# The Attributed Pi Calculus

Mathias John, Cédric Lhoussaine, Joachim Niehren, Adelinde Uhrmacher

# 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, LIFL, CNRS UMR8022
[3] INRIA, Lille, Mostrare project
[4] BioComputing project, LIFL and IRI, Lille

**Abstract.** We present the attributed pi-calculus for modeling concurrent systems with interaction constraints depending on values of attributed of processes. The lambda calculus serves as a constraint language underlying the pi-calculus. We present a non-deterministic and a stochastic semantics for the attributed pi-calculus. We show how to encode the pi-calculus with priorities and the stochastic pi-calculus, and prove the correctness of both translations. Polyadic synchronization as in $\pi$@ or SPiCO are expressible by equality constraints.
We illustrate the usefulness of the attributed pi-calculus for modeling and simulation in systems biology at two examples: Euglena's spatial movement in phototaxis, and cooperative protein binding in gene regulation of bacteriophage lambda. A stochastic simulation algorithm for the attributed pi-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

A plethora of formal concurrent modeling languages has been proposed for systems biology since the seminal work of Regev and Shapiro [1, 2] on the $\pi$-calculus. These languages subscribe to two main paradigms: in the object-centered paradigm, interaction capacities are attached to concurrent actors, as for instance in the $\pi$-calculus [3–7]. Rule-based languages focus on chemical reactions and pathways being composed thereof [8–10]. Deterministic models in terms of differential equations can be obtained by averaging over possible behaviors of non-deterministic models in concurrent languages [11, 12, 8].

In this paper, we propose a uniform framework to express interaction constraints for object-centered approaches based on the $\pi$-calculus. It generalizes on numerous of instance in the literature, such as the $\pi$-calculus with priorities [13], which assigns priority values to potential interactions and licences only interactions with highest priority, or the stochastic $\pi$-calculus [14, 15, 7, 16], which assigns stochastic rates to interactions, and constrains the stochastic simulator by these rates. Another example is polyadic synchronization [17, 13], which restricts interactions by imposing equality on tuples of channels.

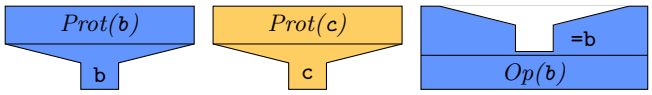


**Fig. 1.** Only the protein of sort b is permitted to bind to the operator of sort b, but not the protein of sort c. Equality of sorts is tested by the operator, once it sees the sort of the protein, by applying the test function $\lambda x.x{=}\mathtt{b}$.

We introduce the attributed $\pi$-calculus as an extension of the $\pi$-calculus by attributed processes and interaction constraints, that may depend on attribute values of processes. Attribute values of various types are useful in order to define diverse 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 $comp \in \{\mathtt{nucleus}, \ldots\}$ in which it is located.

Whether two processes of the attributed $\pi$-calculus are allowed to interact may depend on constraints on their attribute values. 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 $\pi$-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(b)* and *Op(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 $\pi$-calculus where priorities may depend on attribute values.

We propose to use a call-by-value lambda calculus $\mathcal{L}$ as a language, in which to define attribute values and constraints. We leave the choice of constants of $\mathcal{L}$

parametric, in order to allow for the many reasonable choices of data values, such as positive real numbers for stochastic rates or ordered sets for priorities. We then define the $\pi$-calculus $\pi(\mathcal{L})$, in which $\mathcal{L}$ is a parameter, similarly to constraint logic programming CLP($\mathcal{L}$) [18] or concurrent constraint programming cc($\mathcal{L}$) [19]. We present two semantics for $\pi(\mathcal{L})$, a prioritized non-deterministic semantics and a stochastic semantics (under the assumptions that successful constraints evaluate to stochastic rates in $\mathbb{R}^+$). The stochastic semantics is a proper refinement of the non-deterministic semantics, in that it permits the same reduction steps (Proposition 7).

We show how to translate the $\pi$-calculus with priorities in an ordered set $(R, <)$ into the attributed $\pi$-calculus $\pi(\lambda(R), <)$ with priorities as constants, and prove this encoding correct (Theorem 1). We also show how to encode the $\pi$-calculus with priorities and polyadic synchronization $\pi@$ into $\pi(\lambda(R, =), <)$, by constraining communication actions by equality constraints on tuples of channels, and lift our correctness result (Theorem 2). This calculus was introduced by Versari [13, 20] as a uniform framework in which to express spatial aspects, and in particular to encode BioAmbients [21] and Brane [22]. The same translation as for the $\pi$-calculus with priorities (but with different constants) can be used to encode of the stochastic $\pi$-calculus, where priorities are annotated to communication prefixes into the attributed $\pi$-calculus $\pi(\lambda(\mathbb{R}_+^\infty))$ with two levels of priorities, again with correctness proof (Theorem 4). We also present a type system for the attributed $\pi$-calculus, that extends on a type system for the simply typed lambda calculus, and prove type safety (Theorem 3). All encodings except that one for $\pi@$ preserve typings. These results confirm uniformity and expressiveness of our framework.

The article has 3 parts. We first recall existing pi calculi, second introduce the attributed $\pi$-calculus and relate it to the previous languages, and third discuss stochastic simulation and applications to systems biology.

The first part starts with the $\pi$-calculus with priorities (Section 2) and then turns it into the the stochastic $\pi$-calculus by instantiating the set of priorities by stochastic rates in $\mathbb{R}_+^\infty$ (Section 3). Apart from that, both calculi have the same syntax. We chose to annotate output prefixes by priorities or stochastic rates, rather than annotating channels in contrast to Biospi [7], SPiM [16], and SPiCO [15]. Note also, that our stochastic semantics is not restricted to processes in biochemical form. This is relevant, since it simplifies all technical results in contrast to most previous approaches (including the conference version of the present article). We also recall a type system for this language.

The second part of this article starts with Section 4. It first introduces the syntax of the attributed $\pi$-calculus and presents its non-deterministic operational semantics. We then encode the $\pi$-calculus with priorities and $\pi@$ and prove both encodings correct. In Section 3, we present the stochastic semantics of the attributed $\pi$-calculus, encode the stochastic $\pi$-calculus and prove the encoding correct. The last contribution of this section is a type system for the attributed $\pi$-calculus, which lifts the simple type system of the lambda calculus on process level. We prove type safety of processes, under the assumption that the attribute

language is type safe, and show that all encodings preserve types, with the exception of the encoding of $\pi@$, for which convincing type systems are difficult to find.

In the third part, we illustrate the usefulness of the attributed $\pi$-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. The second example regards cooperative binding, a phenomenon, which is often observed in gene regulatory systems [23, 24]. Afterwardwe present a simulation algorithm that is inferred directly from the stochastic semantics. To give an impression of its performance, run-time experiments are executed with three different simulators, i.e. a $\pi(\mathcal{L})$ simulator realized in the modeling and simulation framework JAMES II [25], a the stochastic $\pi$-calculus simulator also realized in JAMES II, and SPiM [6]. Variants of the Euglena model are the basis for this evaluation.

*Related work* The $\pi$-calculus with polyadic synchronization was first proposed in [17] but could be extended by more general data terms as in [26] without particular difficulties. An extended $\pi$-calculus with polyadic synchronization, data terms, and explicit substitutions motivated by security applications was proposed in [27].

Another instance of the idea of interaction constraints is present in BlenX [28, 4], where interactions are allowed only between prefixes sharing a common type annotation. There exists a stochastic variant of $\pi@$ called $s\pi@$ [29], which is a stochastic simulation algorithm that accounts for compartments with variable volumes. However, this algorithm remains ad hoc since it is not derived from a stochastic semantics of the calculus. Therefore, we refrain from a formal comparison here.

Further modeling approaches based on the stochastic $\pi$-calculus can be found in [30, 31]. All stochastic $\pi$ models can be translated into the attributed $\pi$-calculus. This requires to encode the versions of the stochastic $\pi$-calculus used there into the attributed $\pi$-calculus. We show in Section 5.3, that we can express the version of the stochastic $\pi$-calculus where stochastic rates are annotated to channels. Here, we rely on having pairs in the attributed $\pi$-calculus. This encoding can be lifted to an encoding of stochastic $\pi$-calculus with concurrent objects (SPiCO) [5], as used in the modeling approaches of [23, 24]. The only new point here is to encode the weak form of polyadic synchronization of SPiCO in a stochastic setting.

The closest approach to ours, is stochastic concurrent constraint programming (SCCP) [32]. There, one can indeed express attributed processes with stochastic rates depending on attribute values. There are 4 main differences. 1) All interactions in SCCP are by indirection though a global constraint store, 2) this store is imperative, 3) all constraints are first-order, and 4) chemical reactions with n reactants can be expressed in SCCP based on the imperative features. N-ary reactions are difficult to express in the attributed $\pi$-calculus. This can be solved in analogy to SCCP in the imperative $\pi$-calculus, a recent extension of the attributed $\pi$-calculus with a global imperative store [33].

Compared to the conference version of this paper, we added priorities to the non-deterministic version of the attributed $\pi$-calculus, and extended the encoding of $\pi@$ such that it accounts for priorities. The whole presentation has been changed, such that the attributed $\pi$-calculus becomes a proper generalization of the $\pi$-calculus with priorities and the stochastic $\pi$-calculus. Annotations have been moved from channels to communication prefixes, so that no global environments needed to be introduced. The changes have lead to an important simplification of the stochastic semantics, which enables our correctness proofs. We have performed an implementation of the attributed $\pi$-calculus. First practical experiments confirm decent performance and thus relevance in practice.

## 2 Pi-Calculus with Priorities

We recall the non-deterministic version of the $\pi$-calculus with priorities [13], concomitantly with a type system, that excludes arity mismatches during communication. Our presentation permits parametrized process definitions, that are very useful for modeling in systems biology, instead of a replication operator as in [13]. In contrast to previous presentations, we do not impose any syntactic restrictions such as biochemical forms [6, 34]. We require that definitions are always exhaustively applied in order to select a communication step of highest priority. The resulting operational semantics remains simple, and can be generalized properly in order to obtain the attributed $\pi$-calculus.

We annotate priorities to sender prefixes only, for the sake of simplicity. Alternatively, we could add them to both sender and receiver prefixes. The priority of a communication step could then be computed by taking the minimum, or else, one could constrain all communication steps to senders and receivers with equal priority [13].

### 2.1 Syntax

Let $\mathtt{Bool} = \{\mathtt{true}, \mathtt{false}\}$ be the set of Booleans, $\mathbb{N}$ the set of natural numbers starting from 1, $\mathbb{N}_0$ for $\mathbb{N} \cup \{0\}$, $\mathbb{R}_+$ the set of non-negative real numbers, and $\mathbb{R}_+^\infty = \mathbb{R}_+ \cup \{\infty\}$. Furthermore, we assume a partially ordered set $(R, <)$ of priorities.

