A Correct Abstract Machine for the Stochastic Pi-calculus

Andrew Phillips and Luca Cardelli

Microsoft Research,
7 JJ Thomson Avenue
Cambridge, UK
{v-anphil,luca}@microsoft.com

Abstract. This paper presents an abstract machine for a variant of the stochastic pi-calculus, in order to correctly model the stochastic simulation of biological processes. The abstract machine is proved sound and complete with respect to the calculus, and then used as the basis for implementing a stochastic simulator. The correctness of the machine helps ensure that the simulator is correctly implemented, giving greater confidence in the simulation results. A graphical representation for the pi-calculus is also presented, as a potential front-end to the simulator.

1 Introduction

Process calculi have been seen traditionally as a theoretical framework for the study of concurrent computation, or as a paradigm for more practical concurrent languages, or as a specification language for software and hardware systems that are coded in more pragmatic ways. Therefore, the direct implementation of process calculi for the purpose of execution has never been a high-priority enterprise. Recently, though, a range of process calculi have been adapted or freshly developed for applications in biology, where highly concurrent processes are the norm. In this application domain, process calculi do not act as a paradigm, but as a direct way to describe systems. Therefore, there is a new interest in correct implementation techniques for process calculi, particularly if a quantitative aspect can be added for the purpose of stochastic execution. This paper focuses on implementation techniques for a variant of the stochastic pi-calculus, in order to correctly model the stochastic simulation of biological processes.

The remainder of the paper is structured as follows. In Sect. 2 a variant of the stochastic pi-calculus is described, along with a corresponding graphical representation. In Sect. 3 an abstract machine for the stochastic pi-calculus is presented, and in Sect. 4 the machine is proved sound and complete with respect to the calculus. An implementation of the stochastic machine is described in Sect. 5, and various simulation results are reported.

2 The Stochastic Pi-Calculus

The variant of the stochastic pi-calculus used in this paper is summarised in Definitions 1, 2 and 3. The calculus is based largely on [1] and [2], but uses a

form of guarded replication presented in [3], which is easier to implement. Each channel x is associated with a corresponding reaction rate given by $rate(x)$, and each reduction is labelled with a reaction rate, as in [2]. A corresponding graphical representation is presented in Definitions 4, 5 and 6.

$P, Q ::= \nu x\, P$	Restriction	(1)		$\Sigma ::= \mathbf{0}$	Null	(5)
$\mid P \mid Q$	Parallel	(2)		$\mid \pi.P + \Sigma$	Action	(6)
$\mid \Sigma$	Summation	(3)		$\pi ::= x\langle n \rangle$	Output	(7)
$\mid\ !\pi.P$	Replication	(4)		$\mid\ x(m)$	Input, $x \neq m$	(8)

Definition 1. *Syntax of* SPi

$$Q \equiv P \xrightarrow{r} P' \equiv Q' \quad \Rightarrow \quad Q \xrightarrow{r} Q' \tag{9}$$

$$P \xrightarrow{r} P' \quad \Rightarrow \quad \nu x\, P \xrightarrow{r} \nu x\, P' \tag{10}$$

$$P \xrightarrow{r} P' \quad \Rightarrow \quad P \mid Q \xrightarrow{r} P' \mid Q \tag{11}$$

$$x\langle n \rangle.P + \Sigma \mid x(m).Q + \Sigma' \xrightarrow{rate(x)} P \mid Q_{\{n/m\}} \tag{12}$$

Definition 2. *Reduction in* SPi

$$P \mid \mathbf{0} \equiv P \tag{13}$$

$$P \mid Q \equiv Q \mid P \tag{14}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \tag{15}$$

$$!\pi.P \equiv \pi.(P \mid !\pi.P) \tag{16}$$

$$\nu x\, \mathbf{0} \equiv \mathbf{0} \tag{17}$$

$$\nu x\, \nu y\, P \equiv \nu y\, \nu x\, P \tag{18}$$

$$\pi.P + \pi'.P' + \Sigma \equiv \pi'.P' + \pi.P + \Sigma \tag{19}$$

$$\Sigma \equiv \Sigma' \Rightarrow \pi.P + \Sigma \equiv \pi.P + \Sigma' \tag{20}$$

$$x \notin fn(P) \Rightarrow \nu x\,(P \mid Q) \equiv P \mid \nu x\, Q \tag{21}$$

