initial amount.

In order to verify the behavior of our model with respect to predictions obtained from the Master Equation, we performed two simulation experiments, named A and B. There are constantly five depth levels ($\mathtt{m} = 4$), 100 Euglenas on each depth level ($\mathtt{n} = 100$), a rate of upward motion $\mathtt{u} = 0.4$, intensity rates $\mathtt{I}_1 = 5.0, \mathtt{I}_2 = 15.0$ ($\mathtt{I} = 20.0$), and transparency factors $\sigma = 0.1$ in experiment A and $\sigma = 0.2$ in experiment B. Each experiment consists in a single simulation run, all of them performed until simulation time $t = 10.0$.

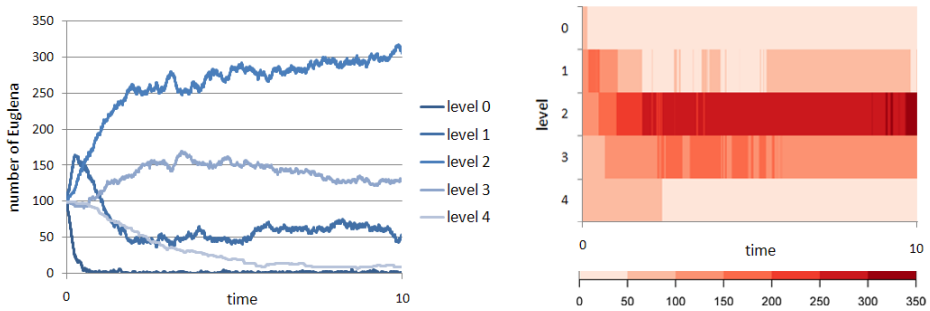
The simulation results are presented in Fig. 18. Heat maps and line charts show the amounts of Euglena on each depth level over time. Below them, the solutions of the Master Equation with the respective model parameters are given. The simulation results confirm the predictions, with slight derivations due to stochasticity. The comparison of both experiments shows, that with a higher transparency Euglena accumulates on a deeper level, as a consequence of more light being available.

We can translate our Euglena model with attributed processes into the stochastic π-calculus without attributes, since all parameters are finitely valued. The idea is to duplicate the *down*-channels for all depth levels, so that its rates can be made dependent for the depth. This leads to processes $\mathsf{Euglena}_d()$, $\mathsf{Light}_{1,d}()$, and $\mathsf{Light}_{2,d}()$ for all possible depth levels, see Fig. 19.

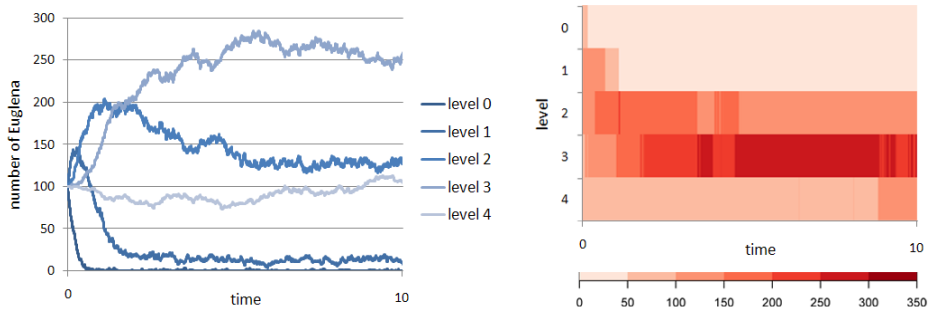
4.2 Cooperative Enhancement: Gene Regulation at Lambda Switch

Cooperative binding is a frequent and often decisive aspect in gene regulatory networks, where proteins stabilize each other's binding to neighboring DNA sites by adhesive contacts. In quantitative terms, the unbinding rate of one DNA-protein complex decreases by the existence of another. This is an instance of cooperative enhancement of reaction rates by third partners. As shown in [27, 26], cooperative enhancement can be modeled in the stochastic π-calculus. It however requires nontrivial encodings, that can be alleviated within the attributed π-calculus.

A well understood instance of cooperative binding occurs during transcription initiation control at the lambda switch. The lambda switch is a segment of the



Experiment A, Predictions: $l_0 = 1.16, l_1 = 57.84, l_2 = 289.20, l_3 = 144.65, l_4 = 7.15$



Experiment B, Predictions: $l_0 = 0.26, l_1 = 12.81, l_2 = 128.14, l_3 = 256.28, l_4 = 102.51$

Fig. 18. Experiments with Euglena: two experiments (A,B), with $\mathtt{m} = 4$, $\mathtt{n} = 100$, $\mathtt{u} = 0.4$, $\mathtt{I}_1 = 5.0, \mathtt{I}_2 = 15.0$ ($\mathtt{I} = 20.0$), and $\sigma = 0.1$ in experiment A and $\sigma = 0.2$ in experiment B. For each experiment a single simulation run until simulation time $t = 10.0$. The line charts and heat maps show the development of the number of Euglenas on each depth level over time. Predictions for the amounts on the different depth levels in equilibrium as obtained by solving the Master Equation are shown below each experiment.

```
// Euglenas on different depth levels
Euglena_0() ≜ down_0?().Euglena_1()
Euglena_1() ≜ up?().Euglena_0() + down_1?().Euglena_2()
...
Euglena_m() ≜ up?().Euglena_{m-1}()
// light from first source on different levels
Light_{1,0}() ≜ down_0:σ^0*I_1!().Light_{1,0}()
...
Light_{1,m}() ≜ down_m :σ^m*I_1!().Light_{1,m}()
// light from second source on different levels
Light_{2,0}() ≜ down_0:σ^0*I_2!().Light_{2,0}()
...
Light_{2,m}() ≜ down_m :σ^m*I_2!().Light_{2,m}()
Dummy() ≜ up:u!().Dummy()
```

Example solution

$\prod_{d=0}^{m}$ ($\prod_{i=0}^{n}$ $\mathtt{Euglena}_d()$ | $\mathtt{Light}_{1,d}()$ | $\mathtt{Light}_{2,d}()$)

Fig. 19. An equivalent model of Euglena in the stochastic π-calculus.

DNA of bacteriophage lambda. It contains two binding sites Or_1 and Or_2, where rep and cro proteins can bind. An unstable binding of a rep molecule to Or_2 is stabilized by the simultaneous presence of another rep at the neighboring site Or_1. As illustrated in Fig. 20, the two proteins actually touch each other.

A model of cooperative binding at Or_2 in $\pi(\lambda(\mathbb{R}_+^\infty, =)_{<_2})$ is presented in Fig. 21. It contains the parametric definition Prot(*type*), which emulates the behavior of the proteins. The parameter *type* can be instantiated by *rep* or *cro*, modeling either protein sort. Proteins can bind to both sites Or_1 and Or_2. Free sites are defined by processes $\text{Or}_1()$ and $\text{Or}_2()$, where proteins can attach via channel *bind*. As this occurs the channel *release* is created, and henceforth connects the protein to the site (complexation). Later communication on *release* breaks the complex. The reaction rate of complexation is fixed to `0.098`. For decomplexation the rate is determined by the sender, i.e. the binding site, the receiving protein accepts it by applying the identity function.

Now consider the models for the protein bound DNA sites. $\text{Or}_\text{i}\text{B}$(*type*,*release*) describes the unbinding from the occupied site Or_i, where *type* indicates the type of the bound protein. For $i = 1$ the rate of the unbinding reaction merely depends on the protein type.

For the second site ($i = 2$) decomplexation is influenced by cooperative binding. To model this, Or_1 and Or_2 are linked via the channel *or2Delay*, illustrated in Fig. 20. Additionally, the release operation is decomposed into an interaction on channel *or2Delay*, with a reaction rate defining the actual unbinding delay, and an immediate communication on *release*. As stated in the definition of the global channel *or2Delay* the unbinding delay depends not only on the type of the

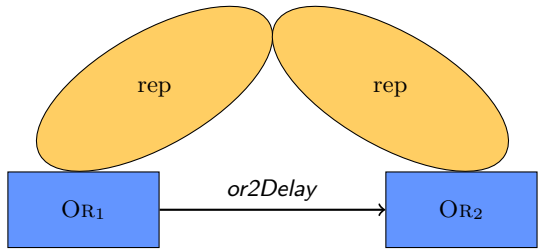


Fig. 20. The decay of the rep-Or_2 complex: in order to make the decay rate of the rep-Or_2 complex dependent on Or_1's state, the two sites communicate on *or2Delay* before Or_2 unbinds.

bound protein, but also on the state of Or_1, which can be either *free*, bound to *rep* or bound to *cro*.

A previous model [26] in the stochastic π-calculus requires to keep Or_2 constantly informed about state changes of Or_1, which is implemented by immediate communication steps. Keeping state information consistent in this manner is error-prone, it may easily lead to deadlocks. A subsequent model [27] in SPiCO requires significantly fewer updates. In $\pi(\lambda(\mathbb{R}_+^\infty, =)_{<_2})$, reaction rates directly depend on the attribute values of the interaction partners. State changes are propagated without additional communication steps.

4.3 Population-based Modeling

The stochastic π-calculus supports individual-based modeling where molecules are mapped to objects. The attributed π-calculus enables in addition a population-based modeling style, where reactions are mapped to objects.

For illustration, we consider a chemical system with three species A, B, and C and the following two reactions with rates k_1 and k_2:

$$r_1 : A + B \xrightarrow{k_1} C \qquad r_2 : B + C \xrightarrow{k_2} A$$

Figure 22 shows a population-based description of this system in $\pi(\lambda(\mathbb{R}, +, -, *, /, pow, \leq)_{<_1})$. The model parameters $\mathsf{a}_0, \mathsf{b}_0, \mathsf{c}_0 \in \mathbb{N}_0$ represent the initial amounts of the three species. A process $\mathsf{Reac}(f, d_a, d_b, d_c)$ defines a reaction with kinetic function f, while the other parameters d_a,d_b, and d_c reflect how the reaction affects the population, i.e. the differences in the amounts of the species. The latter parameters could also be regarded as stoichiometric factors, except that reactants are always associated with negative numbers. The kinetics of all reactions in this example follow the law of Mass action. For instance, the kinetics of r_1 yields the product of the amounts of species A and B and rate k_1. It changes the population by decreasing the amount of A and B by one each and increasing the amount of C by one. Thus, to represent reaction r_1, our initial solution comprises a process $\mathsf{Reac}(\lambda a.\lambda b.\lambda c.a * b * \mathtt{k}_1, \text{-1}, \text{-1}, 1)$, where the function

Process definitions

```
Prot(type) ≜ (νrelease) bind[_]!(type, release).release[λr.r]?().Prot(type)
Or1() ≜ bind[λ_. 0.098]?(type, release).Or1B(type,release)
      + or2Delay[free]!.Or1()
Or1B(type,release) ≜ release[
                        if type = rep then 0.155 else
                        if type = cro then 2.45
                        ]!().Or1()
                    + or2Delay[type]!().Or1B(type,release)
Or2() ≜ bind[λ_. 0.098]?(type, release).Or2B(type,release)
Or2B(type,release) ≜
  or2delay[λt.
    if t = rep then
     if type = rep then 0.155 else // big delay (cooperative)
     if type = cro then 3.99 // small delay
    else 2.45 // cro or free
    ]?().release[∞]!().Or2()
```

Example process

$\mathrm{Or}_1() \mid \mathrm{Or}_2() \mid \prod_{i=1}^{28} \mathsf{Prot}(rep) \mid \prod_{i=1}^{67} \mathsf{Prot}(cro)$

Fig. 21. A model of cooperative binding between Or_1 and Or_2 at the λ switch.

parameters a, b, c represent the amounts of A, B, C, respectively. Consequently, we also start with one process $\mathsf{Reac}(\lambda a.\lambda b.\lambda c.b * c * \mathtt{k}_2,1,-1,-1)$ to model reaction r_2. Notice, that it is also possible to account for different kinetic laws and different stoichiometric factors.

In order to represent populations, we define processes $\mathsf{Pop}(a,b,c)$, whose parameters stand for the amounts of the three species. In addition to the two reactions, the initial process thus also comprises $\mathsf{Pop}(\mathtt{a}_0,\mathtt{b}_0,\mathtt{c}_0)$. Interactions on channel r indicate the occurrence of a reaction. For the computation of reaction kinetics $\mathsf{Reac}(f,d_a,d_b,d_c)$ provides its kinetics function f as the constraint argument. $\mathsf{Pop}(a,b,c)$ defines a constraint function, which applies f to the current amounts of the species. When an interaction occurs, $\mathsf{Reac}(f,d_a,d_b,d_c)$ sends its population changes and $\mathsf{Pop}(a,b,c)$ applies them to the current amounts, which is implemented by recursively calling $\mathsf{Pop}(a+d_a,b+d_b,c+d_c)$. Afterward, the next reaction can happen. Function f will evaluate to 0 whenever one of the populations becomes 0.

The model can be generalized to systems in which new species can be created dynamically, by using property lists as parameters, where each element contains a pair of a species and its amount. Even new reactions could be dynamically introduced. Such a model is close to the way biochemistry is expressed in sCCP, pointing to the possibility of generally encoding sCCP in attributed processes, which is, however, still subject to future work.

Parameters

$\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0 \in \mathbb{N}_0$ // initial amounts of A, B, and C
$\mathtt{k}_1, \mathtt{k}_2 \in \mathbb{R}$ // reaction rates

Process definitions

$\mathsf{Reac}(f, d_a, d_b, d_c) \triangleq r[f]!(d_a, d_b, d_c).\mathsf{Reac}(f, d_a, d_b, d_c)$
$\mathsf{Pop}(a, b, c) \triangleq r[\lambda f.(f\ a\ b\ c)]?(d_a, d_b, d_c).\mathsf{Pop}(a + d_a, b + d_b, c + d_c)$

Example solution

$\mathsf{Reac}(\lambda a.\lambda b.\lambda c.\, a * b * \mathtt{k}_1, -1, -1, 1) \mid \mathsf{Reac}(\lambda a.\lambda b.\lambda c.\, b * c * \mathtt{k}_2, 1, -1, -1) \mid \mathsf{Pop}(\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0)$

Fig. 22. A population-based model of three species and two reactions in $\pi(\lambda(\mathbb{R}, +, -, *, /, pow, \leq)_{<_1})$. Process $\mathsf{Reac}(f, d_a, d_b, d_c)$ defines reactions, where parameter f is a function reflecting the reaction kinetics and parameters d_a,d_b,d_c account for the way species amounts are changed when the reaction occurs. Process $\mathsf{Pop}(a, b, c)$ reflects species amounts and interacts with process $\mathsf{Reac}(f, d_a, d_b, d_c)$ for reaction execution.