We start from an infinite set *Vars* of channel names $x, y \in \mathit{Vars}$ and a infinite set of process names $A \in \mathit{Proc}$, each of which comes with a fixed arity 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 $\pi$-calculus with priorities is defined in Fig. 2. In addition to channel names $x \in \mathit{Vars}$ and priorities $r \in R$ there are 4 syntactic categories: prefixes $\pi$, processes $P$, sums $M$, and definitions $D$. A prefix is either a receiver

| | | | |
|---|---|---|---|
| Prefixes | $\pi$ | $::= x?\tilde{y}$ | 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 $\pi$-calculus with channels $x, \tilde{x}, \tilde{y}, \tilde{z} \in \mathit{Vars}$ and priorities $r \in R$.

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

**Fig. 3.** Free channel names.

$x?\tilde{y}$ 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 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 no variable is bound twice in $P$, if no bound variable of $P$ has a free occurrence in $P$, and if 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

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

**Fig. 4.** Axioms of structural congruence

of the neutral element of $\mathbf{0}$ wrt. parallel composition, and scope intrusion and extrusion for $\nu$-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 $M$ 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.

### 2.2 Operational Semantics

The operational semantics of the $\pi$-calculus with priorities is defined by a reduction relation $\rightarrow$ that is based on two binary relations $\xrightarrow[nd]{r}$ and $\xrightarrow[nd]{err}$ defined by a collection of axioms and closure rules.

Fig. 5 contains the axioms of the binary relations on processes $\xrightarrow[nd]{\alpha}$ where $\alpha \in R \cup \{app, err\}$. 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 final reduction relation $\rightarrow$ between processes. Communication and error steps are closed under structural congruence (STRUCT), and permitted below parallel composition (PAR) and new binders (NEW). Rule (PRIOR) states that only communication steps with highest available priority may be selected by final reduction 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 $P \xrightarrow[nd]{err} \perp$ block all communication steps on $P$. Finally note, that the reflexive transitive

**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 $\pi$-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** $(r \in R)$

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

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

closure $(\xrightarrow[nd]{app})^*$ used in rule (CONV) is defined 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 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:

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

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

$$\mathit{Ring}_3() \triangleq x_1{:}\mathit{low}!(z).\mathbf{0} \mid x_2{:}\mathit{low}!(z).\mathbf{0} \mid \mathit{Fwd}(x_1, x_2) \mid \mathit{Fwd}(x_2, x_3) \mid \mathit{Fwd}(x_3, x_1)$$

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.

$$\begin{aligned}\mathit{Ring}_3() &\to x_1{:}\mathit{low}!(z).\mathbf{0} \mid \mathit{Fwd}(x_1, x_2) \mid \mathit{Fwd}(x_2, x_3) \mid x_3{:}\mathit{high}!(z).\mathbf{0} \mid \mathit{Fwd}(x_3, x_1)\\ &\to x_1{:}\mathit{low}!(z).\mathbf{0} \mid x_1{:}\mathit{high}!(z).\mathbf{0} \mid \mathit{Fwd}(x_1, x_2) \mid \mathit{Fwd}(x_2, x_3) \mid \mathit{Fwd}(x_3, x_1)\\ &\to x_1{:}\mathit{low}!(z).\mathbf{0} \mid \mathit{Fwd}(x_1, x_2) \mid x_2{:}\mathit{high}!(z).\mathbf{0} \mid \mathit{Fwd}(x_2, x_3) \mid \mathit{Fwd}(x_3, x_1)\\ &\to \ldots\end{aligned}$$

**Lemma 1.** *The rewrite relation $\xrightarrow[nd]{app}$ is confluent on equivalence classes of processes modulo structural congruence. The normal forms of this rewrite system are equivalence classes of processes of the form $(\nu\tilde{x})\prod_{i=1}^{n} M_i$.*

*Proof.* A standard analysis of the structural congruence shows the following claim.

*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 [35], 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 [36]. 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 $\xrightarrow[nd]{app}$ may not always 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.

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

*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.

### 2.3 Type System

We present a type system 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* 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. We denote type environments by $\Gamma$.

*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 $\tau$. The second subprocess of $P$, requires type environments containing $x{:}ch(ch())$. Both conditions together are unsatisfiable.

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

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

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

**Fig. 7.** Type system for $\pi$-calculus with priorities.

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

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

Type checking rules for processes are given in Fig. 7. They infer judgments $\Gamma \vdash P$ that state the consistency of a process $P$ with a type environment $\Gamma$ as usual. Inconstencies may be raised by arity mismatches of receivers (T.REX), sender (R.SEND), process applications (T.APP), and process definitions (T.DEF). Furthermore, there are three structural rules, and a rule for the null process $\mathbf{0}$.

**Proposition 2 (Subject reduction).** *If $\Gamma \vdash P$ and $P \to Q$ then $\Gamma \vdash Q$.*

The proof works as usual. See the proof of Theorem 3 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 2; 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 prenix 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 serie 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 $|\tilde{y}| \neq |\tilde{z}|$. □

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

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

**Fig. 8.** Axioms of operational semantics of stochastic $\pi$-calculus.

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

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

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

**Fig. 9.** Rules of the stochastic semantics of the $\pi$-calculus with priorities.

## 3 Stochastic Pi-Calculus

We recall a variant of the stochastic $\pi$-calculus where stochastic rates are assigned to output prefixes, rather than to channel names as in [7, 16, 15], or to both input and output prefixes [14]. We obtain our variant by defining a stochastic semantics for the $\pi$-calculus with priorities, where stochastic rates in $\mathbb{R}_+^\infty$ are used as priorities. In contrast to most previous approaches, the syntax of processes remains without change.

As usual in the stochastic $\pi$-calculus, we assume two levels of priorities, the lower level for all numbers in $\mathbb{R}_+$ and the higher level for $\infty$. This means that communication steps with finite rates can be applied only if no communication step with infinite rate is available.

The stochastic semantics of a process $P$ in the stochastic $\pi$-calculus is a continuous time Markov chain (CTMC). The states of such CTMCs are classes of processes $[P]_\equiv$ modulo structural congruence. 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 probabiblity 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, that the transition is impossible, since some transition with infinite rate 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 pro-

portional to the number of possible interactions of its reactants in the solution (when assuming given 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 interaction 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. 8 defines communication steps labeled by positions $\ell \in \mathbb{N}^4$, where the interaction occurs $P \xrightarrow[\ell]{r} P'$. Given a prenex normal form $P = (\nu\tilde{x}) \prod_{i=1}^{n} \sum \pi_i^j.P_i^j$, a tuple $\ell = (i_i, j_1, i_2, j_2)$ define the pairs of communication prefixes $\pi_{i_1}^{j_1}.P_{i_1}^{j_1}$ and $\pi_{i_2}^{j_2}.P_{i_2}^{j_2}$. In order to count interaction positions $\ell$ for a congrence class $[P]_\equiv$, we have to fix a representative of of this class that is in prenex normal form. The is process $P_1$ in rule (COM$_\ell$). 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 positon $\ell$, where $r$ is the rate annotated to the sender.

The transitions of the CTMC are defined in Fig. 9. 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 $(r', \ell)$ of 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 $P \xrightarrow{\infty(r)} P'$ is $r = n/m$ where $n$ is the number of interactions with rate $\infty$ leading from $P$ to a processes congruent to $P'$, and $m$ the number of interactions with rate $\infty$ starting from $P$. Given these probabilities, immediate transitions can be eliminated in order to obtain a CTMC preserving the *probabilities of transitions* and *sojourn times* (see e.g. [5] for details).

For illustration, consider a system of two chemical reactions, $x : A \oplus B \xrightarrow{0.5} A \oplus C$ and the inverse $y : A \oplus C \xrightarrow{5} A \oplus B$ whose rate is 10-fold higher. In Fig. 10, we define species $A, B, C$ as processes in the stochastic $\pi$-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 $\pi$-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. 10.

Chemical rules:

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

$\pi$-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 $A^2 \mid B^2 \mid C^1$:

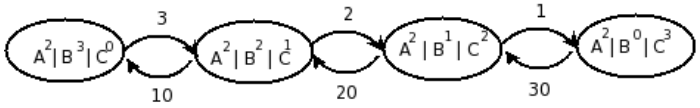


**Fig. 10.** Example of a CTMC generated by the stochastic $\pi$-calculus.

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

**Proposition 3.** *If the partial order $<$ on successful values of $\mathcal{L}$ satisfies $r < \infty$ and not $r < r'$ for all $r, r' \in \mathbb{R}_+$ then for all processes $P, P'$:*

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

*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 apply create a transitions with infinite propensity:

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

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

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

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

□

## 4 Attributed $\pi$-Calculus with Priorities

We introduce the attributed $\pi$-calculus $\pi(\mathcal{L})$, by extending the $\pi$-calculus with priorities with richer sets of values and expressions in same attribute language $\mathcal{L}$. We permit such expressions in generalized senders and receivers, in order to impose constraint on communication steps, which extend on priorities and stochastic rates.

The attribute language $\mathcal{L}$ will be a call-by-value $\lambda$-calculus, in which we keep the choice of constants parametric, in order to avoid reinventing independent calculi for the many useful choices in practice. We present two semantics for $\pi(\mathcal{L})$, a non-deterministic semantics with priorities, and a stochastic semantics. We thus preserve the analogy between the $\pi$-calculus with priorities and the stochastic $\pi$-calculus. Both semantics are defined independently of the concrete choice of the attribute language (except that the set of successful values must be $R = \mathbb{R}_+^\infty$ with 2 levels of priorities in the stochastic case).

### 4.1 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, $+$, $*$, $\geq$, `fst`, `repressor`), and from variables $x, y \in \mathit{Vars}$, which may play the roles of channel names in particular. Whenever ambiguity might arise, we write constants in courier font (such as `fst` or `repressor`) and variables in italics such as *bind*. As an example, consider the expression $x + y$ which defines a stochastic rate or priority in definition $A(x, y) \triangleq z[x + y]!().P$.

An attribute language with variables in *Vars* is a tuple $\mathcal{L} = (\mathit{Consts}, \Downarrow, R, <)$. It contains a set of constants $c \in \mathit{Consts}$, a big-step evaluator $\Downarrow$ for $\lambda$-expressions with these 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 $\lambda$-calculus. The set of expressions *Exprs* of $\mathcal{L}$ is defined as the set of all $\lambda$ expressions with constants in *Consts* and variables in *Vars*, and the set *Vals* of values of $\mathcal{L}$ as the set of all values of this $\lambda$-calculus.

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

$$(\textsc{val})\ \frac{v \in \mathit{Vals}}{v \Downarrow v} \qquad (\textsc{fun})\ \frac{e_1 \Downarrow \lambda x.e_1' \quad e_2 \Downarrow v' \quad e_1'[v'/x] \Downarrow v}{e_1 e_2 \Downarrow v}$$

$$(\textsc{pair})\ \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\langle e_1, e_2\rangle \Downarrow \langle v_1, v_2\rangle} \qquad (\textsc{select})\ \frac{e \Downarrow \langle v_1, v_2\rangle}{\texttt{fst}\ e \Downarrow v_1 \qquad \texttt{snd}\ e \Downarrow v_2}$$

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

**Fig. 11.** Rules of big-step evaluators of the basic attribute language $\lambda$: the call-by-value lambda calculus with pairs and conditionals.