$$P \equiv P' \Rightarrow \nu x\, P \equiv \nu x\, P' \tag{22}$$

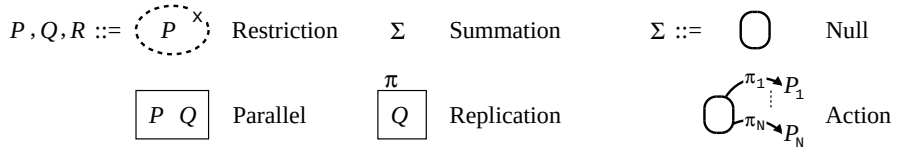
$$P \equiv P' \Rightarrow P \mid Q \equiv P' \mid Q \tag{23}$$

$$P \equiv P' \Rightarrow !\pi.P \equiv !\pi.P' \tag{24}$$

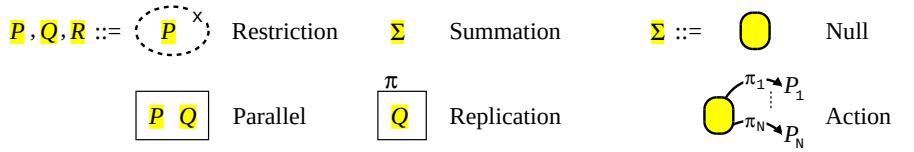
$$P \equiv P' \Rightarrow \pi.P + \Sigma \equiv \pi.P' + \Sigma \tag{25}$$

Definition 3. *Structural congruence in* SPi

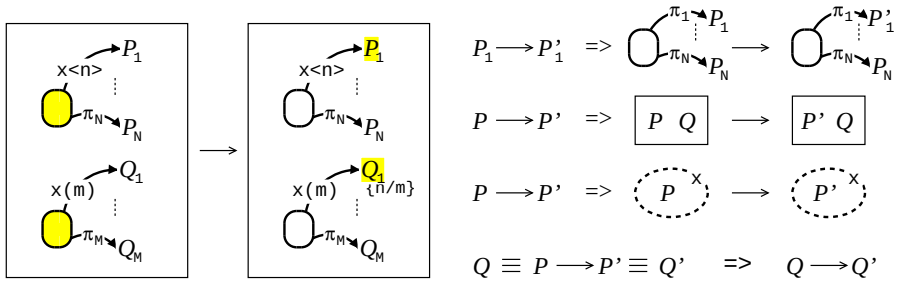
A biological system can be modelled in the stochastic pi-calculus by representing each component of the system as a calculus process, which precisely describes what the component can do. According to Definition 1, the most basic component is a summation Σ, which is a choice between zero or more output $x\langle n \rangle$ or input $x(m)$ actions that the component can perform. Two components P and Q can be combined together using parallel composition $P \mid Q$, and a given component P can contain a restricted reaction channel $\nu x\, P$. In addition, multiple copies of a given component $\pi.P$ can be defined using replication $!\pi.P$. Standard syntactic abbreviations are used, such as writing π for $\pi.\mathbf{0}$ and $\pi.P$



Definition 4. *Graphical Syntax of* SPi



Definition 5. *Graphical Shading in* SPi



Definition 6. *Graphical Reduction in* SPi

for $\pi.P + \mathbf{0}$. A corresponding graphical syntax is given in Definition 4, which is isomorphic to the textual syntax. For convenience, the graphical notation $\boxed{x\langle m\rangle}$ is used to represent a parallel output $x\langle n\rangle$.

Two components in a biological system can react by performing complementary input and output actions on a common reaction channel. According to Definition 2, a summation containing an output $x\langle n\rangle.P$ can react with a parallel summation containing an input $x(m).Q$. The reaction occurs with $rate(x)$, after which the name n is bound to m in process Q (written $Q_{\{n/m\}}$) and processes P and $Q_{\{n/m\}}$ execute in parallel (12). Components can also react in parallel with other components (11), inside a restriction (10), and up to re-ordering of components (9). Components can be re-ordered according to the structural congruence rules given in Definition 3. In particular, these rules allow a replicated input $!x(m).Q$ to spawn a new copy of Q by reacting with an output $x\langle n\rangle.P$. A corresponding definition of graphical reduction is given in Definition 6. The

graphical reduction rules are similar to the textual rules, and make use of a definition of graphical structural congruence that is isomorphic to the textual definition. However, instead of replacing one process with another, graphical reduction uses *shading* to represent the execution of successive processes. This is analogous to highlighting states between successive transitions in a state machine. Shaded processes are described in Definition 5, and are used to represent a process that is currently being executed. Note that the use of shading requires an additional rule to allow reduction inside a summation.