4.4 Global Information in Individual-Based Modeling

Priorities are necessary in order to track global information in individual-based models in a consistent manner, for instance in order to model changes in compartment structures [18]. The general idea is to propagate changes globally by a sequence of prioritized local interactions, before enabling the next possible reactions, since these are given lower priority.

In the example Fig. 23, we trace global information on population sizes, i.e. numbers of individuals. It rephrases the population-based model in Section 4.3 in an individual-based way. We model molecules of the three species by processes $\mathsf{A}()$,$\mathsf{B}()$, and $\mathsf{C}()$ and define a process $\mathsf{Pop}(a, b, c)$ such that it tracks molecule numbers as obtained by interactions between molecules. Chemical reactions are mimicked by interactions on channel r, where $\mathsf{A}()$ or $\mathsf{C}()$ send to $\mathsf{B}()$ with rates $\mathtt{k}_1$ and $\mathtt{k}_2$, respectively. The changes in the population are updated by prioritized interaction over channel u once a reaction occurred. Process $\mathsf{Pop}(a, b, c)$ receives such changes with infinite rate, i.e. with priority, such that no reaction can occur before the population information is updated. By this, it is ensured, that the effect of each reaction is correctly reflected in the species amounts, i.e. that the population information is consistent.

This example indicates, that priorities might permit to implement individual-based models with global stores, as proposed in [44].

4.5 Species-Based Modeling

As a last example, we show how one can rephrase the model given in Section 4.3 in a *species-based* style, see Figure 24. We make use of priorities (i.e. immediate

Parameters

$\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0 \in \mathbb{N}_0$ // initial amounts of A, B, and C
$\mathtt{k}_1, \mathtt{k}_2 \in \mathbb{R}$ // reaction rates

Process definitions

$\mathsf{A}() \triangleq r[\mathtt{k}_1]!().u[_]!(-1,-1,1).\mathsf{C}()$
$\mathsf{B}() \triangleq r[\lambda k.k]?().0$
$\mathsf{C}() \triangleq r[\mathtt{k}_2]!().u[_]!(1,-1,-1).\mathsf{A}()$
$\mathsf{Pop}(a,b,c) \triangleq u[\lambda_.\infty]?(d_a,d_b,d_c).\mathsf{Pop}(a+d_a,b+d_b,c+d_c)$

Example solution

$\prod_{i=1}^{\mathtt{a}_0} \mathsf{A}() \mid \prod_{i=1}^{\mathtt{b}_0} \mathsf{B}() \mid \prod_{i=1}^{\mathtt{c}_0} \mathsf{C}() \mid \mathsf{Pop}(\mathtt{a}_0,\mathtt{b}_0,\mathtt{c}_0)$

Fig. 23. An individual-based variant of the population-based model in Figure 22. Processes A(),B(), and C() represent species, process Pop(a,b,c) accounts for species amounts, which are updated by prioritized interactions on channel u.

communications) and the fact that reactions have at most two reactants. A process $\mathsf{A}(a)$ represents the species A which is attributed by the number a of molecules of A. Species B and C are implemented analogously. In contrast to the individual-based model, the solution always contains a single process for each species.

Reaction r_1 is modeled as an interaction between processes $\mathsf{A}(a)$ and $\mathsf{B}(b)$ on channel r_1. The corresponding communication constraint yields a rate, which follows Mass action kinetics. After an interaction, $\mathsf{A}(a\text{-}1)$ and $\mathsf{B}(b\text{-}1)$ are called recursively, thus decreasing the number of molecules of species A and B. In parallel, a request is sent with priority on channel u_c in order to increase the number of molecules of species C. Reaction r_2 is implemented analogously.

5 Expressiveness of the Attributed Pi-Calculus

We show that the attributed π-calculus provides a unifying framework that generalizes on various dialects of the π-calculus in the literature.

5.1 Encoding of the π-Calculus with Priorities

We start with an encoding of the π-calculus with priorities, and prove its correctness with respect to both semantics, non-deterministic and stochastic. The encoding can be refined such that it preserves well-typing.

The translation of the π-calculus with priorities in $(R, <)$ into the attributed π-calculus $\pi(\lambda(R)_<)$ is given in Fig. 25. Senders $x{:}r!\tilde{y}.P$ are mapped to $x[r]!\tilde{y}.P$ and receivers $x?\tilde{y}.P$ to $x[\lambda z.z]?\tilde{y}.P$.

Parameters

$\mathtt{a}_0, \mathtt{b}_0, \mathtt{c}_0 \in \mathbb{N}_0$ // initial amounts of A, B, and C
$\mathtt{k}_1, \mathtt{k}_2 \in \mathbb{R}$ // reaction rates

Process definitions

$$\mathsf{A}(a) \triangleq r_1[a]!()\,.(\mathsf{A}(a-1) \mid u_c[_]!(1)) + u_a[\lambda_.\infty]?(d)\,.\mathsf{A}(a+d)$$
$$\mathsf{B}(b) \triangleq r_1[\lambda a.\mathtt{k}_1 * a * b]?()\,.\mathsf{B}(b-1) + r_2[\lambda c.\mathtt{k}_2 * c * b]?()\,.\mathsf{B}(b-1)$$
$$\mathsf{C}(c) \triangleq r_2[c]!()\,.(\mathsf{C}(c-1) \mid u_a[_]!(1)) + u_c[\lambda_.\infty]?(d)\,.\mathsf{C}(c+d)$$

Example solution

$$\mathsf{A}(\mathtt{a}_0) \mid \mathsf{B}(\mathtt{b}_0) \mid \mathsf{C}(\mathtt{c}_0)$$

Fig. 24. An species-based variant of the population-based model in Figure 22, also in $\pi(\lambda(\mathbb{R}, +, -, *, /, pow, \leq)_{<_1})$. Processes $\mathsf{A}(a)$, $\mathsf{B}(b)$, and $\mathsf{C}(c)$ represent species parametrized by their multiplicities possibly updated through channel u_a and u_c.

$$[\![x?\tilde{y}.P]\!] = x[\lambda z.z]?\tilde{y}.[\![P]\!] \qquad [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$$
$$[\![x{:}r!\tilde{y}.P]\!] = x[r]!\tilde{y}.[\![P]\!] \qquad [\![M_1 + M_2]\!] = [\![M_1]\!] + [\![M_2]\!] \qquad [\![\mathbf{0}]\!] = \mathbf{0}$$
$$[\![(\nu x)P]\!] = (\nu x)[\![P]\!] \qquad [\![A(\tilde{x}) \triangleq P]\!] = A(\tilde{x}) \triangleq [\![P]\!]$$

Fig. 25. Encoding the π-calculus with priorities $(R, <)$ into $\pi(\lambda(R)_<)$, and the stochastic π-calculus into the attributed π-calculus with stochastic semantics.

Theorem 2. *The encoding of the π-calculus with priorities $(R, <)$ into the attributed π-calculus with priorities $\pi(\lambda(R)_<)$ is correct in that for all processes P, P' and attributed processes Q:*

1. *if $P \to P'$ then $[\![P]\!] \to [\![P']\!]$.*
2. *if $[\![P]\!] \to Q$ then there exists a process $\hat{Q}$ of the π-calculus with priorities such that $[\![\hat{Q}]\!] \equiv Q$ and $P \to \hat{Q}$.*

The proof is elaborated in Appendix B. It is mostly straightforward, but covers several pages since all rules of both calculi must be inspected in detail.

The same encoding is also correct with respect to the stochastic operational semantics, under the assumption that we choose the set of stochastic rates $(\mathbb{R}_+^\infty, <_2)$ as priorities.

Theorem 3. *The encoding of the π-calculus with priorities in $(\mathbb{R}_+^\infty, <_2)$ into the attributed π-calculus $\pi(\lambda(\mathbb{R}_+^\infty)_{<_2})$ is correct with respect to the stochastic operational semantics. For all processes $P, P', \hat{P}$, attributed processes Q and labels $\alpha \in \{r, \infty(n) \mid r \in \mathbb{R}_+, n \in \mathbb{N}\}$:*

1. *if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$*

2. if $[\![\hat{P}]\!] \xrightarrow{\alpha} Q$ then there exists a process $\hat{Q}$ such that $\hat{P} \xrightarrow{\alpha} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$.

Proof. The stochastic semantics of both calculi are build on top of their on their non-deterministic semantics. We prove in the appendix (see the proof of Theorem 2) that the translation is invariant under substitution and that it reflects and preserves the structural congruence and errors. Furthermore, we proved there that if $P \; \rho \; Q$ then $[\![P]\!] \; \rho \; [\![Q]\!]$, for $\rho \in \{\Downarrow, \rightarrow\} \cup \{\xrightarrow[nd]{\alpha} \mid \alpha \in \{app\} \cup R\}$.

Claim. Relation $\xrightarrow[\ell]{r}$ is preserved and reflected by translation (i.e., positions ℓ of redexes remain unchanged):

1. if $P \xrightarrow[\ell]{r} Q$ then $[\![P]\!] \xrightarrow[\ell]{r} [\![Q]\!]$,
2. if $[\![\hat{P}]\!] \xrightarrow[\ell]{r} Q$ then exists $\hat{Q}$ such that $\hat{P} \xrightarrow[\ell]{r} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$.

Proof. 1. If $P \xrightarrow[\ell]{r} Q$ then rule (COM_ℓ) can be applied as follows:

$$\frac{\ell = (i_1, j_1, i_2, j_2) \quad \pi_{i_1}^{j_1} = x?\tilde{y} \quad \pi_{i_2}^{j_2} = x{:}r!\tilde{z} \quad |\tilde{y}| = |\tilde{v}|}{P \xrightarrow[\ell]{r} Q}$$

where

$$P = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \text{ and } Q = (\nu\tilde{x})(\prod_{i=1, i\neq i_1, i_2}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \mid P_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid P_{i_2}^{j_2})$$

Thus, $[\![P]\!] = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} [\![\pi_i^j]\!].[\![P_i^j]\!]$, with $[\![\pi_{i_1}^{j_1}]\!] = x[\lambda y.y]?\tilde{y}$ and $[\![\pi_{i_2}^{j_2}]\!] = x[r]!\tilde{z}$. Now, rule ($\text{COM}_\ell$) of $\pi(\mathcal{L})$ applies to the translations, while using (VAL) and (FUN):

$$\frac{\ell = (i_1, j_1, i_2, j_2) \quad \begin{array}{c} [\![\pi_{i_1}^{j_1}]\!] \Downarrow x[\lambda y.y]?\tilde{y} \\ [\![\pi_{i_2}^{j_2}]\!] \Downarrow x[r]!\tilde{z} \end{array} \quad (\lambda y.y)r \Downarrow r \in \mathbb{R}_+^\infty \quad |\tilde{y}| = |\tilde{z}|}{[\![P]\!] \xrightarrow[\ell]{r} [\![Q]\!]}$$

where

$$[\![P]\!] = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} [\![\pi_i^j]\!].[\![P_i^j]\!] \text{ and}$$

$$[\![Q]\!] = (\nu\tilde{x}) \Big(\prod_{i=1, i\neq i_1, i_2}^{n} \sum_{j=1}^{m_i} [\![\pi_i^j]\!].[\![P_i^j]\!] \mid [\![P_{i_1}^{j_1}]\!][\tilde{v}/\tilde{y}] \mid [\![P_{i_2}^{j_2}]\!] \Big)$$

The last equality follows from the substitution claim $[\![P_{i_1}^{j_1}[\tilde{v}/\tilde{y}]]\!] = [\![P_{i_1}^{j_1}]\!][\tilde{v}/\tilde{y}]$ and the compositionality of the translation.

2. If $[\![\hat{P}]\!] \xrightarrow[\ell]{r} Q$ then rule (COM$_\ell$) must be applicable as follows:

$$\frac{\ell = (i_1, j_1, i_2, j_2) \quad \pi_{i_1}^{j_1} \Downarrow x[v_1]?\tilde{y} \quad \pi_{i_2}^{j_2} \Downarrow x[v_2]!\tilde{v} \quad v_1 v_2 \Downarrow r \in \mathbb{R}_+^\infty \quad |\tilde{y}| = |\tilde{v}|}{[\![\hat{P}]\!] \xrightarrow[\ell]{r} Q}$$

where

$$[\![\hat{P}]\!] = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \text{ and}$$

$$Q = (\nu\tilde{x})\Big(\prod_{i=1, i\neq i_1, i_2}^{n} \sum_{j=1}^{m_i} \pi_i^j.P_i^j \mid P_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid P_{i_2}^{j_2} \Big)$$

Since the translation is compositional, process $\hat{P}$ must have the form $\hat{P} = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j$, with $[\![\hat{\pi}_i^j]\!] = \pi_i^j$ and $[\![\hat{P}_i^j]\!] = P_i^j$. Furthermore, we have that $v_1 = \lambda y.y$, $v_2 = r$, such that $\hat{\pi}_{i_1}^{j_1} = x?\tilde{y}$ and $\hat{\pi}_{i_2}^{j_2} = x{:}r!\tilde{z}$, with $\tilde{v} = \tilde{z}$. We define $\hat{Q} = (\nu\tilde{x})(\prod_{\substack{i=1 \\ i\neq i_1, i_2}}^{n} \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j \mid \hat{P}_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid \hat{P}_{i_2}^{j_2})$. Since the translation is substitution invariant, we obtain $[\![\hat{Q}]\!] = Q$. Rule (COM$_\ell$) applies as follows:

$$\frac{\ell = (i_1, j_1, i_2, j_2) \quad \hat{\pi}_{i_1}^{j_1} = x?\tilde{y} \quad \hat{\pi}_{i_2}^{j_2} = x{:}r!\tilde{z} \quad |\tilde{y}| = |\tilde{z}|}{\hat{P} \xrightarrow[\ell]{r} \hat{Q}}$$

where

$$\hat{P} = (\nu\tilde{x}) \prod_{i=1}^{n} \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j \text{ and } \hat{Q} = (\nu\tilde{x})\Big(\prod_{\substack{i=1 \\ i\neq i_1, i_2}}^{n} \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j \mid \hat{P}_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid \hat{P}_{i_2}^{j_2} \Big)$$

Given two processes P, Q we define a set $I(P,Q) \subseteq \mathbb{R}_+^\infty \times \mathbb{N}^4$ and a number $S(P,Q) \in \mathbb{R}_+^\infty$ as used in rule (SUM) as follows:

$$I(P,Q) = \{(r,\ell) \mid \exists Q'.\ P \xrightarrow[\ell]{r} Q' \equiv Q\} \qquad \text{and} \qquad S(P,Q) = \sum_{(r,\ell)\in I(P,Q)} r$$

Claim. $S(P,Q) = S([\![P]\!], [\![Q]\!])$

Proof. It is sufficient to show that $I(P,Q) = I([\![P]\!], [\![Q]\!])$. There are two inclusions to be shown:

"$\subseteq$" If $(r,\ell) \in I(P,Q)$ then there exists Q' such that $P \xrightarrow[\ell]{r} Q' \equiv Q$. The first part of the previous claim shows that $[\![P]\!] \xrightarrow[\ell]{r} [\![Q']\!]$, and since translation preserves structural congruence also $[\![Q']\!] \equiv [\![Q]\!]$. Hence $(r,\ell) \in I([\![P]\!], [\![Q]\!])$.