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

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

**Fig. 12.** Additional rules for the big-step evaluator of attribute languages $\lambda(\mathbb{N}_0, +)$, $\lambda(\mathbb{N}_0, +, =)$ and $\lambda(=)$.

We use variables $x \in \mathit{Vars}$ as variables of expressions of the $\lambda$-calculus. As usual, such expressions provide abstractions $\lambda x.e$ and applications $e_1 e_2$ for specifying functions and their 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` $e_2$.

We assume that the third component is a set $R \subseteq \mathit{Vals}$, that defines successful values enabling communication steps, such as priorities or stochastic rates. The fourth component $<$ is a partial order on $R$, which fixes the priorities. The last component of $\mathcal{L}$ is a big-step evaluator $\Downarrow : D \to \mathit{Vals}$ for some subset of expressions $D \subseteq \mathit{Exprs}$. This is a black box algorithm that evaluates all expressions in $D$ to values. For all other expressions, either it raises an error or it does not terminate. Instead of $\Downarrow(e) = v$ we will write $e \Downarrow v$ and call $v$ the value of $e$.

We assume that all big-step evaluators satisfy the rules in Fig. 11. Rule (VAL) states that values evaluate to themselves. Rule (FUN) defines the usual meaning of call-by-value function application. It says that $e_1 e_2$ is evaluated by evaluating $e_1$ to some function $\lambda x.e_1'$ and $e_2$ to some value $v'$ and then returning the value obtained by evaluating $e_1'[v'/x]$. Rule (PAIR) requires that pairs are evaluated by evaluating both components (as usual in call-by-value languages). Rule (SELECT) states how to apply selectors to evaluated pairs. Rule ($\textsc{cond}_1$) and ($\textsc{cond}_2$) require that conditionals are evaluated as usual, such that only needed branches are evaluated.

For richer attribute language with further constants (such as $+$, $*$, or fixed point operators `fix`), we need to add further rules to the definition of the big-step evaluator. This is possible, since we kept the big-step evaluator abstract, too.

As examples, consider the attribute language $\lambda(\mathbb{N}_0, +)$ which provides natural numbers and 0, with addition $+$. We will assume that all nonzero positive numbers are successful, and that that all values are on the same priority level. Or, we consider the attribute language $\lambda(=)$, whcih provides an equality for constants and channels. For syntactical convenciance, we will freely write $e{=}e'$ instead of $((= e)\ e')$. Or else, we can use the union $\lambda(\mathbb{N}_0, +, =)$ of these two extensions. The additional rules for the big-step-evaluator of the attribute language is given in Fig. 12.

Further extensions of the attribute language, that cannot be obtained by adding constants only, such as tuples, lists, or case statements, might be useful in various applications. Even though we don't expect particular difficulties, we refrain from generalizing or extending the attribute language any further, for sake of simplicity.

Values of expressions may be undefined, i.e. there exist expressions $e$ such that there is no 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 4.6 we will present a type system for the attributed $\pi$-calculus, which prevents from type errors. If not adding any constants to the attribute language, then it excludes non-termination. For more general attribute languages with additional constants, however, the type systems may neither exclude all program errors (e.g. division by 0), nor ensure termination (e.g. fixed point operators).

### 4.2 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 $v \in \mathit{Vals}$ of $\mathcal{L}$.

The syntax of the attributed $\pi$-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 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 $e_1e_2 \Downarrow v$ for some successful value $v \in R$. This value then fixes the priority or stochastic rates of the communication step.

For illustration, we consider 4 instances of the attributed $\pi$-calculus with 4 different attribute languages. As a first example, we consider the calculus $\pi(\lambda)$

| | | | |
|---|---|---|---|
| Prefixes | $\pi$ | $::= e_1[e_2]?\tilde{x}$ | receiver |
| | | $\mid e_1[e_2]!\tilde{e}$ | sender |
| Sums | $M$ | $::= \pi.P$ | guarded process |
| | | $\mid M_1 + M_2$ | choice |
| Processes | $P$ | $::= M$ | sums |
| | | $\mid A(\tilde{e})$ | defined process |
| | | $\mid P_1 \mid P_2$ | parallel composition |
| | | $\mid (\nu x)P$ | channel creation |
| | | $\mid \mathbf{0}$ | 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 $e_1, e_2, \tilde{e} \in \mathit{Exprs}$.

with the basic attribute language with pairs and conditionals. The second example is $\pi(\lambda(=))$, where we add a single constant, for testing equality of channels or constants. The third example is $\pi(\lambda(\mathbb{N}_0, +))$ which provides for natural numbers with addition, and the fourth example is $\pi(\lambda(\mathbb{N}_0, +, =))$ where all of these constants are permitted.

As an example, consider the following definitions in $\pi(\lambda(\mathbb{N}_0, +))$, which encode schemes for chemical rules $z : A(x) \oplus B(y) \xrightarrow{x+y} A(x+1) \oplus B(y)$, in which $x$ and $y$ can be instantiated by all possible values. Such reaction schemes (but possibly with more than 2 interactants) were used in [37] to model chemical reactions in biochemical systems.

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

The process $A(2)|B(5)$ may communicate on channel $z$ and become $A(3) \mid B(5)$. The strength of this interaction is 7. It is computed by summing up the value of the $x$-attribute of $A(2)$ and the value of the $y$-attribute of $B(5)$.

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

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

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

### 4.3 Non-deterministic Operational Semantics

The non-deterministic operational semantics of the attributed $\pi$-calculus is given by the rules in Fig. 14 and previous closure rules from Fig. 6. The rules in Fig.

**Communication and application steps**

$$(\text{SEND})\ \frac{e_1 \Downarrow x \quad e_2 \Downarrow v_2 \quad \tilde{e} \Downarrow \tilde{v}}{e_1[e_2]!\tilde{e} \Downarrow x[v_2]!\tilde{v}} \qquad (\text{REC})\ \frac{e_1 \Downarrow x \quad e_2 \Downarrow v_2}{e_1[e_2]?\tilde{y} \Downarrow x[v_2]?\tilde{y}} \qquad (\text{TUP})\ \frac{\wedge_{i=1}^{n} e_i \Downarrow v_i}{(e_i)_{i=1}^{n} \Downarrow (v_i)_{i=1}^{n}}$$

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

**Program errors**

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

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

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

14 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 $\pi$-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 senders $x[v_1]?\tilde{y}.P_1$ and receivers $x[v_2]!\tilde{v}.P_2$ 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 $\bot$ 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 the changes are imported from the changes in communication, application, and error steps.

*Example 4.* Let us consider a client server system in the attributed $\pi$-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:

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

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

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

No other third client has access to the communication of server with some client, since they use a private channel for their exchange. Note however, that the second communication action gets highest priority, so that client *Client*(`fish`) cannot not intervene before *Client*(`chicken`) obtained the price for the `chicken`.

The next lemma extends on Lemma 1. It states that application of definitions is confluent, so that exhaustive application must lead to a unique result in case of termination.

**Lemma 3.** *The rewrite relation* $\xrightarrow[nd]{app}$ *is confluent on equivalence classes of processes modulo structural congruence. The normal forms of this rewrite system are equivalence classes of processes of the form* $(\nu\tilde{x})\prod_{i=1}^{n} P_i$ *such that all* $P_i$ *are sums or match some irreducible defined process* $A_i(\tilde{e}_i)$ *with* $\neg\exists\tilde{v}.\tilde{e}_i \Downarrow \tilde{v}$.

*Proof.* A standard analysis of the structural congruence shows the following claim.

*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:

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

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. □

$$[\![v?\tilde{y}.P]\!] = v[\lambda x.x]?\tilde{y}.[\![P]\!] \qquad [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$$

$$[\![v{:}r!\tilde{v}'.P]\!] = v[r]!\tilde{v}'.[\![P]\!] \qquad [\![M_1 + M_2]\!] = [\![M_1]\!] + [\![M_2]\!] \qquad [\![\mathbf{0}]\!] = \mathbf{0}$$

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

**Fig. 15.** Encoding the $\pi$-calculus with priorities $(R, <)$ into $\pi(\lambda(R), <)$, and the stochastic $\pi$-calculus into the attributed $\pi$-calculus with stochastic semantics.

The following two properties of $\pi(\mathcal{L})$ are precisely the same that we obtained for the $\pi$-calculus with priorities (but now from Lemma 3 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 3.

**Lemma 4.** *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 $P \xrightarrow[nd]{app}{}^{*} P'$ (this is the reflexive case in our definition of $\xrightarrow[nd]{app}{}^{*}$), so that $P \Downarrow P'$ by rule (CONV). Conversely, if $P \Downarrow P'$, then $P \xrightarrow[nd]{app}{}^{*} P'$ by rule (CONV). Since $[P]_{\equiv}$ is irreducible with respect to $\xrightarrow[nd]{app}$ by Lemma 3, this yields $P \equiv P'$.

### 4.4 Encoding of the $\pi$-Calculus with Priorities

The $\pi$-calculus with priorities in $(R, <)$ from Section 2 can be encoded into the attributed $\pi$-calculus $\pi(\lambda(R), <)$. Here we use all priorities in $R$ as constants, and consider all of them as the only successful values. The order on priorities remains unchanged.

The translation is given in Fig. 15. We map senders $v{:}r!\tilde{v}'.P$ to $v[r]!\tilde{v}'.P$ and replace all receivers $v?\tilde{y}.P$ by $v[\lambda x.x]?\tilde{y}.P$.

**Theorem 1.** *The encoding of the $\pi$-calculus with priorities $(R, <)$ into the attributed $\pi$-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 $\pi$-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.* $[\![P[\tilde{v}/\tilde{y}]]\!] = [\![P]\!][\tilde{v}/\tilde{y}]$ for $\tilde{v}$ be 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 of $P \equiv Q$ and $[\![P]\!] \equiv [\![Q]\!]$ respectively. Here 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 same error rule is provided by both calculi. We omit the details.

(E.PREF) Suppose that $P \xrightarrow[nd]{err} \bot$, then $P = \pi.Q + M$ for some $Q$ and $M$, and either $\pi = c?\tilde{y}$ or $\pi = c{:}r!\tilde{v}$. In either case, a prefix with constant subject translates to a prefix with constant subject as well, thus $\neg\exists\pi'.[\![\pi]\!] \Downarrow \pi'$ by (SEND) and (REC). Therefore, by (E.PREF), $[\![P]\!] \xrightarrow[nd]{err} \bot$.