By definition, a graphical process is a tree of nested processes. However, *links* between summations in the tree can be easily encoded in order to represent cyclic behaviour, where a link is a labelled arc with a double-headed arrow. Each summation Σ with one or more inbound links is encoded as $\nu x\,(x\langle\rangle \mid !x().\Sigma)$, and a summation containing a link to Σ with label π is encoded as $\pi.x\langle\rangle + \Sigma'$, where the scope of x is extended accordingly. Note that links can only be defined between two summations that share a common root summation, in order to guarantee mutual exclusion between the source and target of a link.

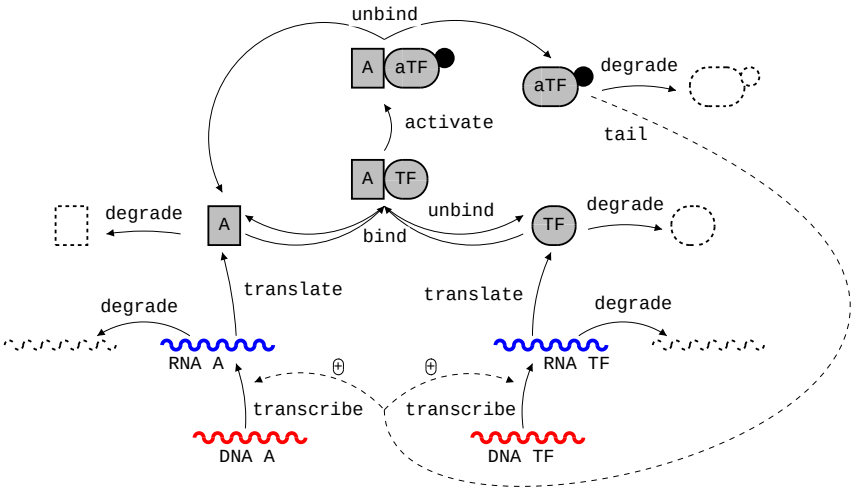
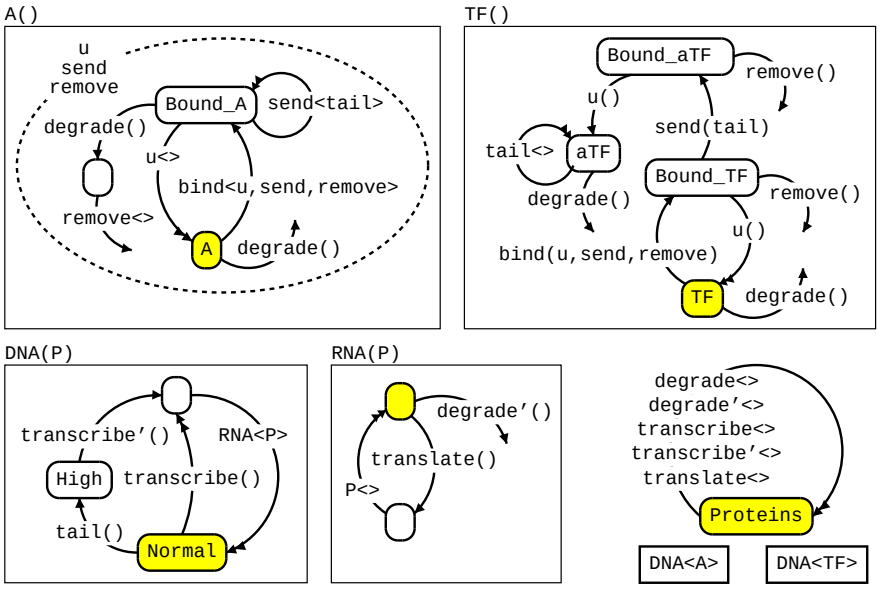


Fig. 1. Interaction Map for Regulating Gene Expression by Positive Feedback [4]