"$\supseteq$" If $(r, \ell) \in I([\![P]\!], [\![Q]\!])$ then there exists Q'' such that $[\![P]\!] \xrightarrow[\ell]{r} Q'' \equiv [\![Q]\!]$. The second part of the previous claim shows that there exists Q' such that $P \xrightarrow[\ell]{r} Q'$ with $[\![Q']\!] \equiv Q'' \equiv [\![Q]\!]$. This implies $Q' \equiv Q$ since translation reflects structural congruence, so that $(r, \ell) \in I(P, Q)$.

Claim. Let Q be a process and $P_1 \equiv P_2$ processes. If P_1 and P_2 are prenex normal forms in which all bound variables are renamed apart, then $S(P_1, Q) = S(P_2, Q)$.

Proof. Suppose that $P_1 = (\nu x_1) \ldots (\nu x_k) \prod_{i=1}^{m} \sum_{j=1}^{n_i} M_i^j$ for guarded processes M_i^j. An analysis of the structural congruence shows that there exists a sequence of variables $(y_1, \ldots, y_k)$ and permutations $\sigma : \{1, \ldots, k\} \to \{1, \ldots, k\}$, $\theta : \{1, \ldots, m\} \to \{1, \ldots, m\}$, and $\theta_i : \{1, \ldots, n_i\} \to \{1, \ldots, n_i\}$ such that:

$$P_2 = (\nu y_{\sigma(1)}) \ldots (\nu y_{\sigma(k)}) \prod_{i=1}^{m} \sum_{j=1}^{n_i} {M'}_{\theta(i)}^{\theta_i(j)} \text{ and } M_i^j \equiv {M'}_i^j [y_{\sigma(1)}/x_1, \ldots, y_{\sigma(k)}/x_k]$$

Given this representation of P_2, and since all bound variables are renamed apart, it is easy to check that $(r, (\theta(i_1), \theta_{i_1}(j_1), \theta(i_2), \theta_{i_2}(j_2))) \in I(P_1, Q)$ iff $(r, (i_1, j_1, i_2, j_2)) \in I(P_2, Q)$.

We next prove the theorem for reductions with finite rates.

Claim. Translation preserves and reflects relations $\xrightarrow{r}$ for all $r \in \mathbb{R}_+$.

1. if $P \xrightarrow{r} P'$ then $[\![P]\!] \xrightarrow{r} [\![P']\!]$
2. if $[\![\hat{P}]\!] \xrightarrow{r} Q$ then there exists $\hat{Q}$ such that $\hat{P} \xrightarrow{r} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$.

Proof. 1. Assumption $P \xrightarrow{r} Q$ must be inferred by rule (SUM) as follows:

$$\frac{P \Downarrow P_1 \quad S(P_1, Q) = r \neq 0 \quad \neg\exists\ell\exists Q'.P_1 \xrightarrow[\ell]{\infty} Q'}{P \xrightarrow{r} Q}$$

We have shown in the proof of Theorem 2 that $P \Downarrow P_1$ implies $[\![P]\!] \Downarrow [\![P_1]\!]$. The second claim above shows that $S(P_1, Q) = S([\![P_1]\!], [\![Q]\!])$. The second part of the first claim above ensures that $\neg\exists\ell\exists Q'.[\![P_1]\!] \xrightarrow[\ell]{\infty} Q'$. Thus, the following rule is applicable:

$$\frac{[\![P]\!] \Downarrow [\![P_1]\!] \quad S([\![P_1]\!], [\![Q]\!]) = r \neq 0 \quad \neg\exists\ell\exists Q'.[\![P_1]\!] \xrightarrow[\ell]{\infty} Q'}{[\![P]\!] \xrightarrow{r} [\![Q]\!]}$$

2. We assume $[\![\hat{P}]\!] \xrightarrow{r} Q$ for $r \in \mathbb{R}_+$. Since the stochastic semantics refines the nondeterministic semantics by Proposition 5 we know that $[\![\hat{P}]\!] \to Q$. Theorem 2 on the preservation of the non-deterministic semantics shows that there exists a process $\hat{Q}$ such that $[\![\hat{Q}]\!] \equiv Q$ and $\hat{P} \to \hat{Q}$. In the following we

only make use of $[\![\hat{Q}]\!] \equiv Q$. Assumption $[\![\hat{P}]\!] \xrightarrow{r} Q$ must be inferred by rule (SUM):

$$\frac{[\![\hat{P}]\!] \Downarrow P_1 \quad S(P_1, Q) = r \neq 0 \quad \neg\exists\ell\exists Q'.P_1 \xrightarrow[\ell]{\infty} Q'}{[\![\hat{P}]\!] \xrightarrow{r} Q}$$

In particular, P_1 must be in prenex normal form, and w.l.o.g. all its variables are renamed apart. Since $[\![\hat{P}]\!] \Downarrow P_1$ there exists $\hat{P}_1$ such that $\hat{P} \Downarrow \hat{P}_1$ and $[\![\hat{P}_1]\!] \equiv P_1$, as we showed in the proof of Theorem 2. Process $\hat{P}_1$ is a prenex normal form, and we can assume w.l.o.g. that all its bound variables are renamed apart. The above claims show that:

$$S(P_1, Q) = S([\![\hat{P}_1]\!], [\![\hat{Q}]\!]) = S(\hat{P}_1, \hat{Q})$$

Since the translation reflects $\xrightarrow[\ell]{\infty}$ steps, we can apply rule (SUM) as follows:

$$\frac{\hat{P} \Downarrow \hat{P}_1 \quad S(\hat{P}_1, \hat{Q}) = r \neq 0 \quad \neg\exists\ell\exists Q'.\hat{P}_1 \xrightarrow[\ell]{\infty} Q'}{\hat{P} \xrightarrow{r} \hat{Q}}$$

Claim. The translation preserves and reflects immediate reactions:

1. if $P \xrightarrow{\infty(n)} P'$ then $[\![P]\!] \xrightarrow{\infty(n)} [\![P']\!]$
2. if $[\![\hat{P}]\!] \xrightarrow{\infty(n)} Q$ then exists $\hat{Q}$ such that $\hat{P} \xrightarrow{\infty(n)} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$.

We omit the proof of this claim. It concerns rule (COUNT), which can be treated quite similarly to rule (SUM) above. □

We can refine our translation such that types are preserved. This is necessary, since we need to translate type annotation on constants in λ-expressions and on newly created channels.

In order to do so, we assume that there exists a type constant R by which to type priorities $r \in R$ during translation. We refine the translation for restriction and output prefixes as follows:

$$[\![(\nu x{:}\tau)P]\!] = (\nu x{:}[\![\tau]\!])[\![P]\!]$$
$$[\![x{:}r!\tilde{y}.P]\!] = x[r_R]!\tilde{y}.[\![P]\!]$$

Types of the π-calculus with priorities are translated to types of $\pi(\lambda(R)_<)$:

$$[\![ch(\tau_1, \ldots, \tau_n)]\!] = [R \to R] \Rightarrow ([\![\tau_1]\!], \ldots, [\![\tau_n]\!])$$

The translation can be lifted homomorphically to type environments $[\![\Gamma, \Delta]\!] = [\![\Gamma]\!], [\![\Delta]\!]$.

Proposition 8 (Type preservation). *Let P be a process of the π-calculus with priorities and Γ a type environment such that $\Gamma \vdash P$ then $[\![\Gamma]\!] \vdash [\![P]\!]$.*

The proof is straightforward by structural induction over type derivations.

5.2 Encoding π@ for Dynamic Compartments

We present an encoding of the π-calculus with polyadic synchronization and priorities π@ [18] into the attributed π-calculus, such that it inherits encodings of BioAmbients [13] and Brane calculus [14], in which different systems with dynamic compartment organizations can be defined. The correctness of the encoding of π@ can be shown only wrt. the non-deterministic semantics, since π@ lacks a stochastic semantics in terms of CTMCs (there only exists a stochastic simulation algorithm for sπ@ that can deal with dynamic compartments of variable volumes [22]).

We show how to encode π@ with priorities in an ordered set $(R, <)$ into the attributed π-calculus with priorities $\pi(\lambda(R, =)_<)$. This result shows that the attributed π-calculus with 3 levels of priorities inherits correct encodings of BioAmbients and Brane from π@.

The syntax of π@ is the same as for the π-calculus with priorities, except that communication now acts on nonempty tuples of channels and that priorities are assigned to both senders and receivers. This means that prefixes now have the following form, where $|\tilde{x}| \geq 1$:

$$\text{polyadic prefixes} \qquad \pi ::= \tilde{x}{:}r?\tilde{y} \mid \tilde{x}{:}r!\tilde{z}$$

The communication rule (COM) is adapted in such a way that tuples of channels and priorities are tested for equality before communication. Otherwise, the non-deterministic semantics of the π-calculus with priorities remains unchanged:

$$(\text{COM}_@)\ \frac{|\tilde{y}| = |\tilde{z}|}{\tilde{x}{:}r?\tilde{y}.P_1 + M_1 \mid \tilde{x}{:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{z}/\tilde{y}] \mid P_2}$$

We decompose the encoding of π@ in two parts. The first part is a preprocessing step that rewrites all tuples in sending or receiving positions, such that they obtain the same arity. Given a process P of π@, let n be the maximal arity of tuples in subject position of polyadic prefixes and x a fresh channel name not occurring in P (which exists since *Vars* is infinite). Sending and receiving tuples in P are completed by x'es until they are of arity n:

$$(x_1, \ldots, x_m) \Rightarrow (x_1, \ldots, x_m, \underbrace{x, \ldots, x}_{n-m})$$

We recursively define functions eq_n that check equality of n-tuples of constants or variables (and thus channel names):

$$\begin{array}{l} eq_0() =_{df} \texttt{true} \\ eq_n(x_1, \ldots, x_n, y_1, \ldots, y_n) =_{df} \\ \qquad \texttt{if } x_1{=}y_1 \texttt{ then } eq_{n-1}(x_2 \ldots, x_n, y_2, \ldots, y_n) \texttt{ else false} \end{array}$$

Lemma 5. *For all constants or variables $v_1, \ldots, v_n, v'_1, \ldots, v'_n$ it is true that:*

1. $eq_n(v_1, \ldots, v_n, v_1, \ldots, v_n) \Downarrow$ ***true***.

2. $eq_n(v_1, \ldots, v_n, v'_1, \ldots, v'_n) \Downarrow \texttt{false}$ *if* $v_i \neq v'_i$ *for some* $1 \leq i \leq n$.

The proof is straightforward by induction on n. It relies on the definition of conditionals and equality in the big-step evaluator in Figs. 11 and 12. See Appendix B for details.

The main translation $[\![_]\!] : \pi@ \rightarrow \pi(\lambda(R, =)_<)$ maps to an attributed π-calculus with additional constants for priorities and equality. Only priorities are successful values. We define encoding $[\![_]\!]$ to be compositional, so that we have only to specify the encoding of communication prefixes. Here, we assume that all subject tuples have the same arity n.

$$\begin{aligned}
[\![(x_1, \ldots, x_n){:}r!\tilde{z}.P]\!] &= x_1[\lambda y_2 \ldots \lambda y_n \lambda r'. \\
&\qquad \texttt{if}\ eq_n(x_2, \ldots, x_n, r, y_2, \ldots, y_n, r')\ \texttt{then}\ r]!\tilde{z}.[\![P]\!] \\
[\![(y_1, \ldots, y_n){:}r'?\tilde{z}.P]\!] &= y_1[\lambda u.u\ y_2 \ldots y_n r']?\tilde{z}.[\![P]\!]
\end{aligned}$$

A sender on channel tuple $(x_1, \ldots, x_n)$ with priority r is translated to a sender on a single channel x_1 and a constraint that checks equality with $(x_2 \ldots x_n, r)$ and if successful returns r. Symmetrically, we translate a receiver on channel tuple $(y_1, \ldots, y_n)$ with priority r' to a receiver on a single channel y_1 that receives the constraint and applies it to $(y_2, \ldots, y_n, r')$. Of course, we also need to translate all process definitions:

$$[\![A(\tilde{x}) \triangleq P]\!] = A(\tilde{x}) \triangleq [\![P]\!]$$

Theorem 4. *The encoding of* $\pi@$ *with priorities in* $(R, <)$ *to the attributed* π*-calculus with priorities* $\pi(\lambda(R, =)_<)$ *is correct, in that all reprocessed processes* P *and* P' *of* $\pi@$ *and attributed processes* Q *satisfy:*

(a) if $P \rightarrow Q$ *then* $[\![P]\!] \rightarrow [\![Q]\!]$
(b) if $[\![P]\!] \rightarrow Q$ *then exists* $\hat{Q}$ *such that* $Q \equiv [\![\hat{Q}]\!]$ *and* $P \rightarrow \hat{Q}$.