Now, suppose now that $[\![P]\!] \xrightarrow[nd]{err} \bot$. Hence $[\![P]\!] = \pi.Q + M$ such that the following rule applies:

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

Since sums can only be obtained by translating sum, there process $P$ must match some process $\hat{\pi}.\hat{Q} + \hat{M}$. Here, $\hat{\pi}$ must be a prefix of the $\pi$-calculus with priorities, and thus of the form $v?\tilde{y}$ or of the form $v{:}r!\tilde{v}'$. In the first case, hypothesis $\neg\exists\pi'.\pi = v[\lambda y.y]?\tilde{y} \Downarrow \pi'$ is valid only if $v \in \mathit{Consts}$, since abstractions and variables are values, so they converge to themselves by rule (VAL). Therefore, by rule (E.PREF), $P \xrightarrow[nd]{err} \bot$. The second case is similar.

(E.CONSTR) By contradiction. So suppose that (E.CONSTR) proves $[\![P]\!] \xrightarrow[nd]{err} \bot$. Thus $[\![P]\!] = \pi_1.P_1 + M_1 \mid \pi_2.P_2 + M_2$ and the following rule applies:

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

By inspecting 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 all relations in $\rho \in \{\Downarrow, \rightarrow\} \cup \{\xrightarrow[nd]{\alpha} \mid \alpha \in \{app\} \cup R\}$, and all processes $P', \hat{P}$ of the $\pi$-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 $\hat{P} \;\rho\; \hat{Q}$.

The proof of 1. is by structural induction on derivations of $P \;\rho\; P'$. We have to consider all rules of the operational semantics of the $\pi$-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.

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

By the proofs of rules (APP) and (COM), it is provided that if $P \Downarrow P'$ then $[\![P]\!] \Downarrow [\![P']\!]$ and if $P' \xrightarrow[nd]{r} Q$ then $[\![P']\!] \xrightarrow[nd]{r} [\![Q]\!]$. 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 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 later.

(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:

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

(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):

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

(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:

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

(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 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. □

### 4.5 Encoding π@ for Dynamic Compartments

The $\pi$-calculus with priorities polyadic synchronization $\pi@$ was introduced in [13] as a uniform framework in which to encode various aspects of spatial compartment organization, and in particular to be able the express Bioambients [21] and Brane calculus [22].

We show how to encode $\pi@$ with priorities in an ordered set $(R, <)$ into the attributed $\pi$-calculus $\pi(\lambda(R, =), <)$, where priorities serve as constants and successful values. This result shows that attributed $\pi$-calculus $\pi(\lambda(\{1, 2, 3\}, =), <)$ with 3 levels of priorities $1 < 2 < 3$ inherits correct encodings of Bioambients and Brane.

The syntax of $\pi@$ is the same as for the $\pi$-calculus with priorities, except that that communication now acts on nonempty tuples of channels, that priorities are assigned to both the sender and the receiver. 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 such that tuples of channels and priorities are tested for equality before communication. Otherwise, the nondeterministic semantics of the $\pi$-calculus with priorities remains unchanged:

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

Our encoding is a two step approach, each step is defined by one encoding: The first encoding $[\![\_]\!]_{pre} : \pi@ \to \pi@$ is a preprocessing step. Its purpose is to ensure that in the second step, we only need to deal with subject tuples of equal size Thus, it obtains an $n \in \{1, 2, \dots\}$ such that for all tuples $(x_1, \dots, x_m)$ it is true that $m <= n$. Furthermore, it chooses a channel name $x$, which is not used in the program to encode. The existence of such an $x$ is ensured since the set of channel names *Vars* is infinite. The name $x$ is used to extend subject tuples of smaller size in the following way:

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

For the second step, we define $\pi(\lambda(=))$, whose priorities are given by $R = \mathbb{N}$. Besides *Vars*, `Bool`, and $R$, its set of constants contains 0 to denote unsuccessful synchronization. The encoding $[\![\_]\!] : \pi@ \to \pi(\lambda(=))$ is compositional; here, only the encoding of communication prefixes deserves special attention, which

assumes that all subject tuples are of the same size $n$. We recursively define functions $eq_n$ that check equality of n-tuples:

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

We use $eq_n$ to check equality of subject tuples and priorities in communication constraints:

$$
\begin{array}{l}
[\![(x_1, \ldots, x_n){:}r!\tilde{z}.P]\!] = x_1[\lambda y_2 \ldots \lambda y_n \lambda r'. \\
\qquad\qquad \texttt{if } eq_n(x_2, \ldots, x_n, r, y_2, \ldots, y_n, r') \texttt{ then } r \texttt{ else } 0]!\tilde{z}.[\![P]\!] \\
[\![(y_1, \ldots, y_n){:}r'?\tilde{z}.P]\!] = y_1[\lambda u.u\ y_2 \ldots y_n r']?\tilde{z}.[\![P]\!]
\end{array}
$$

An output with subject $(x_0, \ldots, x_n)$ and priority $r$ is translated as an output with subject $x_1$ and a constraint that checks whether its arguments bound to $y_2 \ldots \tilde{y}_n, r'$ equal $x_2 \ldots x_n, r$ according to $eq_n$. If this test is successful the constraint evaluates to the priority of the sender $r$. Dually, an input with subject $(y_1, \ldots, y_n)$ and priority $r'$ is translated as an input with subject $y_1$ and a constraint applying its argument to $y_2, \ldots, y_n$ and $r'$. A successful synchronization evaluates to priority level $r$. We also have to translate the definitions of molecules in use:

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

and consider reduction with respect to a set of such encoded definitions. The correctness proof of the translation relies on the correctly testing tuple equality.

**Lemma 5.** *For all variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ it is true that:*

1. $eq_n(x_1, \ldots, x_n, x_1, \ldots, x_n) \Downarrow \texttt{true}$.
2. $eq_n(x_1, \ldots, x_n, y_1, \ldots, y_n) \Downarrow \texttt{false}$ *if* $x_i \neq y_i$ *for some* $1 \leq i \leq n$.

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

$$
\begin{array}{l}
eq_{n+1}(x_1, \ldots, x_{n+1}, x_1, x_1 \ldots, x_{n+1}) =_{df} \\
\quad \texttt{if } x_{n+1} = x_{n+1} \texttt{ then } eq_n(x_1, \ldots, x_n, x_1 \ldots, x_n) \texttt{ else false}.
\end{array}
$$

By rules (EQ$_1$) and (COND$_1$) apply, we obtain $eq_n(x_1, \ldots, x_n, x_1, x_1 \ldots, x_n)$. By induction hypothesis:

$$eq_n(x_1, \ldots, x_n, x_1, \ldots, x_n) \Downarrow \texttt{true}$$

(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 $x_i \neq y_i$ and:

$$
\begin{array}{l}
eq_{n+1}(x_1, \ldots, x_{n+1}, x_1, y_1 \ldots, x_{y+1}) =_{df} \\
\quad \texttt{if } x_{n+1} = y_{n+1} \texttt{ then } eq_n(x_1, \ldots, x_n, y_1 \ldots, y_n) \texttt{ else false}.
\end{array}
$$

Suppose $1 \leq i \leq n$ and $x_{n+1} = y_{n+1}$. Then, by rules ($\text{EQ}_1$) and ($\text{COND}_1$), we obtain $eq_n(x_1, \ldots, x_n, y_1 \ldots, y_n)$, which evaluates to `false` by induction hypothesis. If $i = n + 1$, then we know by rules ($\text{EQ}_2$) and ($\text{COND}_2$):

$$\texttt{if } x_{n+1}{=}y_{n+1} \texttt{ then } eq_n(x_1, \ldots, x_n, y_1 \ldots, y_n) \texttt{ else false} \Downarrow \texttt{false}$$

□

**Theorem 2 (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 1, we proofed the encoding of the $\pi$-calculus with priorities in $\pi(\lambda(R), <)$ correct. The non-deterministic semantics of $\pi@$ and the $\pi$-calculus with priorities are the same, with only one exception: the communication rule. Therefore, in the following proof, we only consider the communication rule.

($\text{COM}$) We define $\kappa_o$ and $\kappa_i$:

$$\begin{aligned}\kappa_o &= \lambda y_2 \ldots \lambda y_n \lambda r'.\\ &\quad \texttt{if } eq_n(x_2, \ldots, x_n, r, y_2, \ldots, y_n, r') \texttt{ then } r \texttt{ else } 0\\ \kappa_i &= \lambda e.e x_2 \ldots x_n r\end{aligned}$$

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

$$\frac{|\tilde{y}| = |\tilde{z}|}{P = (x_1, \ldots, x_n){:}r?\tilde{y}.P_1 + M_1 \mid (x_1, \ldots, x_n){:}r!\tilde{z}.P_2 + M_2 \xrightarrow[nd]{r} P_1[\tilde{v}/\tilde{y}] \mid P_2 = P'}$$

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 ($\text{COM}$) rule of $\pi(R, <)$ applies:

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

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.

($\text{COM}$) We define $\kappa_o$ and $\kappa_i$:

$$\begin{aligned}\kappa_o &= \lambda y_2 \ldots \lambda y_n \lambda r'.\\ &\quad \texttt{if } eq_n(x'_2, \ldots, x'_n, \hat{r}', y_2, \ldots, y_n, r') \texttt{ then } r \texttt{ else } 0\\ \kappa_i &= \lambda e.e x_2 \ldots x_n \hat{r}\end{aligned}$$

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

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

Inspecting the translation reveals that 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:

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

□

It might be worth noticing, however, that all $\lambda$ expressions terminate in linear time. This can be seen by inspecting the occurring terms.

### 4.6 Type System

We present a type system for the $\pi(\mathcal{L})$, which integrates the simple type system for the $\lambda$-calculus $\mathcal{L}$ into the type system of the $\pi$-calculus. Type safety prevents us from type errors, but does not necessarily exclude other errors like division by 0.

We will show that our 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 basic attribute language from Fig. 11 and its extension $(\mathbb{N}_0, +, =)$ 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 defined 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 constant such as `Int`, `Bool`, and `String` and define *Types* with these constants by the following grammar:

| | | | |
|---|---|---|---|
| type constants | $\iota$ | $::=$ `Int` $\mid$ `Bool` $\mid \ldots$ | |
| types | $\tau, \sigma$ | $::= \iota$ | 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 $\tau$ and channel arguments by $\sigma$. 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 $\tau_1$ 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* $\Gamma$ and $\Delta$ are sets of type assignments for variables $x{:}\tau$ and process names $A{:}\tilde{\tau}$.

In the typed version of the attributed $\pi$-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 $\lambda$-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:

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

Rules for typing expressions and processes are given in Fig. 16. Typing rules for expressions are standard from simply typed $\lambda$-calculus. 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 $\pi$-calculus with priorities in Fig. 7. Typing rule (T.NEW) now check 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. 7 and are note repeated here.