The stochastic pi-calculus can be used to model the regulation of gene expression by positive feedback, based on [4]. According to Fig. 1, DNA for proteins A and TF is transcribed to RNA for A and TF, which is then translated to proteins A and TF, respectively. Protein A can bind to TF and then activate TF by sending its protein tail. Active TF then unbinds from A and uses its newly acquired tail to stimulate the production of A and TF, resulting in a positive feedback loop. A pi-calculus model for this system is given in Fig. 2, together



$$
\begin{aligned}
&(\ !\mathrm{A}().\nu \mathrm{u}\, \nu \mathrm{send}\, \nu \mathrm{remove} \\
&\quad (\ \mathrm{degrade}() + \mathrm{bind}\langle \mathrm{u}, \mathrm{send}, \mathrm{remove}\rangle.\mathrm{Bound}\langle\rangle \\
&\quad \mid\ !\mathrm{Bound}().(\mathrm{u}\langle\rangle.\mathrm{A}\langle\rangle + \mathrm{degrade}().\mathrm{remove}\langle\rangle + \mathrm{send}\langle \mathrm{tail}\rangle.\mathrm{Bound}\langle\rangle)) \\
&\mid\ !\mathrm{RNA}(\mathrm{P}).(\mathrm{degrade}'() + \mathrm{translate}().\mathrm{P}\langle\rangle.\mathrm{RNA}\langle \mathrm{P}\rangle) \\
&\mid\ !\mathrm{DNA}(\mathrm{P}).(\mathrm{transcribe}().\mathrm{RNA}\langle \mathrm{P}\rangle.\mathrm{DNA}\langle \mathrm{P}\rangle + \mathrm{tail}().\mathrm{transcribe}'().\mathrm{RNA}\langle \mathrm{P}\rangle.\mathrm{DNA}\langle \mathrm{P}\rangle) \\
&\mid\ !\mathrm{TF}().(\mathrm{degrade}() + \mathrm{bind}(\mathrm{u}, \mathrm{send}, \mathrm{remove}). \\
&\quad (\mathrm{remove}() + \mathrm{u}().\mathrm{TF}\langle\rangle + \mathrm{send}(\mathrm{tail}).(\mathrm{remove}() + \mathrm{u}().\mathrm{aTF}\langle \mathrm{tail}\rangle))) \\
&\mid\ !\mathrm{aTF}(\mathrm{tail}).(\mathrm{degrade}() + \mathrm{tail}\langle\rangle.\mathrm{aTF}\langle \mathrm{tail}\rangle) \mid \mathrm{DNA}\langle \mathrm{A}\rangle \mid \mathrm{DNA}\langle \mathrm{TF}\rangle \\
&\mid\ !\mathrm{degrade}\langle\rangle \mid\ !\mathrm{degrade}'\langle\rangle \mid\ !\mathrm{transcribe}\langle\rangle \mid\ !\mathrm{transcribe}'\langle\rangle \mid\ !\mathrm{translate}\langle\rangle)
\end{aligned}
$$

Fig. 2. Regulating Gene Expression by Positive Feedback [4]

with a corresponding graphical representation. The graphical model clearly illustrates the behaviour of the DNA, RNA, A and TF proteins in the system, and uses a single *Proteins* process to represent the basic functions of degradation, transcription and translation. For clarity, summations in the model can be annotated with names, but these do not have any semantic meaning. The DNA for a protein P can be transcribed to RNA using the *transcribe* channel, and this RNA can be translated to protein P using the *translate* channel. Protein A can bind with a protein TF by sending private u, *send*, and *remove* channels on the *bind* channel. Once bound, A can send a protein *tail* to TF on the *send*

channel. After TF unbinds from A, it can use its newly acquired *tail* to activate the DNA for A or TF, which will then transcribe RNA at a higher rate using the *transcribe′* channel, resulting in a positive feedback loop.

3 The Stochastic Pi-Machine

3.1 Approach

The Stochastic Pi-Machine (SPiM) is a formal description of how a stochastic pi-calculus process can be executed. The machine is inspired by recent work on abstract machines for process calculi [3,5] and uses a list syntax, which is close to an implementation language. The SPi-Machine executes a given process P by first encoding P into a list of summations with a number of top-level private names. The machine then uses a stochastic selection algorithm based on [6] to choose a particular channel x on which to perform a communication. The procedure is repeated until no more communications are possible. A detailed description of the SPi-Machine is given in the remainder of this section.

3.2 Encoding

In order to execute a given process P, the SPi-Machine first needs to encode P into a suitable machine term. The set of machine terms is denoted by SPiM, and individual machine terms V, U are defined using lists A, B. According to Definition 7, a machine term V is a list with zero or more restricted names, and a list A is either an empty list $[]$ or a list containing one or more summations Σ. Note that summations in SPiM are identical to summations in SPi.