The proof is elaborated in Appendix B. It checks that communication steps correspond in both calculi, i.e., that polyadic synchronization of $\pi@$ is translated properly to equality testing in $\pi(\lambda(R, =)_<)$. This mostly follows from Lemma 5 on the correctness of encoding equality of n-tuples.

Finally, notice that the encoding of $\pi@$ does not preserve types in any obvious sense. Finding a convincing type system for $\pi@$ is nontrivial, since there the capabilities of tuples and channels are overloaded while usual type system separate tuples and channel types properly.

5.3 Variants of the Stochastic Pi-Calculus

It remains to discuss the relationship to variants of the stochastic π-calculus where rates are annotated to channels.

BioSpi and spim. The syntax of BioSpi [12] and SPiM [21] differs from ours in that stochastic rates are annotated to channels at creation time, rather than to communication prefixes. The rates of the prefixes can then be deduced from the rate of the communicating channel.

The idea of encoding this variant of the stochastic π-calculus into $\pi(\lambda(\mathbb{R}_+^\infty)_{<_2})$ is to replace channels x with rate r by pairs $\langle x, r\rangle$, that are decomposed at communication time. Here it is relevant that the attributed π-calculus permits pairs, and that it allows for expressions in sender and receiver positions.

We obtain the encoding below that we claim to be correct with respect to both semantics, non-deterministic and stochastic (without proof).

$$\begin{array}{l}
[\![(\nu x{:}r)P]\!] = (\nu x)[\![P]\!][\langle x, r\rangle/x] \\
[\![x?\tilde{y}.P]\!] = (\texttt{fst}\ x)[\lambda z.z]?\tilde{y}.[\![P]\!] \\
[\![x!\tilde{y}.P]\!] = (\texttt{fst}\ x)[\texttt{snd}\ x]?\tilde{y}.[\![P]\!]
\end{array}$$

The first line states, how to remove the annotation r of a channel x, and to substitute all occurrences of x by the $\langle x, r\rangle$. The next two lines state that the channel is extracted from the pair before communication. In the third line, the rate is extracted in the communication constraint.

Stochastic Pi-Calculus with Concurrent Objects. The next more expressive language is SPiCO, the stochastic π-calculus for concurrent objects [20]. It supports a static form of polyadic synchronization, called pattern guarded inputs.

Patterns are tuples $a(\tilde{y})$ that are built from a finite set of function symbols a in some set Σ and a sequence of channels. Senders send tuples $b(\tilde{z})$ to receivers, which match it against a pattern $a(\tilde{y})$. A communication step is allowed only if the function symbol b of the tuple sent matches the function symbol a of the receiving pattern:

$$x?a(\tilde{y}).P \mid x!b(\tilde{z}).P' \rightarrow P[\tilde{z}/\tilde{y}] \mid P' \text{ if } a = b$$

The communication constraint is thus equality $a{=}b$. This is a weak form of polyadic synchronization, since the sending and receiving channels must be checked for equality too. As before, stochastic rates are annotated to channels at creation time. We can encode SPiCO into $\pi(\lambda(\mathbb{R}_+^\infty, \Sigma, =))_{<_2}$ similarly as before, where $a, b \in \Sigma$:

$$\begin{array}{l}
[\![(\nu x{:}r)P]\!] = (\nu x)[\![P]\!][\langle x, r\rangle/x] \\
[\![x?a(\tilde{y}).P]\!] = (\texttt{fst}\ x)[\lambda z.\texttt{if}\ \ z{=}a\ \ \texttt{then}\ \ (\texttt{snd}\ x)]?\tilde{y}.[\![P]\!] \\
[\![x!b(\tilde{y}).P]\!] = (\texttt{fst}\ x)[b]?\tilde{y}.[\![P]\!]
\end{array}$$

The only new aspect here is that we have to check the communication constraint $a{=}b$ in addition.

```
Simulate−naive(P, t)
  // process P, time point t ∈ ℝ
  let P1 such that P ⇓ P1
      // P1 is obtained from P by exhaustively applying definitions.
      // This computation may diverge.
  if P1 --err/nd--> ⊥ then raise error
     // Apply all rules (E.COM), (E.PREF), (E.CONSTR).
     // This computation may diverge since expressions are to be evaluated.
  let Reacts = {(ℓ, r) ∈ ℕ^4 × ℝ+^∞} | ∃P2. P1 --r/ℓ--> P2}  // (COM_ℓ)
  if Reacts ∩ (ℕ4 × {∞}) = ∅
  then
     let ((ℓ, r), Δ) = Gillespie(Reacts)  // (SUM)
     let P2 such that P1 --r/ℓ--> P2
     Simulate−naive(P2, t + Δ)
  else
     select (ℓ, ∞) ∈ Reacts with equal probability //(COUNT)
     let P2 such that P1 --r/ℓ--> P2
     Simulate−naive(P2, t)
```

Fig. 26. Naive simulator interpreting the stochastic semantics.

Original Attributed Pi-Calculus. Finally, in the conference version of the attributed π-calculus at CMSB'08 [25], we annotated stochastic rates to channels, and used a fixed function `val` mapping channels to their rates. This version of $\pi(\mathcal{L})$ can be encoded into the version of $\pi(\mathcal{L})$ presented here:

$$
\begin{array}{lll}
[\![(\nu x{:}v)P]\!] = (\nu x)[\![P[\langle x, v\rangle/x]]\!] & [\![\texttt{val}]\!] = \texttt{snd} & [\![x]\!] = x \\
[\![v[e]?\tilde{v}.P]\!] = (\texttt{fst}\ [\![v]\!])([\![e]\!]?[\![\tilde{v}]\!].[\![P]\!] & [\![\lambda x.e]\!] = \lambda x.[\![e]\!] & \\
[\![v[e]!\tilde{y}.P]\!] = (\texttt{fst}\ [\![v]\!])([\![e]\!]!\tilde{y}.[\![P]\!] & [\![e\ e']\!] = [\![e]\!][\![e']\!] &
\end{array}
$$

6 Stochastic Simulator

We develop a stochastic simulation algorithm that closely follows the stochastic semantics of the attributed π-calculus in terms of CTMCs. Thereby, we show that a simulator for $\pi(\mathcal{L})$ can be obtained independently of the choice of $\mathcal{L}$ by extending previous simulators for the stochastic π-calculus or SPiCO [11, 10, 12].

The stochastic semantics for $\pi(\mathcal{L})$ induces the naive stochastic simulator given in Fig. 26. A simulator's input comprises a process P and a time point $t \in \mathbb{R}$. The next reduction step for process P is chosen in a memoryless stochastic manner. The sojourn time $\Delta \in \mathbb{R}_+$ of P is inferred, and the simulator proceeds with the resulting solution at time point $t + \Delta$. This loop continues until no next reduction step can be found, in fact it may run for ever, if not interrupted externally or equipped with some additional termination condition.

The first step of the simulation algorithm is to apply definitions of P exhaustively. This computation may run into an infinite loop or raise errors, in case of non well-founded definitions or if the evaluation of some expressions diverges ($\neg\exists v.e \Downarrow v$). If application raises an immediate error $P_1 \xrightarrow[nd]{err} \bot$ by rules (E.COM), (E.PREF), or (E.CONSTR), then the simulator throws an exception (which kills its continuation). Note that error checking may run into infinite loops or raise an error too. If P does converge to an error-free process P_1 then P_1 is uniquely determined up to structural congruence (Proposition 4) and must be congruent to some prenex form $(\nu\tilde{x})\prod_{i=0}^{n} M_i$. The remainder of the algorithm is independent of the concrete representative of congruence class $[P_1]_{\equiv}$, so that we can chose this representative arbitrarily. The next step is to compute the set of all labeled reactions of P_1:

$$\mathit{Reacts} = \{(\ell, r) \in \mathbb{N}^4 \times \mathbb{R}_+^\infty \mid \exists P_2.\ P_1 \xrightarrow[\ell]{r} P_2\}$$

Labeled reactions with rate $r = \infty$ are executed with priority and without time consumption. If no reaction with rate $r = \infty$ exists, we apply Gillespie's algorithm [45] to select a reaction $(\ell, r) \in \mathit{Reacts}$ with probability r/s where $s = \sum_{(\ell,r')\in \mathit{Reacts}} r'$. The sojourn time in P is $\Delta = -ln(1/U)/s$ for some uniformly distributed random number $0 < U \leq 1$.

In order to compute *Reacts*, we have to enumerate all possible instances of the communication rule (COM_ℓ). This requires to evaluate all evaluation constraints, by applying the evaluation algorithm of the attribute language $\mathcal{L}$.

Most fortunately, the CTMC itself does not need to be computed by the simulation algorithm. This would be largely unfeasible, since the number of possible outcomes of non-deterministic interactions may grow exponentially. Furthermore, it would require to decide structural congruence (rules (SUM) and (COUNT)), which is a graph isomorphism complete problem [46].

In order to increase efficiency of the naive simulation algorithm, we apply an idea exploited already in the BioSpi implementation [12]. The objective is to avoid the enumeration of all pairs of alternatives (and thus redexes), since there may be quadratically many in the size of P_1. The strategy is to *group* all reactions on the same channel with the same rate. We apply Gillespie's algorithm to such *grouped reactions* and then choose a specific interaction with equal distribution.

In order to identify grouped reactions, we introduce group labels. A *group label* of a process P_1 is a triple in $\mathit{fv}(P_1) \times \mathit{Vals}(P_1)^2$. The group of reactions for $P_1 = \prod_{i=1}^{n}\sum_{j=1}^{m} \pi_i^j.P_i^j$ with label $L = (x, v, r)$ is defined as follows:

$$\begin{aligned}\mathit{Reacts}(L) = \{((i_1, j_1, i_2, j_2), r) \in \mathit{Reacts} \mid \\ \exists v' \exists \tilde{y} \exists \tilde{v}.\pi_{i_1}^{j_1} \Downarrow x[v]!\tilde{y},\ \pi_{i_2}^{j_2} \Downarrow x[v']?\tilde{v},\ v'v \Downarrow r\}\end{aligned}$$

A triple L identifies reaction groups by a communication channel x, a constraint value of senders v, and a rate r yielding the application of the receivers λ-abstraction v' to v.

The stochastic rate for a grouping label L is usually called propensity $\mathit{prop}(L) \in \mathbb{R}^+ \uplus \{\infty(n) \mid n \in \mathbb{N}\}$. It sums up all rates of the labeled reactions that are grouped together, or counts the number of labels of infinite rate

```
Simulate(P,t)  //solution P, time point t ∈ ℝ
  let P1 be such that P ⇓ P1
      // P1 is obtained from P by exhaustively applying definitions.
      // This computation may diverge.
  if P1 --err/nd--> ⊥ then raise error
     // Apply all rules (E.COM), (E.PREF), (E.CONSTR).
     // This computation may diverge since expressions are to be evaluated.
  let GReacts = {(L, prop(L)) | L ∈ Vars(P1) × Vals(P1)^2}
  if {(L,r) ∈ GReacts | r = ∞(n)} = ∅
  then
     let ((L,r), Δ) = Gillespie(GReacts)
     select (ℓ,r) ∈ Reacts(L) with equal probability
     let P2 such that P1 --r/ℓ--> P2
     Simulate(P2, t + Δ)
  else
     select (L, ∞(n)) ∈ GReacts
        with probability n/m where m = Σ_{(L',∞(n'))∈GReacts} n'
     select (ℓ,∞) ∈ Reacts(L) with equal probability
     let P2 such that P1 --∞/ℓ--> P2
     Simulate(P2, t)
```

Fig. 27. Stochastic simulator for $\pi(\mathcal{L})$ (to be implemented incrementally).

reactions if there are any:

$$prop(L) = \begin{cases} \infty(n) & \text{if } n = \#\{\ell \mid (\ell,\infty) \in Reacts(L)\} \geq 1 \\ \sum_{(\ell,r)\in Reacts(L)} r & \text{otherwise} \end{cases}$$

We define the set of grouped reactions with their propensities as follows. These will be used as input of the Gillespie's algorithm:

$$GReacts = \{(L, prop(L)) \mid L \in Vars(P_1) \times Vals(P_1)^2\}$$

The cardinality of *GReacts* is linear in the size of P_1. In many practically relevant cases, only a fixed number of values will ever be used. This is e.g. the case in our example models of Euglena's movement and cooperative enhancement, where none of the processes succeeding the initial solution introduces new channels or new constraint values, see Section 4. By contrast, the cardinality of set *Reacts* becomes quadratic in the size of P_1, e.g., if all senders and receivers may interact.

Fig. 27 gives a simulation algorithm based on grouped reactions. In contrast to the naive simulator, it first selects a grouped reaction based on Gillespie's algorithm, and then a label of a reaction within this group with equal distribution.

What remains is to compute the propensities of all labels of grouped reactions in a process P_1. These can be derived from the values below if $P_1 =$

$\prod_{i=1}^{n} \sum_{j=1}^{m} \pi_i^j.P_i^j$:

$$\begin{aligned} \mathit{out}(x,v) &= \#\{(i,j) \mid \exists \tilde{v} : \ \pi_i^j \Downarrow x[v]!\tilde{v}\} \\ \mathit{in}(x,v,r) &= \#\{(i,j) \mid \exists v' \exists \tilde{y} : \pi_i^j \Downarrow x[v']?\tilde{y},\ v'v \Downarrow r\} \\ \mathit{mixin}(x,v,r) &= \#\{(i,j_1,j_2) \mid \exists v' \exists \tilde{v} \exists \tilde{y} : \ \pi_i^j \Downarrow x[v]!\tilde{v},\ \pi_i^j \Downarrow x[v']?\tilde{y},\ v'v \Downarrow r\} \end{aligned}$$

Lemma 6. *$\mathit{prop}(x,v,r) = (\mathit{out}(x,v) * \mathit{in}(x,v,r) - \mathit{mixin}(x,v,r)) * r$, if the solution does not contain infinite rates.*

Proof. Let $L = (x,v,r)$. It is enough to show that $\mathit{out}(x,v) * \mathit{in}(x,v,r) - \mathit{mixin}(x,v,r) = \#\mathit{Reacts}(L)$. This follows, since all pair of indices counted by $\mathit{out}(x,v) * \mathit{in}(x,v,r)$ form a redex according to rule (COM) except for those that are counted by $\mathit{mixin}(x,v,r)$.