**Proposition 5 (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 $\Gamma \vdash v{:}\tau$.*

*Proof (Hint).* 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 $\Gamma \vdash e[v/x] : \sigma$. □

**Lemma 6.** *The following properties holds for the typing rules of processes:*

**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

1. *(strengthening) if* $\Gamma, x{:}\tau \vdash P$ *and* $x \notin \mathit{fv}(P)$ *then* $\Gamma \vdash P$,
2. *(weakening) if* $\Gamma \vdash P$ *and* $x \notin \mathit{fv}(P)$ *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 (and depend on) to 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 if for scope extrusion, that is, assuming $x \notin \mathit{fv}(Q)$, $\Gamma \vdash (\nu x{:}\tau)(P \mid Q) \Leftrightarrow \Gamma \vdash (\nu x{:}\tau)P \mid Q$.
($\Rightarrow$) By rules (T.NEW) and (T.PAR), we have $\Gamma, x{:}\tau \vdash P$ and $\Gamma, x{:}\tau \vdash Q$. Since $x \notin \mathit{fv}(Q)$, 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 7.** *Let $P$ be a process with definitions $\mathcal{D}$ in the attribute $\pi$-calculus with a type safe attribute language, and $\Delta$ 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 6(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 6(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 6(3) yield $\Gamma \vdash P[\tilde{v}/\tilde{x}]$. □

**Theorem 3 (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 $\Gamma$ 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 7, 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 ensure 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 6(3), $\Gamma \vdash P_1[\tilde{v}/\tilde{y}]$. Finally, from $\Gamma \vdash P_2$ and rule (T.PAR), $\Gamma \vdash P_1[\tilde{v}/\tilde{y}] \mid P_2$. □

The translation from the $\pi$-calculus with priorities $(R, <)$ into $\pi(R, <)$ can be refined such that types are preserved. In order to do so, we assume that there exists a type constant $R$ by which to type priorities $r \in R$ during translation.

First of all, we have to adapt the translation to typed expressions, and to introduce annotate types to priorities:

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

Second, we have to translate types of the $\pi$-calculus with priorities into types of the attributed $\pi(R, <)$.

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

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

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

**Fig. 17.** 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 $\pi$-calculus in Fig. 9.

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

**Proposition 6 (Type preservation).** *Let $P$ be a process of the $\pi$-calculus with priorities and $\Gamma$ a type environment such that $\Gamma \vdash P$ then $[\![\Gamma]\!] \vdash [\![P]\!]$.*

The proof is straightforward by structural induction over type derivations.

Finally, notice that one cannot extend this type preservation result to the encoding of $\pi@$. Finding a convincing type system for $\pi@$ is nontrivial, since there the same channel may be used in several tuples, each of which may receive arguments of different types.

## 5 Stochastic Semantics of Attributed Pi-Calculus

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 $R = \mathbb{R}_+^\infty$. As in the stochastic $\pi$-calculus, we assign highest priority to communication steps with infinite rates, and lowest priority to all others.

### 5.1 Stochastic Semantics

The axioms of the stochastic semantics of $\pi(\mathcal{L})$ is given in Fig. 17. The whole presentation of the article is done such, that only very few changes are needed with respect to the stochastic $\pi$-calculus. In particular, both calculi have the same closure rules, that were already presented in Fig. 9. This is a major improvement compared to the conference version of the present article [34].

The main difference concernes the new communication rule ($\text{COM}_\ell$), 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 $\pi$-calculus with priorities.

**Proposition 7.** *If the partial order $<$ on successful values of $\mathcal{L}$ satisfies $r < \infty$ and $\neg r < r'$ for all $r, r' \in \mathbb{R}_+$ then for all processes $P, P'$:*

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

The proof is mostly the same as for Proposition 3, which relates the two operational semantics of the stochastic $\pi$-calculus. The only minor difference is in the treatment basic interaction steps.

### 5.2 Encoding of the Stochastic $\pi$-Calculus

As we have seen, the stochastic $\pi$-calculus has the same syntax than the $\pi$-calculus with priorities in $\mathbb{R}_+^\infty$. Indeed, we can encode the stochastic $\pi$-calculus into the attributed $\pi(\emptyset)$ by the same translation that encodes the $\pi$-calculus with priorities into the attributed $\pi(\emptyset)$. See Fig. in Fig. 15.

**Theorem 4.** *The encoding of the stochastic $\pi$-calculus into the stochastic attributed $\pi$-calculus $\pi(\lambda(\mathbb{R}_+^\infty)$ is correct, in that for all processes $P, P', \hat{P}$, attributed processes $Q$ and labels $\alpha \in \{r, \infty(r) \mid r \in \mathbb{R}_+\}$:*

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 the stochastic $\pi$-calculus and $\pi(\mathcal{L})$ directly build on their non-deterministic semantics. We proved in Section 4.4 that the translation is invariant under substitution and reflects and preserves both structural congruence and errors. Furthermore, we proved 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 $\ell$ 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 $[\![\hat{Q}]\!] = Q$.

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

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

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):

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

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 ($\text{COM}_\ell$) must be applicable as follows:

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

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 ($\text{COM}_\ell$) applies as follows:

$$\frac{\ell = (i_1, j_1, i_2, j_2) \quad \hat{\pi}_{i_1}^{j_1} = x?\tilde{y} \quad \hat{\pi}_{i_2}^{j_2} = x{:}r!\tilde{z} \quad |\tilde{y}| = |\tilde{z}|}{\hat{P} = (\nu\tilde{x}) \prod_{i=1}^n \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j \xrightarrow[\ell]{r} (\nu\tilde{x})(\prod_{\substack{i=1 \\ i\neq i_1, i_2}}^n \sum_{j=1}^{m_i} \hat{\pi}_i^j.\hat{P}_i^j \mid \hat{P}_{i_1}^{j_1}[\tilde{v}/\tilde{y}] \mid \hat{P}_{i_2}^{j_2}) = \hat{Q}}$$

Given two process $P, Q$ 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 $[\![Q']\!] = Q'' \equiv [\![Q]\!]$. 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 appart, then $S(P_1, Q) = S(P_2, Q)$.

*Proof.* Suppose that $P = (\nu x_1) \ldots (\nu x_k) \prod_{i=1}^m \sum_{j=1}^{n_i} M_i^j$ 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:

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

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

We next prove the theorem for reductions with finite rates.

*Claim.* Translation preserves and reflect 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 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 showed in the proof of Theorem 1 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 7 we know that $[\![\hat{P}]\!] \rightarrow Q$. Theorem 1 on the preservation of the non-deterministic semantics shows that there exists a process $\hat{Q}$ such that $[\![\hat{Q}]\!] \equiv Q$ and $\hat{P} \rightarrow \hat{Q}$. 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 1. 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 $P \xrightarrow{\infty(r)} P'$ then $[\![P]\!] \xrightarrow{\infty(r)} [\![P']\!]$
2. if $[\![\hat{P}]\!] \xrightarrow{\infty(r)} Q$ then exists $\hat{Q}$ such that $\hat{P} \xrightarrow{\infty(r)} \hat{Q}$ 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. □

### 5.3 Encodings Variants

As stated before, the most frequent variant of the the stochastic $\pi$-calculus annotates stochastic rates to channels rather than to communication prefixes. Here, we present translations for these variants into the attributed $\pi$-calculus. Since we cannot formalize such variants here, and since the proof should work as usual, we refrain from proving the correctness of these encodings here.

The most basic variant of this the the stochastic $\pi$-calculus of Biospi [7] and SPiM [16]. It can be encoded into $\pi(\mathbb{R}_+^\infty)$ as follows. Here it is relevant that the attributed $\pi$-calculus permits pairs, and that it allows for expressions in sender and receiver positions:

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

A more expressive variant is SPiCO, the stochastic $\pi$-calculus for concurrent objects [15]. Spice assumes a finite set of function symbols $\Sigma$ that are annotated to all sending and receiving actions. The communication constraint is that the function symbols of receivers and senders must be the same. This is a weak form of polyadic synchronization. We can encode SPiCO into $\pi(\mathbb{R}_+^\infty, \Sigma, =)$ as follows, where $a, b \in \Sigma$:

$$\begin{array}{l}
[\![(\nu x{:}r)P]\!] = (\nu x)[\![P]\!][\langle x, r\rangle/x] \\
[\![x?a(\tilde{y}).P]\!] = (\texttt{fst}\ x)[\lambda z.\texttt{if}\ z{=}a\ \texttt{then}\ (\texttt{snd}\ x)\ \texttt{else}\ 0]?\tilde{y}.[\![P]\!] \\
[\![x!b(\tilde{y}).P]\!] = (\texttt{fst}\ x)[b]?\tilde{y}.[\![P]\!]
\end{array}$$

Finally, in the conference version of the attributed $\pi$-calculus at CMSB'08 [34], 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:

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

## 6 Modeling Techniques and Biological Examples

We present two examples for modeling biological systems with attribute dependent rates, in order to illustrate the advantages of $\pi(\mathcal{L})$ as a modeling language. The first example introduces a simplistic, discrete spatial model of Euglena's phototaxis [38], while the second more complex example shows how to deal with cooperative enhancement in gene regulation at the lambda switch [23].

### 6.1 Space

Spatial aspects of molecular systems gain increasing interest in systems biology [39]. We illustrate the use of attribute dependent rates for spatial modeling with a simple example, which is the modeling of Euglena's light dependent motion (phototaxis) in $\pi(\lambda(\texttt{Int}, +, -, *, /))$. More complex compartment structures can be modeled in the attributed $\pi$-calculus as well, as we will show by encoding $\pi@$ in Section 4.5.

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, [40]: if the amount of light decreases it moves towards a lighter zone, and vice versa. This behavior is specified in the attributed $\pi$-calculus model in Fig. 18. The model assumes two light sources of intensity $i = 0.5$ and $i = 0.6$ and distinguishes $m$ discrete zones of water of depths $\texttt{d} \in \{0, 1, 2, 3, \ldots m\}$, where $m$ is defined by the model parameter `m`.

The attribute language $(\lambda(\texttt{Int}, +, -, *, /))$provides the set of integers `Int` as constants of base type, and functional constants for addition (`+`), subtraction (`-`), multiplication (`*`) and division (`/`). We freely use infix notation. The evaluator imposes the following additional rules:

$$\frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2 \quad n_1 \odot_{\mathbb{N}} n_2 = n}{e_1 \odot e_2 \Downarrow n},$$

with $\odot \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}\}$. All natural numbers are successful, i.e. $R = \mathbb{N}$, but not 0 or negative numbers. Only a single level of priorities is needed, such that we assume $\neg n < m$ for all $n, m \in R$.

Euglenas at depth `d` are defined by processes `Euglena(d)`. There are two channels `up` and `down` for upward respectively downward motions. The speed of upward motions is `d*(1-i)` where $\texttt{i} \in [0, 1]$ is the current light intensity, and the speed of the downwards movement is `(m-d)*i`. Euglenas start at level `m/2`. For $\texttt{i} = 0.5$, they are expected to concentrate at depth level $\texttt{d} = \text{m}/2$. Note that `Euglena`(0) cannot climb further, since the value of $0 * (1 - \texttt{i})$ is 0 and thus not successful for all `i`. For the same reason, `Euglena`(m) cannot descend. The interaction partners are processes `Light(i)` modeling the light sources of intensity $\texttt{i} \in [0, 1]$. `Euglena(d)` can adapt to different intensities of light. For this we use $\lambda$`i` abstractions in its definition.