The SPi-Machine encodes a given process P into a machine term using an *encoding function* $(\!|P|\!)$. According to Definition 9, a process P is encoded by adding it to an empty list $[]$ using a *construction operator* $(:)$. The construction operator $P : V$ adds a process P to an arbitrary machine term V, producing an updated machine term as a result. According to Definition 8, if a process P is added to a term $\nu x\, V$ containing a private name x, then P is added to V and the scope of x is extended to the top level, provided x is not known to P (30). Once the scope of each private name has been extended in this way, the process P can be added to the remaining list A. The null process $\mathbf{0}$ is not added to the list (31), and the parallel composition process $P \mid Q$ is split so that each parallel process is added separately (32). The restriction process $\nu y\, P$ is modified by replacing y with a fresh name x, the scope of x is extended to the top level and the process $P_{\{x/y\}}$ is added to the list (33). The replicated action $!\pi.P$ is expanded to a summation consisting of a single action, and the resulting summation is placed at the head of the list (34). Finally, the non-empty summation $\pi.P + \Sigma$ is placed at the head of the list (35).

3.3 Execution

Once a process has been encoded to a machine term using the construction operator, it can then be executed by the machine. In general, a machine term is

$V, U ::= \nu x\, V$	Restriction	(26)	$A, B ::= []$	Empty	(28)
$\mid A$	List	(27)	$\mid \Sigma :: A$	Summation	(29)

Definition 7. *Syntax of* SPiM

$$n \notin fn(P) \quad \Rightarrow \quad P : (\nu x\, V) \triangleq \nu x\, (P : V) \tag{30}$$
$$\mathbf{0} : A \triangleq A \tag{31}$$
$$(P \mid Q) : A \triangleq P : Q : A \tag{32}$$
$$x \notin fn(P, A) \quad \Rightarrow \quad (\nu y\, P) : A \triangleq \nu x\, (P_{\{x/y\}} : A) \tag{33}$$
$$!\pi.P : A \triangleq \pi.(P \mid !\pi.P) :: A \tag{34}$$
$$(\pi.P + \Sigma) : A \triangleq (\pi.P + \Sigma) :: A \tag{35}$$

Definition 8. *Construction in* SPiM

$$(\!|P|\!) \triangleq P : [] \tag{36}$$

Definition 9. *Encoding* SPi *to* SPiM

a list of summations with a number of top-level private names:

$$\nu x_1\, \nu x_2\, ... \nu x_N\, (\Sigma_1 :: \Sigma_2 :: ... :: \Sigma_M :: [])$$

A given term is executed by the machine in steps, according to a labelled reduction relation $\xrightarrow{r}$. The relation $V \xrightarrow{r} V'$ is true if the machine can transform a term V into a term V' with rate r during a single execution step. According to Definition 10, if a term V can reduce to V' with rate r then this reduction can also take place if V contains a private name x (37). This rule allows the machine to execute a list A with an arbitrary number of private names. The machine executes a list A by first choosing the next channel x on which to perform a communication, using the function $Next(A)$. The machine then uses a *selection operator* $\succ$ to choose a summation $x(m).P + \Sigma$ with an input on channel x and another summation $x\langle n\rangle.Q + \Sigma'$ with an output on x. The value n is then sent along channel x and bound to m in process P. The summations Σ and Σ' are discarded, and the processes $P_{\{n/m\}}$ and Q are added to the remainder of the list (38).

The selection operator $\succ$ chooses a particular action from inside a list by first moving a summation to the head of the list and then moving an action to the front of the summation. The relation $A \succ B$ is true if the list A can be re-arranged to match list B. According to Definition 11, a list can be re-arranged by bringing a summation to the head of the list (39) or by bringing an action to the front of a summation (40), where $A@B$ appends list B to list A. Note that

$$V \xrightarrow{r} V' \quad \Rightarrow \quad \nu x\, V \xrightarrow{r} \nu x\, V' \tag{37}$$

$$\left|\begin{array}{l} x = Next(A) \\ \wedge A \succ (x(m).P + \Sigma) :: A' \\ \wedge A' \succ (x\langle n\rangle.Q + \Sigma') :: A'' \end{array}\right. \quad \Rightarrow \quad A \xrightarrow{rate(x)} P_{\{n/m\}} : Q : A'' \tag{38}$$

Definition 10. *Reduction in* SPiM

$$A@\Sigma :: A' \succ \Sigma :: A@A' \tag{39}$$

$$\Sigma :: A \succ (\pi'.P' + \Sigma') :: A \Rightarrow (\pi.P + \Sigma) :: A \succ (\pi'.P' + \pi.P + \Sigma') :: A \tag{40}$$

Definition 11. *Selection in* SPiM

the selection operator only allows a single action inside a single summation to be selected, leaving the remainder of the list unaltered. This prevents the contents of the list from being permuted unnecessarily, resulting in a more efficient machine.