The computation of mixins can still produce an output of quadratic size and thus need quadratic time. The square factor, however, is in the maximal number of alternatives in sums defining molecules of P_1, which will be small in practice. All other needed values can be computed in linear time in the size of P_1, when ignoring the time for evaluating expressions, which is justified in many practical cases.

The final step toward an efficient simulator consists in computing the propensities $\mathit{prop}(x,v,r)$ incrementally, so that they do not need to be recomputed from scratch in every reduction step. This can be based on Lemma 6, since the values of $\mathit{out}(x,v)$, $\mathit{in}(x,v,r)$, $\mathit{mixin}(x,v,r)$ can be updated incrementally, when adding new solutions or canceling alternative choices by communication.

7 Implementation and Performance Evaluation

We discuss our implementation of the stochastic simulator for $\pi(\mathcal{L})$, and present some experimental results in order to give an impression of its performance.

We implemented the $\pi(\mathcal{L})$ simulator on top of an existing simulator for the stochastic π-calculus in the modeling and simulation framework JAMES II [29]. The implementation is freely available[6]. We deployed a two layer approach: the base layer is the simulator of the stochastic π-calculus along the lines of [21], i.e. for each communication channel the propensity is calculated under consideration of the corresponding senders and receivers. The obtained propensities are passed to a Stochastic Simulation Algorithm (SSA) that determines the next communication to perform and the sojourn time. There are three alternative SSAs that can be freely chosen, the First Reaction Method (the original version), the Direct Reaction Method [47], and the Next Reaction Method [48]. The top layer implements the grouping as explained in Section 6, i.e. it groups the communication pairs in a solution by the combinations of channels and rate constants resulting from the application of receiver abstractions to sender arguments. For each group it creates a communication channel and assigns the rate constant and

[6] See the James-Imp-Pi web page at http://biopi-lille-ros.gforge.inria.fr.

the corresponding senders and receivers to it. The set of thus obtained communication channels is passed to the base layer in order to determine the following solution.

In our performance experiments, we compare the simulators for $\pi(\mathcal{L})$ and the stochastic π-calculus based on JAMES II, with the stochastic Pi Machine (SPiM) by Phillips and Cardelli [11]. We only consider the Direct Reaction Method, since this is the SSA chosen by SPiM. Our experiments are performed on a WindowsXP machine, with an Intel Core 2 Duo 2.00 GHz processor and 2 GB RAM providing a SciMark 2.0 Java benchmark score [49] of 383.9 Mflops. Notice, that there exists a faster version of SPiM for Linux based on native code compilation, an aspect that is irrelevant for our comparison.

As test example, we use the Euglena model from Section 4.1, since it allows us to gradually raise the number of grouped reactions and process definitions by increasing the number of depth levels. Furthermore, it can be implemented in both the stochastic π-calculus and $\pi(\mathcal{L})$. Our Euglena benchmark model comprises two light source with intensity rates $\mathtt{I}_1 = 5.0$ and $\mathtt{I}_2 = 15.0$ and 100 Euglenas on each depth level ($\mathtt{n} = 100$). The rate of Euglena's upwards motion equals $\mathtt{u} = 2.0$ and the water opacity is set to $\sigma = 0.2$. Among the experiments, we gradually increased the number of depth levels from 10 to 100 by steps of 10, i.e. $\mathtt{m} \in \{9, 19, \ldots, 99\}$. Implementations in the stochastic π-calculus are obtained by enumerating the Euglena processes for different depth levels in the same way as shown in Section 4.1. To ensure comparability, we used two $\pi(\mathcal{L})$ implementations for each experiment, one enumerating the depth levels as in the stochastic π-calculus (enum) and one in the more compact form with the depth level as a parameter of Euglena (comp). We measured the time needed to simulate until time point 100.0, see Appendix A. For each experiment, we averaged over three simulation runs with small deviations resulting from both the stochastic nature of the simulation and the work load of the machine. The results of the experiment sets are shown in Figs. 28. The implementations are labeled according to the used formalism, *Sto* for the stochastic π-calculus or *attr* for $\pi(\mathcal{L})$, the tool, SPiM or JAMES II, and the implementation, *Enum* or *Comp*.

The results in Fig. 28 show a general increase of simulation time with a rising number of depth levels. Presumably, due to our choice of operating system, SPiM performs slower. All other implementations need similar amounts of time. The maximal simulation time required is around $160s$. Our results indicate that the computational complexity of the $\pi(\mathcal{L})$ simulator is moderate.

8 Conclusion and Outlook

We presented the attributed π-calculus in order to define attribute processes with interaction constraints depending on attribute values. We used the call-by-value λ-calculus as a sequential language in which to define data values and constraints for concurrent interactions.

The attributed π-calculus forms a unifying framework, which extends on the one hand the π-calculus with priorities and its extension by polyadic synchro-

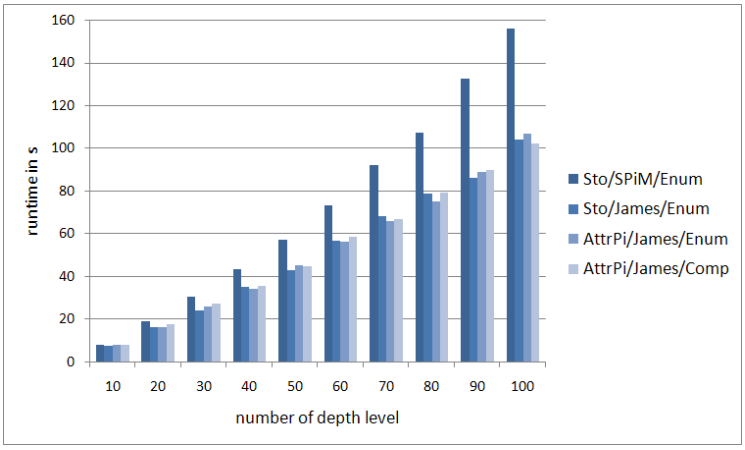


Fig. 28. Runtime of different simulators in s for the Euglena model with two light sources with intensity rates $\mathtt{I}_1 = 5.0, \mathtt{I}_2 = 15.0$, 100 Euglenas on each depth level ($\mathtt{n} = 100$), an opacity $\sigma = 0.2$, a rate for upward motion $\mathtt{u} = 2.0$, and different numbers of depth levels ranging from 10 to 100, i.e. $\mathtt{m} \in \{9, 19, \ldots, 99\}$. Simulation runs were performed until simulation time $t = 100.0$: Sto = stochastic π-calculus, Attr = attributed π-calculus, SPiM = SPiM, James = JamesII, Enum = model with enumerated depth levels, Comp = model with depth level as species parameter.

nization π@, and, on the other hand, the stochastic π-calculus, and its extension by concurrent objects SPiCO. This allows us to express existing π-calculus models of biological systems in the attributed π-calculus. Furthermore, the attributed π-calculus permits to model spatial aspects in systems biology depending on numeric attributes, and dynamic compartments with various nesting structures such as in BioAmbients and Brane.

Due to this combination of a concurrent process language, i.e. the π-calculus, and a sequential core language, i.e. the λ-calculus, the modeling of biological systems becomes possible on multiple levels. In addition to the traditional individual-based modeling of the stochastic π-calculus which maps molecules to objects, the attributed π-calculus enables a population-based modeling style. In combining both levels within one model priorities play a key role. Thereby, n-ary chemical reactions with $n > 2$ reactions and diverse kinetics can be realized. E.g. in a recent model of the Wnt-pathway, which is aimed at analyzing the cytoplasmic-nuclear shuttling of β-catenin, Michaelis-Menten dynamics that are defined at population level interact with other processes described at individual-based level. Thereby the objective of the simulation study and data availability are taken into account [51].

We have presented and implemented a stochastic simulator for the attributed π-calculus, which shows that this general approach is feasible in practice with moderate performance.

The imperative π-calculus is a recent extension of the attributed π-calculus with a global imperative store [44]. The original motivation was to enable a simpler encoding of dynamic compartments, that does not rely on priorities. Note that a global imperative store is equally supported by sCCP [16], but in sCCP all interactions are forced to pass through the global store.

Quite some questions remain for future research. The first question refers to the precise relationship of individual-based modeling in the attributed π-calculus to rule-based modeling in the kappa calculus [52, 7] or Bigraphs [9]. Independently of population-based modeling, it would be interesting to know whether sCCP can be expressed by the attributed π-calculus, and similarly, independently of species-based modeling, it would be of interest to know whether Bio-Pepa can be encoded into the attributed π-calculus. There is also the question, whether there exists a fragment of $\pi(\lambda(R, =)_<)$ that corresponds precisely to $\pi@$ with priorities in $(R, <)$. Finally, there is an open issue to develop and implement object-oriented abstractions with inheritance for attributed π-calculus, in the spirit of spiCo [10].

Acknowledgements

We thank Francois Lemaire for helping us to compute the equilibrium level for our Euglena models, Stefan Rybacki for his support in implementing the attributed π-calculus, and Céline Kuttler for having initiated the present cooperation between Lille and Rostock. The anonymous reviewers send us valuable comments, that helped us improving the article.

This research has been supported by the Anr jeune project BioSpace (2009-2012) of the national French research founding agency and as part of the Research Training School "dIEM oSiRiS" (2006-2011) of the Deutsche Forschungsgemeinschaft (DFG).

References

1. Hillston, J.: Process algebras for quantitative analysis. In: 20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings, IEEE Comp. Soc. Press (2005) 239–248
2. Cardelli, L.: On process rate semantics. Theoretical Computer Science **391** (2008) 190–215
3. Chabrier-Rivier, N., Fages, F., Soliman, S.: The Biochemical Abstract Machine BIOCHAM. In: Computational Methods in Systems Biology. (2004) 172–191
4. Regev, A.: Computational Systems Biology: A Calculus for Biomolecular Knowledge. Tel Aviv University (2003) PhD thesis.
5. Regev, A., Shapiro, E.: Cells as Computation. Nature **419** (2002) 343
6. Gilbert, D., Heiner, M., Lehrack, S.: A unifying framework for modelling and analysing biochemical pathways using petri nets. In: Computational Methods in Systems Biology, International Conference, CMSB 2007. Volume 4695 of Lecture Notes in Computer Science., Springer Verlag (2007) 200–216

7. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: CONCUR - Concurrency Theory, 18th International Conference. Volume 4703 of Lecture Notes in Computer Science., Springer Verlag (2007) 17–41
8. Faeder, J.R., Blinov, M.L., Goldstein, B., Hlavacek, W.S.: Rule-Based Modeling of Biochemical Networks. Complexity **10** (2005) 22–41
9. Krivine, J., Milner, R., Troina, A.: Stochastic bigraphs. In: 24th Conference on the Mathematical Foundations of Programming Semantics. Volume 218 of Electronical notes in theoretical computer science., Elsevier (2008) 73–96
10. Kuttler, C., Lhoussaine, C., Niehren, J.: A stochastic pi calculus for concurrent objects. In: Second International Conference on Algebraic Biology. Volume 4545 of Lecture Notes in Computer Science., Springer Verlag (2007) 232–246
11. Phillips, A., Cardelli, L.: Efficient, correct simulation of biological processes in the stochastic pi-calculus. In: Computational Methods in Systems Biology, International Conference. Volume 4695 of Lecture Notes in Computer Science., Springer Verlag (2007) 184–199
12. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes. Information Processing Letters **80** (2001) 25–31
13. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: BioAmbients: An Abstraction for Biological Compartments. TCS **325** (2004) 141–167
14. Cardelli, L.: Brane calculi. In: Computational Methods in Systems Biology, International Conference CMSB 2004. Volume 3082 of Lecture Notes in Computer Science., Springer Verlag (2005) 257–278
15. Ciocchetta, F., Hillston, J.: Bio-PEPA: An Extension of the Process Algebra PEPA for Biochemical Networks. ENTCS **194** (2008) 103–117
16. Bortolussi, L., Policriti, A.: Modeling biological systems in stochastic concurrent constraint programming. Constraints, an International Journal **13** (2008) 66–90
17. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in pi-calculus. Nordic Journal of Computing **10** (2003) 70–98
18. Versari, C.: A Core Calculus for a Comparative Analysis of Bio-inspired Calculi. Programming Languages and Systems (2007) 411–425
19. Priami, C.: Stochastic π-calculus. Computer Journal **6** (1995) 578–589
20. Kuttler, C., Lhoussaine, C., Niehren, J.: A stochastic pi calculus for concurrent objects. In: 1st International Workshop on Probabilistic Automata and Logics. (2006)
21. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Proceedings of BioConcur '04. (2004)
22. Versari, C., Busi, N.: Stochastic simulation of biological systems with dynamical compartment structure. In: Computational Methods in Systems Biology, International Conference. Volume 4695 of Lecture Notes in Computer Science., Springer Verlag (2007) 80–95
23. Jaffar, J., Lassez, J.L.: Constraint Logic Programming. In: POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, New York, NY, USA, ACM (1987) 111–119
24. Saraswat, V.A., Rinard, M.C.: Concurrent constraint programming. In: ACM SICPLAN-SIGACT Symposium on Principles of Programming Languages, ACM-Press (1990) 232–245
25. John, M., Lhoussaine, C., Niehren, J., Uhrmacher, A.: The attributed pi calculus. In: Computational Methods in Systems Biology, 6th International Conference. Volume 5307 of Lecture Notes in Computer Science., Springer Verlag (2008) 83–102