The same system can be modeled in the stochastic $\pi$-calculus, since all parameters are finitely valued: $\texttt{d} \in \{0, \ldots, m\}$ and $\texttt{i} \in \{0.5, 0.6\}$. The idea is to

**Process definitions**

```
Euglena(d) ≜ up[λi.d∗(1−i)]?().Euglena(d−1)
           + down[λi.(m−d)∗i]?().Euglena(d+1)
Light(i) ≜ up[i]!().Light(i)
         + down[i]!().Light(i)
```

**Example solution**

$$\prod_{i=1}^{n} \texttt{Euglena(m/2)} \mid \texttt{Light(0.5)} \mid \texttt{Light(0.6)}$$

**Fig. 18.** $\pi(\mathcal{L})$ model of Euglena's light-dependent motion: the rates for climbing and falling depend on the organism's current depth level and the light intensity. `m` and `n` are model parameters, representing the maximum depth level and the total number Euglena, respectively.

duplicate channels for the different levels and to define processes $\texttt{Euglena}_\texttt{d}()$ and $\texttt{Light}_{\texttt{d,i}}()$ for all possible depth levels and light intensities, see Fig. 19.

Whether arbitrary processes of $\pi(\mathcal{L})$ are expressible in the stochastic $\pi$-calculus is open, particularly for infinitely valued parameters. Even if it is possible, the size of the the stochastic $\pi$-calculus definitions may often become prohibitively large.

*Specific optimizations for light intensities* Consider a model with only one light source of intensity $i = 0.5$. Rates for upward and downward motion at levels $d$ and $m - d$ coincide, since equations $d * (1 - i)$ and $(m - (m - d)) * i = d * i$ show the same results for $i = 0.5$. Thus, in the the stochastic $\pi$-calculus model, the number of channels can be halved, see Figure 20.

This implementation is more efficient, since the computational complexity of the Stochastic Simulation Algorithm, which is called in each simulation step, see Section 7, directly depends on the number of channels. This point is explored in more detail in Section 8. Similar but less effective optimizations can be achieved for different light intensities, e.g. $\texttt{up}_{\ 1,0.25}$ and $\texttt{down}_{\ 1,0.25}$.

The same optimization can be achieved in the $\pi(\lambda(\texttt{Int}, +, -, *, /))$ model by simply replacing channels `up` and `down` by one channel `move` and removing the second summand of `Light(i)`:

```
Euglena(d) ≜ move[λi.d∗(1−i)]?().Euglena(d−1)
           + move[λi.(m−d)∗i]?().Euglena(d+1)
Light(i) ≜ move[i]!().Light(i)
```

The simulator with optimized grouping, as defined in Section 7, automatically merges up and down motions with equal rate.

### 6.2 Cooperative Enhancement

Cooperative binding is a frequent and often decisive aspect in gene regulatory networks, where proteins stabilize each other's binding to neighboring DNA sites

```
// Euglena for different depth levels
Euglena_0() ≜ down_0?().Euglena_1()
Euglena_1() ≜ up_1?().Euglena_0() + down_1?().Euglena_2()
...
Euglena_m() ≜ up_m?().Euglena_m-1()
// Light with intensity 0.5 for different levels
Light_0,0.5() ≜ down_0,0.5 : m * 0.5!().Light_1,0.5()
Light_1,0.5() ≜ up_1,0.5 : 0.5!().Light_0,0.5() + down_1,0.5 : (m − 1) * 0.5!().
    Light_2,0.5()
...
Light_m,0.5() ≜ up_m,0.5 : m * 0.5!().Light_m-1,0.5()
// Light with intensity 0.6 for different levels
Light_0,0.6() ≜ down_0,0.6 : m * 0.6!().Light_1,0.6()
Light_1,0.6() ≜ up_1,0.6 : 0.4!().Light_0,0.6() + down_1,0.6 : (m − 1) * 0.6!().
    Light_2,0.6()
...
Light_m,0.6() ≜ up_m,0.6 : m * 0.4!().Light_m-1,0.6()
```

**Example solution**

$\prod_{i=0}^{n} \mathsf{Euglena}_{m/2}() \mid \prod_{i=0}^{m} (\mathsf{Light}_{m,0.5}() \mid \mathsf{Light}_{m,0.6}())$

**Fig. 19.** A model of Euglena's in the stochastic $\pi$-calculus. Distinct definitions for $\mathtt{Euglena}_d$ are used for all depth levels $d \in \{0, \ldots m\}$ and light intensity $\mathtt{i} \in \{0.5, 0.6\}$.

```
// Euglena for different depth levels
Euglena_0() ≜ move_m?().Euglena_1()
Euglena_1() ≜ move_1?().Euglena_0() + move_m-1?().Euglena_2()
...
Euglena_m() ≜ move_m?().Euglena_m-1()
// Light for different depths levels
Light_1,0.5() ≜ move_1 : 0.5!().Light_1,0.5()
...
Light_m,0.5() ≜ move_m : m * 0.5!().Light_m,0.5()
```

**Example solution**

$\prod_{i=0}^{n} \mathsf{Euglena}_{m/2}() \mid \prod_{i=1}^{m} \mathsf{Light}_{m,0.5}()$

**Fig. 20.** An optimized model of Euglena's light dependent motion in the stochastic $\pi$-calculus for the light intensity 0.5.

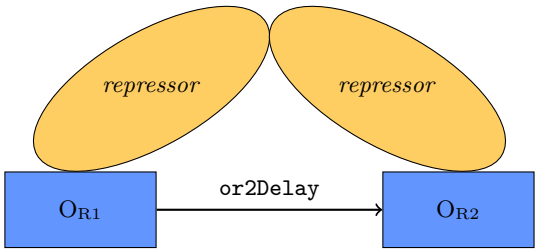


**Fig. 21.** The decay of the *repressor*-$O_{R2}$ complex: in order to make the decay rate of the *repressor*-$O_{R2}$ complex dependent on $O_{R1}$'s state, the two sites communicate on `or2Delay` before $O_{R2}$ unbinds.

by adhesive contacts. In quantitative terms, the decay 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 [24, 23], cooperative enhancement can be modeled in the stochastic $\pi$-calculus. It however requires nontrivial encodings, that can be alleviated within $\pi(\mathcal{L})$.

A well understood instance of cooperative binding occurs during transcription initiation control at the $\lambda$ switch. The $\lambda$ switch is a segment of the DNA of bacteriophage $\lambda$. It contains two binding sites $O_{R1}$ and $O_{R2}$, where *repressor* and *cro* proteins can bind. An unstable binding of a *repressor* molecule to $O_{R2}$ is stabilized by the simultaneous presence of another *repressor* at the neighboring site $O_{R1}$. As illustrated in Fig. 21, the two proteins actually touch each other.

A $\pi(\mathcal{L})$ model of cooperative binding at $O_{R2}$ is presented in Fig. 22. It contains the parametric definition `Prot(type)`, which emulates the behavior of the proteins. The parameter `type` can be instantiated by either `'rep'` or `'cro'`, for modeling *repressor* or *cro* proteins respectively. Proteins can bind to both sites $O_{R1}$ and $O_{R2}$. Free sites are defined by processes $O_{R1}()$ and $O_{R2}()$, 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 $\lambda$`r.r`.

Now consider the models for the protein bound DNA sites. $O_{Ri}$`Bound(type,release)` describes the unbinding from the occupied site $O_{Ri}$, 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, $O_{R1}$ and $O_{R2}$ are linked via the channel `or2Delay`, illustrated in Fig. 21. 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

**Global channel names**

```
bind, or1Delay, or2Delay
```

**Process definitions**

```
Prot(type) ≜ (ν release:_)bind[_]!(type,release).
             release[λr. r]?().Prot(type)
O_R1() ≜
   bind[λ_. 0.098]?(type,release).O_R1Bound(type,release)
 + or2Delay['free']!.O_R1()
O_R1Bound(type,release) ≜
   release[
     if type='rep' then 0.155 else
     if type='cro' then 2.45
     ]!().O_R1()
 + or2Delay[type]!().O_R1Bound(type,release)
O_R2() ≜
  bind[λ_. 0.098]?(type,release).O_R2Bound(type,release)
O_R2Bound(type,release) ≜
   or2delay[λ t.
     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[∞]!().O_R2()
```

**Example solution**

$$O_{R1}() \mid O_{R2}() \mid \prod_{i=1}^{28} Prot('rep') \mid \prod_{i=1}^{67} Prot('cro')$$

**Fig. 22.** $\pi(\mathcal{L})$ model of cooperative binding between $O_{R1}$ and $O_{R2}$ at the $\lambda$ switch.

the bound protein, but also on the state of $O_{R1}$, which can be either `'free'`, bound to `'rep'` or bound to `'cro'`.

A previous model [23] in the the stochastic $\pi$-calculus calculus [1] requires to keep $O_{R2}$ constantly informed about state changes of $O_{R1}$, 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 [24] in SPiCO [5], the stochastic $\pi$-calculus calculus with concurrent objects, requires significantly fewer updates. In $\pi(\mathcal{L})$, reaction rates directly depend on the attribute values of the interaction partners. State changes are propagated without additional communication steps, preventing deadlocks.

```
Simulate−naive(P, t)
  // process P, time point t ∈ ℝ
  let P_1 be such that P ⇓ P_1
       // P_1 is obtained from P by exhaustively applying definitions.
       // This computation may diverge.
  if P_1 ⟶^{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_+^∞} | ∃P_2. P_1 ⟶^{r}_{ℓ} P_2}  // (COM_ℓ)
  if Reacts ∩ (ℕ_4 × {∞}) = ∅
  then
     let ((ℓ, r), Δ) = Gillespie(Reacts)  // (SUM)
     let P_2 such that P_1 ⟶^{r}_{ℓ} P_2
     Simulate−naive(P_2, t + Δ)
  else
     select (ℓ, ∞) ∈ Reacts with equal probability //(COUNT)
     let P_2 such that P_1 ⟶^{r}_{ℓ} P_2
     Simulate−naive(P_2, t)
```

**Fig. 23.** Naive simulator interpreting the stochastic semantics.

## 7 Stochastic Simulator

The development of the simulator, as presented in this section, closely follows the stochastic semantics of the attributed $\pi$-calculus in terms of CTMCs of Section 5. 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 $\pi$-calculus or SPiCO [6, 5, 7].

The stochastic semantics for $\pi(\mathcal{L})$ induces the naive stochastic simulator given in Fig. 23. 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 $t + \Delta$.

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 raises an exception (which kills its continuation). Note that error checking may run into infinite loops or raise errors too. If $P$ does converge to an error-free process $P_1$ then $P_1$ is uniquely determined up to structural congruence (Lemma 3) 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 [41] 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 $S$ is $\Delta = -ln(1/U)/s$ for some uniformly distributed random number $0 < U \leq 1$.

In order to compute *Reacts*, which have to enumerate all possible applications of the communication rule ($\text{COM}_\ell$). Here we have to evaluate all evaluation constraints, by applying the evaluation algorithm for the attribute language $\mathcal{L}$.