The next reaction channel x and the reaction delay τ are calculated using the algorithm described in Definition 12. The algorithm is based on the Gillespie algorithm [6], which uses a notion of *channel activity* in order to stochastically select the next reaction channel. A similar notion of channel activity is defined for the SPi-Machine, where $\mathrm{Act}_x(A)$ denotes the activity of channel x in list A. The activity corresponds to the number of possible combinations of inputs and outputs on channel x in A, and is defined by:

$$\mathrm{Act}_x(A) = (\mathrm{In}_x(A) * \mathrm{Out}_x(A)) - \mathrm{Mix}_x(A)$$

where $\mathrm{In}_x(A)$ and $\mathrm{Out}_x(A)$ are the number of unguarded *inputs* and *outputs* on channel x in A, respectively, and $\mathrm{Mix}_x(A)$ = the sum of $\mathrm{In}_x(\Sigma_i) \times \mathrm{Out}_x(\Sigma_i)$ for each summation Σ_i in A. The formula takes into account the fact that an input and an output in the same summation cannot interact, by subtracting $\mathrm{Mix}_x(A)$ from the product of the number of inputs and outputs on x. Once the values x and τ have been calculated, the machine increments the reaction time by delay τ and randomly chooses one of the available reactions on x with equal probability, using the selection operator. This is achieved by randomly choosing a number $n \in [1..\mathrm{In}_x(A)]$ and selecting the nth input in A, followed by randomly selecting an output from the remaining list in a similar fashion.

For improved efficiency, the machine can store a list of tuples for each channel x in A, of the form:

$$x, \mathrm{In}_x(A), \mathrm{Out}_x(A), \mathrm{Mix}_x(A), a_x$$

After each reduction has been performed, it is only necessary to update the values for those channels that were affected by the reduction, and then use

1. For all $x \in fn(A)$ calculate $a_x = \mathrm{Act}_x(A) * rate(x)$
2. Store non-zero values of a_x in a list (x_μ, a_μ), where $\mu \in 1...M$.
3. Calculate $a_0 = \sum_{\nu=0}^{M} a_\nu$
4. Generate two random numbers $n_{1,} n_2 \in [0, 1]$ and calculate τ, μ such that:

$$\tau = (1/a_0) \ln(1/n_1)$$

$$\sum_{\nu=1}^{\mu-1} a_\nu < n_2 a_0 \leq \sum_{\nu=1}^{\mu} a_\nu$$

5. $Next(A) = x_\mu$ and $Delay(A) = \tau$.

Definition 12. *Calculating $Next(A)$ and $Delay(A)$ according to Gillespie [6].*

Definition 12 on the updated values to choose the next reaction channel and calculate the delay.

4 Correctness of the Stochastic Pi-Machine

4.1 Approach

The correctness of the SPi-Machine is expressed in terms of five main properties: *safety*, *soundness*, *completeness*, *termination* and *duration*. Safety ensures that the machine does not produce any runtime errors, and Soundness ensures that the machine can only perform valid execution steps. Completeness is a much stronger property, which ensures that the machine can accurately execute all possible behaviours of the calculus. Termination ensures that the machine does not loop forever unnecessarily, and Duration ensures that each reduction in the machine takes the same length of time as the corresponding reduction in the calculus, and vice-versa. An outline of the main proofs is given in Appendix A.

4.2 Safety

Safety ensures that the machine does not produce any runtime errors when executing a given term V. According to Theorem 1, if the machine reduces a term V to V' with rate r, then V' will be a valid machine term.