26. Kuttler, C., Niehren, J.: Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. Transactions on Computational Systems Biology (2006) 24–55
27. Kuttler, C.: Modeling Bacterial Gene Expression in a Stochastic Pi Calculus with Concurrent Objects. PhD thesis, Universit des Sciences et Technologies de Lille - Lille 1 (2007)
28. Versari, C.: A Core Calculus for the Analysis and Implementation of Biologically Inspired Languages. PhD thesis, University of Bologna (2009)
29. Himmelspach, J., Uhrmacher, A.M.: Plug'n Simulate. In: ANSS '07: Proceedings of the 40th Annual Simulation Symposium, Washington, DC, USA, IEEE Computer Society (2007) 137–143
30. Baldamus, M., Parrow, J., Victor, B.: A fully abstract encoding of the pi-calculus with data terms. In: Automata, Languages and Programming, 32nd International-Colloquium. Volume 3580 of Lecture Notes in Computer Science., Springer Verlag (2005) 1202–1213
31. Johansson, M., Parrow, J., Victor, B., Bengtson, J.: Extended pi-calculi. In: Automata, Languages and Programming, 35th International Colloquium. Volume 5126 of Lecture Notes in Computer Science., Springer Verlag (2008) 87–98
32. Guerriero, M.L., Priami, C., Romanel, A.: Modeling static biological compartments with beta-binders. In: Algebraic Biology, Second International Conference. Volume 4545 of Lecture Notes in Computer Science., Springer Verlag (2007) 247–261
33. Priami, C., Quaglia, P., Romanel, A.: Blenx static and dynamic semantics. In: Concurrency Theory, 20th International Conference, CONCUR 2009. Volume 5710 of Lecture Notes in Computer Science., Springer Verlag (2009) 37–52
34. Maurin, M., Magnin, M., Roux, O.H.: Modeling of genetic regulatory network in stochastic pi-calculus. In: Bioinformatics and Computational Biology, First International Conference. Volume 5462 of Lecture Notes in Computer Science., Springer Verlag (2009) 282–294
35. Lecca, P.: Stochastic pi-calculus models of the molecular bases of parkinson's disease. In: International Conference on Bioinformatics and Computational Biology. (2008) 298–304
36. Niehren, J.: Uniform confluence in concurrent computation. Journal of Functional Programming **10** (2000) 453–499
37. Huet, G.P.: Confluent reductions: Abstract properties and applications to term rewriting systems. Journal of the ACM **27** (1980) 797–821
38. Kuttler, C., Lhoussaine, C., Nebut, M.: Rule-based modeling of transcriptional attenuation at the tryptophan operon. Transactions on Computational Systems Biology (2009)
39. Tait, W.W.: Intensional interpretations of functionals of finite type i. Journal of Symbolic Logic **32** (1967) 198–212
40. Mitchell, J.C.: Foundations for Programming Languages. MIT Press (1996)
41. John, M., Ewald, R., Uhrmacher, A.M.: A Spatial Extension to the Pi Calculus. ENTCS **194** (2008) 133–148
42. Kholodenko, B.N.: Cell-Signalling Dynamics in Time and Space. Nature Reviews Molecular Cell Biology **7** (2006) 165–176
43. Grell, K.G.: Protozoologie. Springer-Verlag (1968)
44. John, M., Lhoussaine, C., Niehren, J.: Dynamic compartments in the imperative pi calculus. In: Computational Methods in Systems Biology, 7th International Conference. Volume 5688 of Lecture Notes in Computer Science., Springer Verlag (2009) 235–250

45. Gillespie, D.T.: A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. Journal of Computational Physics **22** (1976) 403–434
46. Khomenko, V., Meyer, R.: Checking pi-calculus structural congruence is graph isomorphism complete. Technical Report CS-TR: 1100, School of Computing Science, Newcastle University (2008) 20 pages.
47. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81** (1977) 2340–2361
48. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. **104** (2000) 1876–1889
49. Pozo, R., Miller, B.: SciMark 2.0. http://math.nist.gov/scimark2/ (2009)
50. Degenring, D., Roehl, M., Uhrmacher, A.: Discrete event, multi-level simulation of metabolite channeling. BioSystems **1–3** (2004) 29–41
51. Mazemondet, O., John, M., Maus, C., Uhrmacher, A.M., Rolfs, A.: Integrating diverse reaction types into stochastic models - a signaling pathway case study in the imperative pi-calculus. In Rossetti, M.D., Hill, R.R., Johansson, B., Dunkin, A., Ingalls, R.G., eds.: Proceedings of the Winter Simulation Conference. (2009 (to appear))
52. Danos, V., Laneve, C.: Formal molecular biology. Theoretical Computer Science **325** (2004) 69–110

A Experiment results

Levels	Sto/SPiM/Enum	Sto/James/Enum	AttrPi/James/Enum	AttrPi/James/Comp
10	8.10 (0.09)	7.33 (0.58)	8.00 (0.00)	8.00 (0.00)
20	19.12 (0.12)	16.00 (0.00)	16.33 (0.58)	17.33 (0.58)
30	30.42 (0.12)	24.00 (1.00)	25.67 (0.58)	27.00 (0.00)
40	43.37 (0.13)	35.00 (1.00)	34.33 (0.58)	35.33 (0.58)
50	56.99 (0.18)	43.00 (1.73)	45.00 (2.00)	44.67 (2.00)
60	73.40 (0.20)	56.67 (1.53)	56.33 (0.58)	58.33 (1.16)
70	92.32 (0.34)	68.00 (1.73)	66.00 (0.00)	67.00 (3.00)
80	107.41 (0.16)	78.67 (2.31)	75.00 (2.65)	79.33 (1.53)
90	132.77 (0.26)	86.33 (2.52)	89.00 (1.73)	89.67 (1.53)
100	156.13 (2.16)	104.00 (2.65)	106.67 (1.15)	102.00 (1.00)

Table 1. Runtime of different simulators in s for the Euglena model with two light sources with intensity rates $\mathtt{I}_1 = 5.0, \mathtt{I}_2 = 15.0$, 100 Euglenas on each depth level ($\mathtt{n} = 100$), an opacity $\sigma = 0.2$, a rate for upward motion $\mathtt{u} = 1.0$, and different numbers of depth levels ranging from 10 to 100, i.e. $\mathtt{m} \in \{9, 19, \ldots, 99\}$. Simulation runs were performed until simulation time $t = 100.0$: Sto = stochastic π-calculus, Attr = attributed π-calculus, SPiM = SPiM, James = JAMES II, Enum = model with enumerated depth levels, Comp = model with depth level as species parameter. Standard deviation in parentheses.

B Remaining Proofs

B.1 Subsection 3.7 (Type System)

Corollary 2 (Error freeness). *If $\mathcal{L}$ is an attribute language that is both type safe and normalizing (see $\mathcal{L}$ Propositions 6 and 7) then $\pi(\mathcal{L})$ is error free in that if $\Gamma \vdash P$ and $P \rightarrow^* Q$ then $\neg Q \xrightarrow[nd]{err} \bot$.*

Proof. Assuming that $\Gamma \vdash P$ and $P \rightarrow^n Q$ the proof proceeds by induction on n. The inductive step follows from Theorem 1; it thus remains to prove the initial case that is $P \equiv Q$. We proceed by contradiction, assuming that there exists some process P_0 such that $\Gamma \vdash P_0$ and $P_0 \xrightarrow[nd]{err} \bot$.

Again, a standard analysis of the structural congruence shows the following claim: $P_0 \equiv (\nu\tilde{x}{:}\tilde{\tau}) \prod_{i=1}^{n} P_i$ be a prenex normal form in which all bound variables are renamed apart, and such that all P_i are sums or defined processes. A derivation of $P_0 \xrightarrow[nd]{err} \bot$ necessarily involves one the error axioms given in Figure 14. We now show that neither of them is applicable by case analysis.

(E.COM) In that case, $\exists j,k.1 \leq j < k \leq n$, $P_j = \pi_1.P_1 + M_1$, $P_k = \pi_2.Q_2 + M_2$, $\pi_1 \Downarrow x[v_1]?\tilde{y}$, $\pi_2 \Downarrow x[v_2]!\tilde{v}$ and $|\tilde{y}| \neq |\tilde{v}|$. From $\Gamma \vdash P_0$, by the lemma 3(4), we have $\Gamma \vdash (\nu\tilde{x}{:}\tilde{\tau}) \prod_{i=1}^{n} P_i$ that derives from a series of applications of rules (T.NEW) and (T.PAR) and from statements $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash P_i$ for all $i \in \{1, \ldots, n\}$. In particular, $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \pi_1.Q_1 + M_1$, and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \pi_2.Q_2 + M_2$. By (T.REC), $\pi_1 = e_1[e_1']?\tilde{y}$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_1{:}[\tau_1] \Rightarrow \tilde{\sigma_1}$, $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_1'{:}\tau_1$ and $|\tilde{y}| = |\tilde{\sigma_1}|$. Similarly, $\pi_2 = e_2[e_2']!\tilde{e_2''}$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_2{:}[\tau_2 \to \tau_2'] \Rightarrow \tilde{\sigma_2}$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_2'{:}\tau_2$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \tilde{e_2''}{:}\tilde{\sigma_2}$. Because $e_1[e_1']?\tilde{y} \Downarrow x[v_1]?\tilde{y}$ and $e_2[e_2']!\tilde{e_2''} \Downarrow x[v_2]!\tilde{v}$ and, by Proposition 6, e_1 and e_2 have the same type, that is $\tau_1 = \tau_2 \to \tau_2'$ and $\tilde{\sigma_1} = \tilde{\sigma_2}$. Because, $\tilde{e_2''} \Downarrow \tilde{v}$, by Proposition 6, $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \tilde{v} : \tilde{\sigma_2}$, therefore $|\tilde{v}| = |\tilde{\sigma_2}| = |\tilde{\sigma_1}| = |\tilde{y}|$ which contradicts $|\tilde{y}| \neq |\tilde{v}|$.

(E.PREF) In this case, $\exists j.1 \leq j \leq n$, $P_j = \pi_1.P_1 + M_1$ and $\neg\exists \pi_1'.\pi_1 \Downarrow \pi_1'$. Similarly to the previous case one can show that $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \pi_1.P_1 + M_1$ which is either derived from rule (T.REC) or (T.SEND). Suppose the later (the case (T.REC) is similar), then $\pi_1 = e_1[e_2]!\tilde{e_3}$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_1 : [\tau_1 \to \tau_2] \Rightarrow \tilde{\sigma}$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash e_2 : \tau_1$ and $\Gamma, \tilde{x}{:}\tilde{\tau} \vdash \tilde{e_3} : \tilde{\sigma}$. By Propositions 6 and 7, each of e_1, e_2 and $\tilde{e_3}$ evaluate to some typable value and rule (SEND) is applicable which contradicts $\neg\exists \pi_1'.\pi_1 \Downarrow \pi_1'$.

(E.CONSTR) In this case, $\exists j,k.1 \leq j < k \leq n$, $P_j = \pi_1.P_1 + M_1$, $P_k = \pi_2.Q_2 + M_2$, $\pi_1 \Downarrow x[v_1]?\tilde{y}$, $\pi_2 \Downarrow x[v_2]!\tilde{v}$ and $\neg\exists v.v_1 v_2 \Downarrow v$. Similarly to the case (E.COM) one can show that v_1 has some type $\tau_1 \to \tau_2$ and v_2 has type τ_1. Thus, by rule (T.FUNAPP), $v_1 v_2$ is typable with type τ_2 and, by Proposition 7, evaluates to some value v which contradicts $\neg\exists v.v_1 v_2 \Downarrow v$.

We conclude that none of the error axioms is applicable and thus $P_0 \xrightarrow[nd]{err} \bot$ does not hold. □

B.2 Subsection 5.1 (Encoding the Pi-Calculus with Priorities)

Theorem 2. *The encoding of the π-calculus with priorities $(R, <)$ into the attributed π-calculus $\pi(\lambda(R)_<)$ is correct in that for all processes P, P' with priorities and attributed processes Q:*

1. *$P \to P'$ then $[\![P]\!] \to [\![P']\!]$.*
2. *$[\![P]\!] \to Q$ then there exists a process $\hat{Q}$ of the π-calculus with priorities such that $[\![\hat{Q}]\!] \equiv Q$ and $P \to \hat{Q}$.*

Proof. First, we need to show that the encoding is invariant under substitutions.

Claim. $[\![P[\tilde{v}/\tilde{y}]]\!] = [\![P]\!][\tilde{v}/\tilde{y}]$ where $\tilde{v}$ is a tuple of values.

The proof is by induction on the structure of P. Second, we claim that the translation preserves and reflects structural congruence.

Claim. $P \equiv Q \Leftrightarrow [\![P]\!] \equiv [\![Q]\!]$.

The proof is by structural induction on derivations respectively of $P \equiv Q$ and $[\![P]\!] \equiv [\![Q]\!]$. We have to inspect all axioms of the structural congruence and all structural rules. We omit the details. Third we have to prove that errors are preserved and reflected by the translation.

Claim. $P \xrightarrow[nd]{err} \bot \Leftrightarrow [\![P]\!] \xrightarrow[nd]{err} \bot$.

(E.COM) This case is obvious, since this both calculi provide analogous rules. We omit the details.

(E.PREF) This rule exists only in the attributed π-calculus. Suppose that $[\![P]\!] \xrightarrow[nd]{err} \bot$ is inferred by (E.PREF):

$$\frac{\neg\exists\pi'.\pi \Downarrow \pi'}{[\![P]\!] = \pi.Q + M \xrightarrow[nd]{err} \bot}$$

Since sums can only be obtained by translating sums, P must match $\hat{\pi}.\hat{Q}+\hat{M}$ for some $\hat{Q}$ and $\hat{M}$. Here, $\hat{\pi}$ must be a prefix of the π-calculus with priorities, it follows that $\pi = [\![\hat{\pi}]\!]$ converges to itself, in contradiction to the hypothesis of the rule.

(E.CONSTR) This rules exists only in the attributed π-calculus. So suppose that (E.CONSTR) proves $[\![P]\!] \xrightarrow[nd]{err} \bot$, so it is applied as follows:

$$\frac{\pi_1 \Downarrow x[v_1]?\tilde{y} \quad \pi_2 \Downarrow x[v_2]!\tilde{v} \quad \neg\exists v.v_1v_2 \Downarrow v}{[\![P]\!] = \pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{err} \bot}$$

By inspection of the translation and the first two premises of the rule, we see that P must have the form $x?\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid x{:}r!\tilde{z}.\hat{P}_2 + \hat{M}_2$. Thus, $[\![P]\!] = x[\lambda z.z]?\tilde{y}.P_1+M_1 \mid x[r]!\tilde{z}.P_2+M_2$. This contradicts the third premise, however, since $v_1v_2 = (\lambda z.z)\ r \Downarrow r$ by rule (FUN) and since $r \in R$.

The treatment of structural rules is as before.

Fourth, we generalize the theorem.