Most fortunately, the Markov chain 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 whether two solutions are structurally congruent (rules (SUM) and (COUNT)), which is a graph isomorphism complete problem [42].

In order to increase efficiency of the naive simulation algorithm, we apply an idea exploited already in the BioSpi implementation [7]. 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 $S$. The strategy is to *group* all reactions, using the same channel $x$ and the same constraints $e_2 e_1$, modulo evaluation of $e_1$ and $e_2$, i.e. all reactions with $x[v_1]! \ldots$ and $x[v_2]? \ldots$ for some $v_1$ and $v_2$. We then apply the Gillespie's algorithm to such *grouped* reactions.

A *group label* for 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:

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

$L$ identifies reaction groups by the communication channel $x$, the constraint value of the sender $v$ and the rate yielding the application of the receivers $\lambda$ abstraction to $v$. Consider e.g. the solution $S = A1() \mid A2() \mid B()$, with the following process definitions:

$\mathsf{A1()} \triangleq \mathsf{x[\lambda i.1-i]?().0}$ $\qquad$ $\mathsf{A2()} \triangleq \mathsf{x[\lambda i.i]?().0}$
$\mathsf{B()} \triangleq \mathsf{x[0.5]!().0}$

We obtain one reaction group with label $(\mathtt{x}, 0.5, 0.5)$ containing all reactions in $S$. An alternative definition for the grouping is obtained with label $L = (x, v_1, v_2)$:

$$\mathit{Reacts}(L) = \{((i_1, j_1, i_2, j_2), r) \in \mathit{Reacts} \mid \exists \tilde{y} \exists \tilde{v}.\pi_{i_1}^{j_1} \Downarrow x[v_1]!\tilde{y},\ \pi_{i_2}^{j_2} \Downarrow x[v_2]?\tilde{v}\}$$

However, as it bases on the equality check of receiver constraints, i.e. $\lambda$-abstractions, this grouping seems to be less effective. E.g. for the solution $S$ as defined above, we need to introduce two reaction groups with labels $(\mathtt{x}, 0.5, \lambda\mathtt{i.1-i})$

and $(\mathtt{x}, 0.5, \lambda \mathtt{i}.\mathtt{i})$ to cover all reactions. In our experiments in Section 8, we observed a strong dependency of the computation time of the simulator on the number of reaction groups. Thus, we stick to our initial definition.

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 reactions if there are any:

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

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

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

The cardinality of *GReacts* is linear in the size of $P_1$ in many practically relevant cases, for instance, only a fixed number of values will ever be used. In contrast, the cardinality of set *Reacts* becomes quickly quadratic in the size of $P_1$, for instance if all senders and receivers may interact.

Fig. 24 gives a simulation algorithm based on grouped reactions. In contrast to the naive simulator, it first selects a grouped reaction by the 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 8.** *$\mathit{prop}(x, v, r) = (\mathit{out}(x, v) * \mathit{in}(x, v, r) - \mathit{mixin}(x, v, r)) * r$, if the solution does not contain infinite rates.*

*Proof.* By distributivity, $\sum_{(\ell,r)\in \mathit{Reacts}(L)} r = r * \sum_{(\ell,r)\in \mathit{Reacts}(L)} 1 = r * \sharp \mathit{Reacts}(L)$. Thus, it is enough to show that $\mathit{out}(x, v) * \mathit{in}(x, v, r) - \mathit{mixin}(x, v, r) = \sharp \mathit{Reacts}(L)$. Consider the set $C(x, v, r) = (\{(i, j) \mid \exists \tilde{v} :\ \pi_i^j \Downarrow x[v]!\tilde{v}\} \times \{(i, j) \mid \exists v' \exists \tilde{y} : \pi_i^j \Downarrow x[v']?\tilde{y},\ v'v \Downarrow r\}) \times \{r\}$. Its elements are labels of the form $((i_1, j_1, i_2, j_2), r)$ and its cardinality given by $\sharp C(x, v_1, v_2) = \mathit{out}(x, v_1) * \mathit{in}(x, v_2)$. Inspecting rule (COM) reveals that $C(x, v_1, v_2)/\mathit{Reacts}(x, v_1, v_2) = \{((i_1, j_1, i_2, j_2), r) \mid ((i_1, j_1, i_2, j_2), r) \in C(x, v_1, v_2), i_1 = i_2\}$, such that $\sharp(C(x, v_1, v_2)/\mathit{Reacts}(x, v_1, v_2)) = \mathit{mixin}(x, v_1, v_2) = \sharp C(x, v_1, v_2) - \sharp \mathit{Reacts}(x, v_1, v_2)$.