Theorem 1. $\forall V. V \in \mathrm{SPiM} \wedge V \xrightarrow{r} V' \Rightarrow V' \in \mathrm{SPiM}$

4.3 Soundness

Soundness ensures that each reduction in the machine corresponds to a valid reduction in the calculus. In order to prove the soundness of the machine it is necessary to define a *decoding function* $[\![V]\!]$, which maps a given machine term

V to a corresponding calculus process. According to Definition 13, a term $\nu x\, V$ with a private name x is mapped to the decoded term $[\![V]\!]$ with a private name x (41). The null list is mapped to the null process (42), and a summation at the head of a list is mapped to a summation in parallel with the decoded list (43).

$$[\![\nu x\, V]\!] \triangleq \nu x\, [\![V]\!] \tag{41}$$
$$[\![[]]\!] \triangleq \mathbf{0} \tag{42}$$
$$[\![\Sigma :: A]\!] \triangleq \Sigma \mid [\![A]\!] \tag{43}$$

Definition 13. *Decoding*

Once a decoding from machine terms to calculus processes has been defined in this way, it is possible to state and prove the soundness of the machine. According to Theorem 2, if the machine can reduce a term V to V' with rate r, then the calculus can perform a corresponding reduction with the same rate on the decoding of V.

Theorem 2. $\forall V.V \in \text{SPiM} \wedge V \xrightarrow{r} V' \Rightarrow [\![V]\!] \xrightarrow{r} [\![V']\!]$

4.4 Completeness

Completeness ensures that each reduction in the calculus can be matched by a corresponding reduction in the machine, up to re-ordering of machine terms. In order to prove the completeness of the machine it is necessary to define a structural congruence relation $V \equiv U$, which allows a term V to be re-ordered to match a term U. According to Definition 14, unused private names can be discarded (44), private names can be re-ordered (45) and summations inside a list can also be re-ordered (46). In addition, a list can be re-ordered inside a restriction (47), part of a list can be re-ordered (48) and actions inside a summation can be re-ordered (49).

$$\nu x\, [] \equiv [] \tag{44}$$
$$\nu x\, \nu y\, V \equiv \nu y\, \nu x\, V \tag{45}$$
$$A@B \equiv B@A \tag{46}$$
$$A \equiv A' \Rightarrow \nu x\, A \equiv \nu x\, A' \tag{47}$$
$$A \equiv A' \Rightarrow A@B \equiv A'@B \tag{48}$$
$$\Sigma \equiv \Sigma' \Rightarrow \Sigma :: A \equiv \Sigma' :: A \tag{49}$$

Definition 14. *Structural Congruence in* SPiM

An important property of structural congruence is that congruent terms should be able to perform corresponding reductions that preserve the congruence relation. This property needs to be proved explicitly for the machine, since structural congruence is not used in the definition of reduction. The omission is deliberate, and avoids the need to examine all possible re-orderings of a term in order to perform a reduction. As a result, the efficiency of the machine is significantly improved from $\mathcal{O}(!n)$ to $\mathcal{O}(n)$, where n is the number of summations in the machine. According to Lemma 1, if the machine can reduce a term V to V' with rate r, then it can reduce any term that is congruent to V to a term that is congruent to V', with the same rate.

Lemma 1. $\forall V.V \in \text{SPiM} \wedge U \equiv V \wedge V \xrightarrow{r} V' \Rightarrow \exists U'.U \xrightarrow{r} U' \wedge U' \equiv V'$

Once a structural congruence relation has been defined in this way, it is possible to state and prove the completeness of the machine. According to Theorem 3, if the calculus can reduce a process P to P' with rate r, then the machine can perform a corresponding reduction with the same rate on the encoding of P, up to structural congruence.

Theorem 3. $\forall P.P \in \text{SPi} \wedge P \xrightarrow{r} P' \Rightarrow (\!|P|\!) \xrightarrow{r} \equiv (\!|P'|\!)$

4.5 Termination

Termination ensures that the machine stops executing if there are no more reductions to be performed. This prevents a given simulation from looping forever unnecessarily. According to Theorem 4, if a process P cannot reduce, then the corresponding machine term cannot reduce either.

Theorem 4. $\forall P.P \in \text{SPi} \wedge P \not\longrightarrow \;\Rightarrow\; (\!|P|\!) \not\longrightarrow$

4.6 Duration

The Gillespie algorithm has been proved correct as a means of stochastically selecting a reaction channel [6], and soundness and completeness both ensure that the machine performs each reduction $\xrightarrow{r}$ with the correct rate. However, these properties are not sufficient to express the correctness of the stochastic machine, as illustrated by the following example:

$$P_1 \triangleq (\; x\langle n\rangle.P + x\langle n\rangle.P) \mid x(m).Q$$
$$P_2 \;\triangleq\; x\langle n\rangle.P \mid x(m).Q$$