Claim. For any relations $\rho \in \{\Downarrow, \xrightarrow[nd]{app}, \xrightarrow[nd]{r}, \rightarrow \mid r \in R\}$, and all processes $P', \hat{P}$ of the π-calculus with priorities and attributed processes P, Q:

1. if $P\ \rho\ P'$ then $[\![P]\!]\ \rho\ [\![P']\!]$.
2. if $[\![\hat{P}]\!] \equiv P$ and $P\ \rho\ Q$ then there exists an attributed process $\hat{Q}$ such that $[\![\hat{Q}]\!] \equiv Q$ and $\hat{P}\ \rho\ \hat{Q}$.

We proof the claim for all above relations ρ in the order in which they are given. The proof of point 1. is by structural induction on derivations of $P\ \rho\ P'$. We have to consider all rules of the operational semantics of the π-calculus with priorities.

(COM) This rule yields $P \xrightarrow[nd]{r} P'$ as follows:

$$\frac{|\tilde{y}| = |\tilde{z}|}{P = x?\tilde{y}.P_1 + M_1 \mid x{:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{z}/\tilde{y}] \mid P_2 = P'}$$

Thus $[\![P]\!] = x[\lambda y.y]?\tilde{y}.[\![P_1]\!] + [\![M_1]\!] \mid x[r]!\tilde{z}.[\![P_2]\!] + [\![M_2]\!]$, so that the (COM) rule of $\pi(\lambda(R)_<)$ applies while using (VAL) and (FUN):

$$\frac{x[\lambda y.y]?\tilde{y} \Downarrow x[\lambda y.y]?\tilde{y} \quad x[r]!\tilde{z} \Downarrow x[r]!\tilde{z} \quad (\lambda y.y)r \Downarrow r \in R \quad |\tilde{y}| = |\tilde{z}|}{[\![P]\!] \xrightarrow[nd]{r} [\![P_1]\!][\tilde{z}/\tilde{y}] \mid [\![P_2]\!]}$$

We can now apply our first claim on substitutions above to show that the reduction result is $[\![P_1[\tilde{z}/\tilde{y}]]\!] \mid [\![P_2]\!] = [\![P']\!]$ as required.

(APP) Suppose the following rule is applicable.

$$\frac{A(\tilde{x}) \triangleq P}{A(\tilde{v}) \xrightarrow[nd]{app} P[\tilde{v}/\tilde{x}]}$$

By the substitution assumption, we know that $[\![P[\tilde{v}/\tilde{x}]]\!] = [\![P]\!][\tilde{v}/\tilde{x}]$. The translation is defined, such that $[\![A(\tilde{x}) \triangleq P]\!] = A(\tilde{x}) \triangleq [\![P]\!]$. Thus, the following rule applies

$$\frac{A(\tilde{x}) \triangleq [\![P]\!]}{A(\tilde{v}) \xrightarrow[nd]{app} [\![P]\!][\tilde{v}/\tilde{x}]}$$

(PAR) We assume that the following rule is applicable.

$$\frac{P_1 \xrightarrow[nd]{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow[nd]{\alpha} P_1' \mid P_2}$$

By induction hypothesis, we have that $[\![P_1]\!] \xrightarrow[nd]{\alpha} [\![P_1']\!]$. Since the translation is compositional, the following rule is applicable:

$$\frac{[\![P_1]\!] \xrightarrow[nd]{\alpha} [\![P_1']\!]}{[\![P_1 \mid P_2]\!] \xrightarrow[nd]{\alpha} [\![P_1' \mid P_2]\!]}$$

(NEW) We assume that the following rule is applicable.

$$\frac{P \xrightarrow[nd]{\alpha} P'}{(\nu x)P \xrightarrow[nd]{\alpha} (\nu x)P'}$$

By induction hypothesis, we have that $[\![P]\!] \xrightarrow[nd]{\alpha} [\![P']\!]$. Since the translation is compositional, the following rule is applicable:

$$\frac{[\![P_1]\!] \xrightarrow[nd]{\alpha} [\![P_1']\!]}{[\![(\nu x)P]\!] \xrightarrow[nd]{\alpha} [\![(\nu x)P']\!]}$$

(STRUCT) We assume the following rule is applicable.

$$\frac{P \equiv P_1 \quad P_1 \xrightarrow[nd]{\alpha} P_2 \quad P_2 \equiv Q}{P \xrightarrow[nd]{\alpha} Q}$$

By the claim on the preservation of structural congruence, we have $[\![P]\!] \equiv [\![P_1]\!]$ and $[\![P_2]\!] \equiv [\![Q]\!]$. By induction hypothesis $[\![P_1]\!] \xrightarrow[nd]{\alpha} [\![P_2]\!]$. Thus, the following rule is applicable:

$$\frac{[\![P]\!] \equiv [\![P_1]\!] \quad [\![P_1]\!] \xrightarrow[nd]{\alpha} [\![P_2]\!] \quad [\![P_2]\!] \equiv [\![Q]\!]}{[\![P]\!] \xrightarrow[nd]{\alpha} [\![Q]\!]}$$

(CONV) Suppose that the following rule is applicable.

$$\frac{P \xrightarrow[nd]{app}{}^{*} P' \quad P' \equiv (\nu\tilde{x}) \prod_{i=1}^{n} M_i \quad \neg P' \xrightarrow[nd]{err} \bot}{P \Downarrow P'}$$

Since translation preserves structural congruence and application steps, we know that $[\![P]\!] \xrightarrow[nd]{app}{}^{*} [\![P']\!]$. Since it prevents errors, we have $\neg[\![P']\!] \xrightarrow[nd]{err} \bot$. Thus, the following rule is applicable:

$$\frac{[\![P]\!] \xrightarrow[nd]{app}{}^{*} [\![P']\!] \quad [\![P']\!] \equiv (\nu\tilde{x}) \prod_{i=1}^{n} [\![M_i]\!] \quad \neg[\![P']\!] \xrightarrow[nd]{err} \bot}{[\![P]\!] \Downarrow [\![P']\!]}$$

(PRIOR) Suppose the following rule is applicable.

$$\frac{P \Downarrow P' \quad P' \xrightarrow[nd]{r} Q \quad \neg\exists r_1 \in R.\exists Q_1.\ r < r_1 \wedge P' \xrightarrow[nd]{r_1} Q_1}{P \rightarrow Q}$$

Since we have treated relations $\Downarrow$ and $\xrightarrow[nd]{r}$ before, we have already shown that $P \Downarrow P'$ yields $[\![P]\!] \Downarrow [\![P']\!]$ and that $P' \xrightarrow[nd]{r} Q$ implies $[\![P']\!] \xrightarrow[nd]{r} [\![Q]\!]$. We can show by contradiction that $\neg\exists r_1 \in R.\exists Q_1.r < r_1 \wedge P \xrightarrow[nd]{r_1} Q_1$ then $\neg\exists r_2.\exists Q_2.r < r_2 \wedge [\![P]\!] \xrightarrow[nd]{r_2} Q_2$. Assume that $\neg\exists r_1 \in R.\exists Q_1.r < r_1 \wedge P \xrightarrow[nd]{r_1} Q_1$, but $\exists r_2.\exists Q_2.r < r_2 \wedge [\![P]\!] \xrightarrow[nd]{r_2} Q_2$. $[\![P]\!] \xrightarrow[nd]{r_2} Q'$ only if rule (COM) applies to $[\![P]\!]$, which is true only if $[\![P]\!] \equiv (\nu\tilde{x})(\ldots \mid x[r_2]!\tilde{y}.P_1 + M_1 \mid x[\lambda y.y]?\tilde{z}.P_2 + M_2 \mid \ldots)$. By the definition of the translation, this is fulfilled only if $P \equiv (\nu\tilde{x})(\ldots \mid x{:}r_2!\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid x?\tilde{z}.\hat{P}_2 + \hat{M}_2 \mid \ldots)$. Thus, $P \xrightarrow[nd]{r_2} Q$ exists, which contradicts with our assumption. Thus, the following rule is applicable

$$\frac{[\![P]\!] \Downarrow [\![P']\!] \quad [\![P']\!] \xrightarrow[nd]{r} [\![Q]\!] \quad \neg\exists r_1 \in R.\exists Q_1.\ r < r_1 \wedge [\![P']\!] \xrightarrow[nd]{r_1} Q_1}{[\![P]\!] \rightarrow [\![Q]\!]}$$

The proof of point 2. is by structural induction on derivations of $P \rho Q$, under the assumption that $[\![\hat{P}]\!] \equiv P$. We have to consider all rules of the non-deterministic operational semantics of $\pi(\lambda(R)_<)$ and all rules defining the structural congruence, but skip the latter.

(COM) By assumption, we have $[\![\hat{P}]\!] \equiv P$ and $P \xrightarrow[nd]{r} Q$ by applying the following rule:

$$\frac{\pi_1 \Downarrow x[v_1]?\tilde{y} \quad \pi_2 \Downarrow x[v_2]!\tilde{v} \quad v_1 v_2 \Downarrow r \in R \quad |\tilde{v}| = |\tilde{y}|}{P = \pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{v}/\tilde{y}] \mid P_2 = Q}$$

Inspecting the translation reveals that $\hat{P} \equiv \hat{P}'$ for some process $\hat{P}' = x?\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid x{:}r'!\tilde{v}.\hat{P}_2 + \hat{M}_2$ where $\pi_1 = x[\lambda z.z]?\tilde{y}$, $\pi_2 = x[r']!\tilde{v}$, $[\![\hat{P}_1]\!] \equiv P_1$ and $[\![\hat{P}_2]\!] \equiv P_2$. The prefix equalities yield $v_1 = \lambda z.z$ and $v_2 = r'$. We can deduce $r = r'$ from $v_1 v_2 = (\lambda z.z)r' \Downarrow r'$ by rules (VAL) and (FUN). We define $\hat{Q} = \hat{P}_1[\tilde{v}/\tilde{y}] \mid \hat{P}_2$ so that $[\![\hat{Q}]\!] = Q$ by the substitution claim. Furthermore, rules (COM) and (STRUCT) apply as follows:

$$\frac{\dfrac{|\tilde{y}| = |\tilde{v}|}{\hat{P} \equiv \hat{P}' = x?\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid x{:}r!\tilde{v}.\hat{P}_2 + \hat{M}_2 \xrightarrow[nd]{r} \hat{P}_1[\tilde{v}/\tilde{y}] \mid \hat{P}_2 = \hat{Q} \ \equiv \hat{Q}}}{\hat{P} \xrightarrow[nd]{r} \hat{Q}}$$

(APP) We assume $[\![\hat{P}]\!] \equiv P$ and that $P \xrightarrow[nd]{app} Q$ is inferred as follows:

$$\frac{[\![A(\tilde{x}) \triangleq P_1]\!]}{P = A(\tilde{v}) \xrightarrow[nd]{app} [\![P_1]\!][\tilde{v}/\tilde{x}] = Q}$$

Since $[\![\hat{P}]\!] \equiv A(\tilde{v})$, the translation yields that $\hat{P} = A(\tilde{v})$. The substitution claim shows that $[\![P_1]\!][\tilde{v}/\tilde{x}] = [\![P_1[\tilde{v}/\tilde{x}]]\!]$. We define $\hat{Q} = P_1[\tilde{v}/\tilde{x}]$ so that $[\![\hat{Q}]\!] = Q$. Furthermore, rule (APP) applies as follows:

$$\frac{A(\tilde{x}) \triangleq P_1}{\hat{P} = A(\tilde{v}) \xrightarrow[nd]{app} P_1[\tilde{v}/\tilde{x}] = \hat{Q}}$$

(PAR) We assume $[\![\hat{P}]\!] \equiv P$ and that $P \xrightarrow[nd]{\alpha} Q$ is obtained as follows:

$$\frac{P_1 \xrightarrow[nd]{\alpha} Q_1}{P = P_1 \mid P_2 \xrightarrow[nd]{\alpha} Q_1 \mid P_2 = Q}$$

Since the translation is compositional, assumption $[\![\hat{P}]\!] \equiv P$ implies the existence of two processes $\hat{P}_1$ and $\hat{P}_2$ such that $\hat{P} \equiv \hat{P}_1 \mid \hat{P}_2$ and $[\![\hat{P}_1]\!] \equiv P_1$ and $[\![\hat{P}_2]\!] \equiv P_2$. The induction hypothesis applied to $P_1 \xrightarrow[nd]{\alpha} Q_1$ shows the existence of a process $\hat{Q}_1$ such that $\hat{P}_1 \xrightarrow[nd]{\alpha} \hat{Q}_1$ and $[\![\hat{Q}_1]\!] \equiv Q_1$. We define

$\hat{Q} = \hat{Q}_1 \mid \hat{P}_2$, so that $[\![\hat{Q}]\!] = [\![\hat{Q}_1]\!] \mid [\![\hat{P}_2]\!] \equiv Q_1 \mid P_2 = Q$. Furthermore, we can infer $\hat{P} \xrightarrow[nd]{\alpha} \hat{Q}$ as follows by rules (PAR) and (STRUCT):

$$\frac{\hat{P} \equiv \hat{P}_1 \mid \hat{P}_2 \quad \dfrac{\hat{P}_1 \xrightarrow[nd]{\alpha} \hat{Q}_1}{\hat{P}_1 \mid \hat{P}_2 \xrightarrow[nd]{\alpha} \hat{Q}_1 \mid \hat{P}_2} \quad \hat{Q}_1 \mid \hat{P}_2 \equiv \hat{Q}}{\hat{P} \xrightarrow[nd]{\alpha} \hat{Q}}$$

(NEW) We assume $[\![\hat{P}]\!] \equiv P$ and that $P \xrightarrow[nd]{\alpha} Q$ is obtained as follows:

$$\frac{P_1 \xrightarrow[nd]{\alpha} Q_1}{P = (\nu x)P_1 \xrightarrow[nd]{\alpha} (\nu x)Q_1 = Q}$$

By the definition of the translation, we know that there exists a process $\hat{P}_1$ such that $\hat{P} \equiv (\nu x)\hat{P}_1$ and $P_1 \equiv [\![\hat{P}_1]\!]$. By induction hypothesis, there exists a process $\hat{Q}_1$ with $\hat{P}_1 \xrightarrow[nd]{\alpha} \hat{Q}_1$ and $Q_1 \equiv [\![\hat{Q}_1]\!]$. We define $\hat{Q}$ by $\hat{Q} = (\nu x)\hat{Q}_1$. Hence $[\![\hat{Q}]\!] \equiv Q$ by definition of the translation. Furthermore, we can infer $\hat{P} \xrightarrow[nd]{\alpha} \hat{Q}$ as follows:

$$\frac{\hat{P} \equiv (\nu x)\hat{P}_1 \quad \dfrac{\hat{P}_1 \xrightarrow[nd]{\alpha} \hat{Q}_1}{(\nu x)\hat{P}_1 \xrightarrow[nd]{\alpha} (\nu x)\hat{Q}_1} \quad (\nu x)\hat{Q}_1 \equiv \hat{Q}}{\hat{P} \xrightarrow[nd]{\alpha} \hat{Q}}$$

(STRUCT) We assume $[\![\hat{P}]\!] \equiv P$ and that $P \xrightarrow[nd]{\alpha} Q$ is inferred as follows:

$$\frac{P \equiv P_1 \quad P_1 \xrightarrow[nd]{\alpha} P_2 \quad P_2 \equiv Q}{P \xrightarrow[nd]{\alpha} Q}$$

Since every congruence relation is transitive, we get $[\![\hat{P}]\!] \equiv P_1$. The induction hypothesis applied to $P_1 \xrightarrow[nd]{\alpha} P_2$ thus proves the existence of a process $\hat{P}_2$ such that $\hat{P} \xrightarrow[nd]{\alpha} \hat{P}_2$ and $[\![\hat{P}_2]\!] \equiv P_2$. Transitivity of structural congruence yields $[\![\hat{P}_2]\!] \equiv Q$. We can thus define $\hat{Q} = \hat{P}_2$, so that $\hat{P} \xrightarrow[nd]{\alpha} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$.