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.

```
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) equally distributed
     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. 24.** Stochastic simulator for $\pi(\mathcal{L})$ (to be implemented incrementally).

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 $prop(x, v_1, v_2)$ incrementally, so that they don't have to be recomputed from scratch in every reduction step. This can be based on Lemma 8, since the values of $out(x, v_1)$, $in(x, v_2)$, $mixin(x, v_1, v_2)$ can be updated incrementally, when adding new solutions or canceling alternative choices by communication.

## 8 Implementation and Performance Evaluation

The goal of this section is to give an impression of the performance of the $\pi(\mathcal{L})$ simulator and not to provide a thorough performance study, which will be the subject of future work. Based on the Euglena model in Section 6.1, we compare the runtime of our implementation to existing stochastic $\pi$ simulators. The Euglena model 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 $\pi$-calculus and $\pi(\mathcal{L})$. In the following, we first give a brief introduction into our implementation of the $\pi(\mathcal{L})$ simulator and then describe the experiments and results.

We implemented the $\pi(\mathcal{L})$ simulator on top of an existing stochastic $\pi$ simulator, provided by the modeling and simulation framework JAMES II [25]. We deployed a two layer approach: the base layer is the stochastic $\pi$ simulator along the lines of [16], i.e. for each communication channel the propensity is calculated under consideration of the corresponding senders and receivers. The obtained propensities form the input for the Stochastic Simulation Algorithm that determines the next communication to perform and the sojourn time. The version of the Stochastic Simulation Algorithm (SSA), i.e. First Reaction Method (the original version), Direct Reaction Method [43], Next Reaction Method [44], can be swapped at will. The top layer implements the grouping as explained in Section 7, 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 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 $\pi(\mathcal{L})$ simulator with the stochastic $\pi$ simulator it is based upon. An external reference is given by the stochastic Pi Machine (SPiM) [6]. Since in SPiM the SSA method is fixed, we constantly use the Direct Reaction Method. 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 [45] of 383.9 Mflops. Notice, that there exists a faster version of SPiM for the Linux operating system. However, we omitted further experiments as, here, we do not report on a competition of stochastic $\pi$ simulators.

Our Euglena benchmark model comprises one light source of intensity 0.5 and 10,000 Euglenas. Among the experiments, we gradually increased the number of depth levels from $m = 10$ to $m = 100$ by steps of 10, with all Euglenas starting at $m/2$. Implementations in the stochastic $\pi$-calculus are obtained by enumerating the Euglena processes for different depth levels. To ensure comparability, we used two $\pi(\mathcal{L})$ implementations for each experiment, one enumerating the depth levels as in the stochastic $\pi$-calculus (enum) and one in the more compact form with the depth level as a parameter of Euglena (comp). We performed two sets of experiments, one with the initial and one with the optimized version of the Euglena model that halves the number of channels, see Section 6.1. We measured the time needed to simulate until time point 10.0, see Appendix A. For each experiment, we averaged over three simulation runs with small variances 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. 25 and 26. The implementations are labeled according to the used formalism, *Sto* for the stochastic $\pi$-calculus or *attr* for $\pi(\mathcal{L})$, the tool, SPiM or JAMES II, and the implementation, *Enum* or *Comp*. In the following, we first discuss the results for the initial and then those for the optimized model.

The results in Fig. 25 show a general increase of simulation time with a rising number of depth levels. Due to our choice of operating system, SPiM performs a

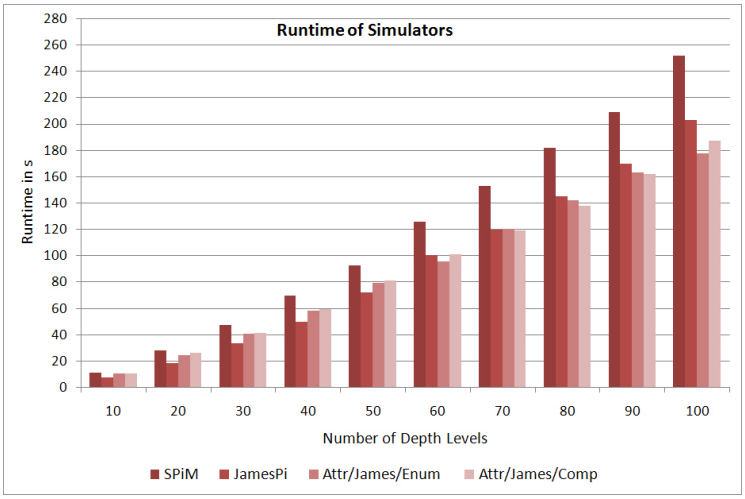


**Fig. 25.** Runtime of simulators for the unoptimized Euglena model with 10,000 Euglenas and different numbers of depth levels ranging from 10 to 100.

bit slower. All other implementations need similar amounts of time. The maximal simulation time required is about 250s. A noticeable point is that with a number of depth levels ranging from $m = 80$ to $m = 100$ the the stochastic $\pi$-calculus simulations need more time than the $\pi(\mathcal{L})$ simulations.

The first experiment set indicates that the computational complexity of the $\pi(\mathcal{L})$ simulator is moderate. For higher numbers of depth levels, it is even faster than the tested stochastic $\pi$ simulators. This can be explained, by tracing the amount of communication channels extracted from solutions. In the case of stochastic $\pi$ it constantly equals the number of depth levels, since for each depth level $d$ there exists a sending process $\texttt{Light}_{\texttt{d,05}}\texttt{()}$, even without a receiver $\texttt{Euglena}_{\texttt{d}}\texttt{()}$. In case of the $\pi(\mathcal{L})$ simulator, groups must at least contain one sender and one receiver, as otherwise no rate constant is given. Thus, less channels are created whenever Euglena does not cover all depth levels. This has noticeable impact as the performance of the underlying SSA heavily depends on the number of channels. Clearly, this effect increases with the number of depth levels. Notice, that a similar optimization can be achieved in stochastic $\pi$ simulators as well, by separating inactive communication channels that lack either a sender or a receiver.

In Fig. 26 a general increase of simulation time with a rising number of depth levels is reported. For higher depth level numbers, SPiM requires a bit more simulation time than the other simulators, about 175s at maximum. Again, this is due to our choice of operating system. Whereas the implementations in $\pi(\mathcal{L})$ show almost equal behavior, the stochastic $\pi$ implementation in JAMES II is always slightly faster.

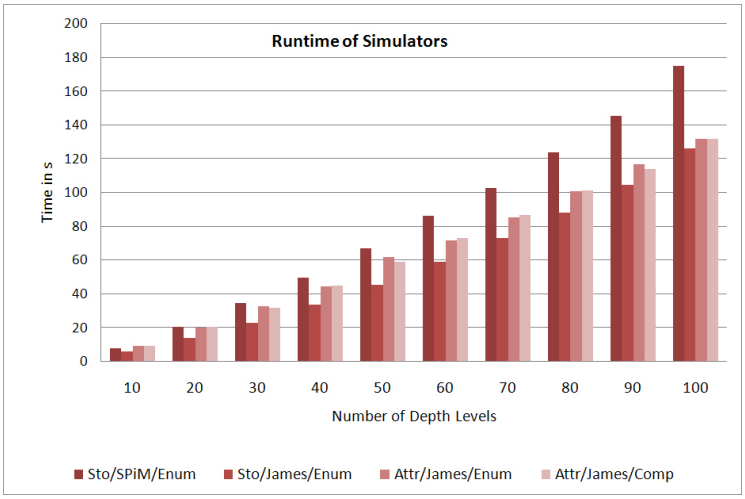


**Fig. 26.** Runtime of simulators for the optimized Euglena model with 10,000 Euglenas and different numbers of depth levels ranging from 10 to 100.

As before, the computational complexity of the $\pi(\mathcal{L})$ simulator appears to be moderate. Comparing both sets, it is apparent that the model optimization, i.e. halving the number of channels, is effective, since the maximal time amount is significantly lower. This underlines the impact of the SSA on the computational complexity of the simulator, because as already stated the SSA iterates over the number of reactions, i.e. channels, in each step to calculate what reaction and when it will occur. As the number of channels has been significantly decreased in the second experiment, the the stochastic $\pi$-calculus simulator outperforms again slightly the $\pi(\mathcal{L})$ simulator, as already observed in the first experiment for lower number of depths.

## 9 Conclusion and Outlook

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

The attributed $\pi$-calculus forms a uniform framework, which extends on the $\pi$-calculus with priorities, its extension by polyadic synchronization $\pi@$, on the stochastic $\pi$-calculus, and its extension by concurrent objects SPiCO. This allows us to compile all existing $\pi$-calculus models of biological systems to the attribute $\pi$-calculus. Furthermore, the attributed $\pi$-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. We have presented and implemented a stochastic simulator for the attributed

$\pi$-calculus, which proves that this general approach is feasible in practice with decent performance.

The imperative $\pi$-calculus is a recent extension of the attributed $\pi$-calculus with a global imperative store [33]. The original motivation was to enable a simpler encoding of dynamic compartments, that does not rely on priorities. Global imperative stores are equally supported by SCCP [32]. There it was noticed, that such stores allow to express n-ary chemical reactions with $n > 2$ reactions and Michaelis-Menten dynamics. This was noticed independently for the imperative $\pi$-calculus in [46]. There a model of the Wnt-pathway in the imperative $\pi$-calculus is developed, that approaches to analyze the cytoplasmic-nuclear shuttling of $\beta$-catenin and the impact of the cell cycle on the Wnt activities.

Quite some questions remain for future research. The first question is on the precise relationship to rule-based formalisms such as the kappa calculus [47, 9] or bigraphs [48]. Another open issue is to develop programming environments for large scale modeling in the attributed or imperative $\pi$-calculus, as for instance by providing object-oriented abstractions with inheritance in the spirit of SPiCO [5].

With respect to language theory, there remain two yet open question. First, whether one can encode $\pi(\lambda)$ into $\pi@$, and second, what is the precise relationship between the imperative $\pi$-calculus and SCCP.

## References


1. Regev, A.: Computational Systems Biology: A Calculus for Biomolecular Knowledge. Tel Aviv University (2003) PhD thesis.
2. Regev, A., Shapiro, E.: Cells as Computation. Nature **419** (2002) 343
3. Ciocchetta, F., Hillston, J.: Bio-PEPA: An Extension of the Process Algebra PEPA for Biochemical Networks. ENTCS **194** (2008) 103–117
4. Dematté, L., Priami, C., Romanel, A.: Modelling and Simulation of Biological Processes in BlenX. SIGMETRICS Performance Evaluation Review **35** (2008) 32–39
5. 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
6. 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
7. 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
8. Chabrier-Rivier, N., Fages, F., Soliman, S.: The Biochemical Abstract Machine BIOCHAM. In: Computational Methods in Systems Biology. (2004) 172–191
9. 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

10. Faeder, J.R., Blinov, M.L., Goldstein, B., Hlavacek, W.S.: Rule-Based Modeling of Biochemical Networks. Complexity **10** (2005) 22–41
11. 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
12. Cardelli, L.: On process rate semantics. Theoretical Computer Science **391** (2008) 190–215
13. Versari, C.: A Core Calculus for a Comparative Analysis of Bio-inspired Calculi. Programming Languages and Systems (2007) 411–425
14. Priami, C.: Stochastic $\pi$-calculus. Computer Journal **6** (1995) 578–589
15. Kuttler, C., Lhoussaine, C., Niehren, J.: A stochastic pi calculus for concurrent objects. In: 1st International Workshop on Probabilistic Automata and Logics. (2006)
16. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Proceedings of BioConcur '04. (2004)
17. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in pi-calculus. Nordic Journal of Computing **10** (2003) 70–98
18. 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
19. Saraswat, V.A., Rinard, M.C.: Concurrent constraint programming. In: ACM SICPLAN-SIGACT Symposium on Principles of Programming Languages, ACM-Press (1990) 232–245
20. Versari, C.: A Core Calculus for the Analysis and Implementation of Biologically Inspired Languages. PhD thesis, University of Bologna (2009)
21. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: BioAmbients: An Abstraction for Biological Compartments. TCS **325** (2004) 141–167
22. 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
23. Kuttler, C., Niehren, J.: Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. Transactions on Computational Systems Biology (2006) 24–55
24. 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)
25. 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
26. 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
27. 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
28. 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

29. 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
30. 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
31. 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
32. Bortolussi, L., Policriti, A.: Modeling biological systems in stochastic concurrent constraint programming. Constraints, an International Journal **13** (2008) 66–90
33. John, M., Lhoussaine, C., Niehren, J.: Dynamic compartments in the imperative pi calculus. In: Computational Methods in Systems Biology, 7th International Conference. Lecture Notes in Computer Science, Springer Verlag (2009)
34. 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
35. Niehren, J.: Uniform confluence in concurrent computation. Journal of Functional Programming **10** (2000) 453–499
36. Huet, G.P.: Confluent reductions: Abstract properties and applications to term rewriting systems. Journal of the ACM **27** (1980) 797–821
37. Kuttler, C., Lhoussaine, C., Nebut, M.: Rule-based modeling of transcriptional attenuation at the tryptophan operon. In: International Winter Simulation Conference. (2009)
38. John, M., Ewald, R., Uhrmacher, A.M.: A Spatial Extension to the Pi Calculus. ENTCS **194** (2008) 133–148
39. Kholodenko, B.N.: Cell-Signalling Dynamics in Time and Space. Nature Reviews Molecular Cell Biology **7** (2006) 165–176
40. Grell, K.G.: Protozoologie. Springer-Verlag (1968)
41. 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
42. 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.
43. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry **81** (1977) 2340–2361
44. 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
45. Pozo, R., Miller, B.: SciMark 2.0. http://math.nist.gov/scimark2/ (2009)
46. 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))
47. Danos, V., Laneve, C.: Formal molecular biology. Theoretical Computer Science **325** (2004) 69–110

48. 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

# A Experiment results

| Levels | Sto/SPiM/Enum | Sto/James/Enum | Attr/James/Enum | Attr/James/Comp |
|---|---|---|---|---|
| 10 | 10.89 (0.00) | 7.22 (0.02) | 10.10 (0.00) | 10.57 (0.02) |
| 20 | 27.83 (0.21) | 18.36 (0.05) | 24.34 (0.13) | 25.90 (0.10) |
| 30 | 47.18 (0.06) | 33.24 (0.27) | 40.47 (0.02) | 41.37 (0.13) |
| 40 | 69.51 (0.07) | 49.53 (0.05) | 57.96 (0.35) | 59.43 (0.07) |
| 50 | 92.60 (0.02) | 72.11 (0.77) | 79.03 (0.15) | 81.15 (0.39) |
| 60 | 125.91 (0.27) | 100.45 (1.41) | 95.56 (1.19) | 100.85 (1.89) |
| 70 | 152.82 (0.05) | 119.66 (1.24) | 120.36 (0.94) | 118.91 (1.42) |
| 80 | 181.75 (0.27) | 145.22 (0.44) | 142.40 (0.63) | 137.72 (0.73) |
| 90 | 209.53 (1.46) | 169.93 (1.19) | 163.10 (1.47) | 161.96 (0.47) |
| 100 | 251.63 (0.93) | 203.15 (0.46) | 177.58 (0.72) | 187.41 (1.12) |

**Table 1.** Runtime of different simulators in $s$ for the unoptimized version of the Euglena model with 10,000 Euglenas and different numbers of depth levels ranging from 10 to 100: Sto = Stochastic Pi Calculus, Attr = Attributed Pi Calculus, SPiM = Stochastic Pi Machine, James = JamesII, Enum = model with enumerated depth levels, Comp = model with depth level as species parameter. Variance in parentheses.

| Levels | Sto/SPiM/Enum | Sto/James/Enum | Attr/James/Enum | Attr/James/Comp |
|---|---|---|---|---|
| 10 | 7.59 (0.01) | 5.72 (0.01) | 8.75 (0.07) | 8.78 (0.01) |
| 20 | 20.28 (0.17) | 13.42 (0.03) | 19.74 (0.14) | 20.13 (0.02) |
| 30 | 34.36 (0.01) | 22.39 (0.02) | 32.25 (1.41) | 31.67 (0.10) |
| 40 | 49.39 (0.51) | 33.61 (0.06) | 44.07 (0.38) | 44.73 (0.02) |
| 50 | 66.84 (0.06) | 45.33 (0.01) | 61.45 (0.35) | 58.91 (0.39) |
| 60 | 86.08 (0.09) | 58.66 (0.13) | 71.31 (0.26) | 72.77 (0.17) |
| 70 | 102.47 (0.05) | 72.84 (1.19) | 85.26 (1.39) | 86.55 (0.38) |
| 80 | 123.92 (0.65) | 88.07 (0.05) | 100.52 (0.87) | 101.19 (0.04) |
| 90 | 145.31 (0.69) | 104.66 (0.74) | 116.89 (1.55) | 113.69 (0.28) |
| 100 | 175.02 (0.25) | 125.90 (1.42) | 131.54 (1.27) | 131.77 (0.28) |

**Table 2.** Runtime of different simulators in $s$ for the optimized version of the Euglena model with 10,000 Euglenas and different numbers of depth levels ranging from 10 to 100: Sto = Stochastic Pi Calculus, Attr = Attributed Pi Calculus, SPiM = Stochastic Pi Machine, James = JamesII, Enum = model with enumerated depth levels, Comp = model with depth level as species parameter. Variance in parentheses.