(CONV) We first show the following claim by induction on derivation length $l \geq 0$:

Claim. For all $l \in \mathbb{N}_0$, if $[\![\hat{P}]\!] \equiv P$ and $P(\xrightarrow[nd]{app})^l Q_l$ then there exists $\hat{Q}_l$, such that $\hat{P}(\xrightarrow[nd]{app})^l \hat{Q}_l$ and $Q \equiv [\![\hat{Q}]\!]_l$.

Proof. For $l = 0$, the assumption $P(\xrightarrow[nd]{app})^0 Q_0$ is equivalent to $P \equiv Q_0$ by definition. Thus, $[\![\hat{P}]\!] \equiv Q_0$, so that we can define $\hat{Q}_0 = \hat{P}$ in order to obtain $[\![\hat{Q}_0]\!] \equiv Q_0$. For the induction step, let $[\![\hat{P}]\!] \equiv P$ such that $P(\xrightarrow[nd]{app})^l Q_l \xrightarrow[nd]{app} Q_{l+1}$. By induction hypothesis, there exists $\hat{Q}_l$ such that $\hat{P}(\xrightarrow[nd]{app})^l \hat{Q}_l$ and $Q_l \equiv [\![\hat{Q}_l]\!]$. Since we have finished the proof for relation $\xrightarrow[nd]{app}$ already, there exists $\hat{Q}_{l+1}$, such that $\hat{Q}_l \xrightarrow[nd]{app} \hat{Q}_{l+1}$ and $Q_{l+1} \equiv [\![\hat{Q}_{l+1}]\!]$. Clearly $\hat{P}(\xrightarrow[nd]{app})^{l+1} \hat{Q}_{l+1}$.

We next assume $P \equiv [\![\hat{P}]\!]$ and that $P \Downarrow Q$ is inferred by rule (CONV) as follows:

$$\frac{P \xrightarrow[nd]{app}{}^{*} Q \quad Q \equiv (\nu\tilde{x}) \prod_{i=1}^{n} M_i \quad \neg Q \xrightarrow[nd]{err} \bot}{P \Downarrow Q}$$

The claim above proves that there exists $\hat{Q}$ such that $\hat{P} \xrightarrow[nd]{app}{}^{*} \hat{Q}$ and $[\![\hat{Q}]\!] \equiv Q$. The definition of the translation yields that $\hat{Q} \equiv (\nu\tilde{x}) \prod_{i=1}^{n} \hat{M}_i$ for some guarded processes $\hat{M}_i$. Since the translation is error-reflecting, $\neg Q \xrightarrow[nd]{err} \bot$ yields $\neg\hat{Q} \xrightarrow[nd]{err} \bot$. We can thus infer $\hat{P} \Downarrow \hat{Q}$ as follows:

$$\frac{\hat{P} \xrightarrow[nd]{app}{}^{*} \hat{Q} \quad \hat{Q} \equiv (\nu\tilde{x}) \prod_{i=1}^{n} \hat{M}_i \quad \neg\hat{Q} \xrightarrow[nd]{err} \bot}{\hat{P} \Downarrow \hat{Q}}$$

(PRIOR) We assume $P \equiv [\![\hat{P}]\!]$ and that $P \Downarrow Q$ is inferred as follows:

$$\frac{P \Downarrow P_1 \quad P_1 \xrightarrow[nd]{r} Q \quad \neg\exists r_1 \in R. \exists Q_1.\ r < r_1 \wedge P_1 \xrightarrow[nd]{r_1} Q_1}{[\![P]\!] \rightarrow Q}$$

Since we have already proved the result for convergence, we know that there exists a process $\hat{P}_1$, such that $\hat{P} \Downarrow \hat{P}_1$ and $P_1 \equiv [\![\hat{P}_1]\!]$. Hence, there exists $\hat{Q}$ such that $\hat{P}_1 \xrightarrow[nd]{r} \hat{Q}$ and $Q \equiv \hat{Q}$. We next show that $\neg\exists r_1 \in R. \exists \hat{Q}_1. r < r_1 \wedge \hat{P}_1 \xrightarrow[nd]{r_1} \hat{Q}_1$.The proof is by contradiction. Suppose that such an r_1 and $\hat{Q}_1$ exist. For the first part of this theorem we have shown that this implies $[\![\hat{P}_1]\!] \xrightarrow[nd]{r_1} [\![\hat{Q}_1]\!]$. Since $P_1 \equiv [\![\hat{P}_1]\!]$, we can choose a $Q_1 \equiv [\![\hat{Q}_1]\!]$, such that, by rule (STRUCT), we obtain $P_1 \xrightarrow[nd]{r_1} Q_1$ in contradiction to the third hypothesis. □

B.3 Subsection 5.2 (Encoding π@ for Dynamic Compartments)

Lemma 5. *For all constants of variables $v_1, \ldots, v_n, v'_1, \ldots, v'_n$ it is true that:*

1. $eq_n(v_1, \ldots, v_n, v_1, \ldots, v_n) \Downarrow$ ***true***.

2. $eq_n(v_1, \ldots, v_n, v'_1, \ldots, v'_n) \Downarrow$ `false` *if* $v_i \neq v'_i$ *for some* $1 \leq i \leq n$.

The proof is straightforward by induction on n. It relies on the definition of conditionals and equality in the big-step evaluator in Figs. 11 and 12.

Proof. (1) The proof is by induction on n. For $n = 0$, we obtain $eq_0() \Downarrow$ `true` by definition. For the induction step, we have that:

$$\begin{array}{l} eq_{n+1}(v_1, \ldots, v_{n+1}, v_1, \ldots, v_{n+1}) =_{df} \\ \quad \texttt{if } v_{n+1}{=}v_{n+1} \texttt{ then } eq_n(v_1, \ldots, v_n, v_1 \ldots, v_n) \texttt{ else false}. \end{array}$$

By rules (EQ$_1$) and (COND$_1$) apply, we obtain $eq_n(v_1, \ldots, v_n, v_1 \ldots, v_n)$. By induction hypothesis:

$$eq_n(v_1, \ldots, v_n, v_1, \ldots, v_n) \Downarrow \texttt{true}$$

(2) The proof is by induction. The case $n = 0$ is trivial, since the hypothesis of the implication is always wrong. For the induction step from n to $n + 1$, we assume that $1 \leq i \leq n + 1$ such that $v_i \neq v'_i$ and:

$$\begin{array}{l} eq_{n+1}(v_1, \ldots, v_{n+1}, v'_1, \ldots, v_{n+1}) =_{df} \\ \quad \texttt{if } v_{n+1}{=}v'_{n+1} \texttt{ then } eq_n(v_1, \ldots, v_n, v'_1 \ldots, v'_n) \texttt{ else false}. \end{array}$$

Suppose $1 \leq i \leq n$ and $v_{n+1} = v'_{n+1}$. Then, by rules (EQ$_1$) and (COND$_1$), we obtain $eq_n(v_1, \ldots, v_n, v'_1 \ldots, v'_n)$, which evaluates to `false` by induction hypothesis. If $i = n + 1$, then we know by rules (EQ$_2$) and (COND$_2$):

$$\texttt{if } v_{n+1}{=}v'_{n+1} \texttt{ then } eq_n(v_1, \ldots, v_n, v'_1 \ldots, v'_n) \texttt{ else false} \Downarrow \texttt{false}$$

□

Theorem 4 (Operational correspondence).

(a) if $P \to Q$ *w.r.t.* $\mathcal{D}$*, then* $[\![P]\!] \to [\![Q]\!]$ *w.r.t.* $[\![\mathcal{D}]\!]$
(b) if $[\![P]\!] \to Q$ *w.r.t* $[\![\mathcal{D}]\!]$*, then* $\exists \hat{Q}$ *s.t.* $Q \equiv [\![\hat{Q}]\!]$*, and* $P \to \hat{Q}$ *w.r.t* $\mathcal{D}$.

Proof. With Theorem 2, we proved the encoding of the π-calculus with priorities in $\pi(\lambda(R)_<)$ correct. The non-deterministic semantics of $\pi@$ and the π-calculus with priorities are the same, with only one exception: the communication rule. Therefore, in the following proof, we only consider the communication rule.

(a) We define κ_o and κ_i:

$$\begin{array}{l} \kappa_o = \lambda y_2 \ldots \lambda y_n \lambda r'. \texttt{ if } eq_n(x_2, \ldots, x_n, r, y_2, \ldots, y_n, r') \texttt{ then } r \\ \kappa_i = \lambda e.e x_2 \ldots x_n r \end{array}$$

Rule (COM) yields $P \xrightarrow[nd]{r} P'$ as follows:

$$\frac{|\tilde{y}| = |\tilde{z}|}{P = \tilde{x}{:}r?\tilde{y}.P_1 + M_1 \mid \tilde{x}{:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{v}/\tilde{y}] \mid P_2 = P'}$$

where $\tilde{x} = (x_1, \ldots, x_n)$. Thus, $[\![P]\!] = x_1[\kappa_i]?\tilde{y}.[\![P_1]\!] + [\![M_1]\!] \mid x_1[\kappa_o]!\tilde{z}.[\![P_2]\!] + [\![M_2]\!]$. By Lemma 5, we know that $\kappa_i\kappa_o \Downarrow r$, such that the (COM) rule of $\pi(R, <)$ applies:

$$\frac{x_1[\kappa_i]?\tilde{y} \Downarrow x_1[v_1]?\tilde{y} \quad x_1[\kappa_o]!\tilde{z} \Downarrow x_1[v_2]!\tilde{z} \quad v_1v_2 \Downarrow r \in R \quad |\tilde{y}| = |\tilde{z}|}{[\![P]\!] \xrightarrow[nd]{r} [\![P_1]\!][\tilde{v}/\tilde{y}] \mid [\![P_2]\!]}$$

By the claim on substitutions, we can show that the reduction result is $[\![P_1[\tilde{v}/\tilde{y}]]\!] \mid [\![P_2]\!] = [\![P']\!]$ as required.

(b) We define κ_o and κ_i:

$$\begin{array}{l} \kappa_o = \lambda y_2 \ldots \lambda y_n \lambda r'.\ \texttt{if}\ eq_n(x'_2, \ldots, x'_n, \hat{r}', y_2, \ldots, y_n, r')\ \texttt{then}\ r \\ \kappa_i = \lambda e.ex_2 \ldots x_n\hat{r} \end{array}$$

By assumption, we have $[\![\hat{P}]\!] \equiv P$ and $P \xrightarrow[nd]{r} Q$ by applying the following rule:

$$\frac{\pi_1 \Downarrow x_1[v_1]?\tilde{y} \quad \pi_2 \Downarrow x_1[v_2]!\tilde{v} \quad v_1v_2 \Downarrow r \in R \quad |\tilde{v}| = |\tilde{y}|}{P = \pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{v}/\tilde{y}] \mid P_2 = Q}$$

Inspecting the translation reveals that a $\hat{P}'$ exists, s.t. $\hat{P} \equiv \hat{P}'$ and $\hat{P}' = (x_1, x_2 \ldots, x_n){:}\hat{r}?\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid (x_1, x'_2, \ldots, x'_n){:}\hat{r}'!\tilde{z}.\hat{P}_2 + \hat{M}_2$, with $[\![\hat{P}_1]\!] \equiv P_1$, $[\![\hat{P}_2]\!] \equiv P_2$, $\pi_1 = x_1[\kappa_i]?\tilde{y}$, and $\pi_2 = x_1[\kappa_o]!\tilde{z}$. The prefix equalities yield $v_1 = \kappa_i$ and $v_2 = \kappa_o$. By Lemma 5, we know that $x'_2 = x_2 \ldots, x'_n = x_n$, $\hat{r}' = \hat{r}$, and $r = \hat{r}$, since $P \xrightarrow[nd]{r} Q$ by the rule above. We define $\hat{Q} = \hat{P}_1[\tilde{v}/\tilde{y}] \mid \hat{P}_2$ so that $[\![\hat{Q}]\!] = Q$ by the substitution claim. Furthermore, rules (COM) and (STRUCT) apply as follows:

$$\frac{\dfrac{|\tilde{y}| = |\tilde{z}|}{\hat{P} \equiv \hat{P}' = \tilde{x}{:}r?\tilde{y}.\hat{P}_1 + \hat{M}_1 \mid \tilde{x}{:}r!\tilde{z}.\hat{P}_2 + \hat{M}_2 \xrightarrow[nd]{r} \hat{P}_1[\tilde{z}/\tilde{y}] \mid \hat{P}_2 = \hat{Q} \equiv \hat{Q}}}{\hat{P} \xrightarrow[nd]{r} \hat{Q}}$$

□