In this example, both P_1 and P_2 can reduce to the same process $P \mid Q_{\{n/m\}}$, with the same reduction $\xrightarrow{rate(x)}$, yet the reduction is twice as fast in process P_1 as it is in process P_2. This is because two competing actions with exponential distributions of rate r can be viewed as a single action with an exponential distribution of rate $2r$, as explained in [7,2]. In order to distinguish between such

processes, it is necessary to take into account the number of possible interactions on a chosen channel x in a list A, i.e. the activity $\mathrm{Act}_x(A)$ of x in A. This can be achieved by defining a corresponding notion of channel activity $\mathrm{Act}_x(P)$ for calculus processes, and ensuring that channel activity is preserved by decoding and encoding, as described in Propositions 1 and 2 respectively:

$$\mathrm{Act}_x(P) = (\mathrm{In}_x(P) * \mathrm{Out}_x(P)) - \mathrm{Mix}_x(P)$$

Proposition 1. $\forall V \in \mathrm{SPiM}.\mathrm{Act}_x(V) = \mathrm{Act}_x(\llbracket V \rrbracket)$

Proposition 2. $\forall P \in \mathrm{SPi}.\mathrm{Act}_x(P) = \mathrm{Act}_x(\llparenthesis P \rrparenthesis)$

This ensures that reactions in the machine have the same duration as reactions in the calculus, and vice-versa.

5 Implementation

5.1 Approach

A stochastic simulator has been implemented in a functional language (OCaml), based on the abstract machine specification. The simulator, also known as SPiM, consists of a single binary executable, which reads in a source file and simulates reactions for a given duration. The simulation results are stored in a log file as a list of comma-separated values, which can be visualised using third-party software. In addition, a polymorphic type system for channel communication has been implemented based on [5], and a static type-checker accurately reports syntax and type errors before a given source file is executed.

5.2 Data Types

The terms of the machine can be readily implemented as functional datatypes. By definition, a term V is a list of summations with a number of top-level private names $\nu x_1 ... \nu x_N\, A$. In practice, however, the privacy of these top-level names does not need to be implemented explicitly, since each simulator will have its own private address space for storing and manipulating names. Therefore, a machine `term` can be implemented as a list of summations, where a summation is a list of (`action`, `process`) pairs. A `name` is implemented as a (`string`, `float`) pair, where the float corresponds to the reaction rate. The implementation also allows constants and tuples to be sent and received over channels, using `value` and `pattern` data types accordingly.

```
type process =
  Null
| Parallel of process*process
| Restriction of name*process
| Replication of (action*process)
| Summation of (action*process) list
```

```
type term =
  ((action*process) list) list

type action =
  Input of value*pattern
| Output of value*value
```

5.3 Encoding

The simulator executes a pi-calculus source file, written in a standard ascii syntax, by first parsing the file to produce a corresponding `process`. The process is then encoded to a `term` using a `cons` function to add the process to an empty list. The `cons` function is a direct implementation of the construction operator (:). In particular, a restriction process `Restriction(n,p)` is added to a term by a generating a fresh name based on `n`, substituting `n` with this fresh name in `p` using the `bind` function, and adding the resulting process to the list. The function `fresh(n : name)` uses a naming convention to guarantee that each generated name is globally fresh for the duration of the simulation. This is achieved by appending a global counter (such as a time stamp) to the name using a reserved suffix, such as $\sim$. If the generated name is globally fresh then the restriction can be brought to the top level according to Definition 8, which means that it does not need to be explicitly represented by the machine.

```
let rec cons (p:process) (l:term) = match p with
    Null -> l
  | Parallel(p,p') -> cons p (cons p' l)
  | Restriction(n,p) -> cons (bind (fresh n) n p) l
  | Replication(a,p') -> [a,Parallel(p',p)]::l
  | Summation(s) -> s::l
```

5.4 Execution

The implementation uses the function `reduce(l : term)` to perform a single execution step on a term `l`. The function `gillespie(l : term)` returns a channel inside the list that is able to communicate, along with the time elapsed. Both of these values are calculated according to the stochastic algorithm in Definition 12. The function `select(a : action)(l : term)` randomly chooses an action from inside the list `l` that matches the action `a`. The match is performed based solely on the type of the action and the channel, so arbitrary constants are used for the input pattern `m0` and output value `v0` in the arguments.

```
let reduce (l:term) =
  let (x:value),(t:float) = gillespie l
  in match select (Input(x,m0)) l with
      Some((Input(x,m),p),s,l) -> (
        match select (Output(x,v0)) l with
            Some((Output(x',v),p'),s',l) ->
              if x==x'
              then Some(t,cons (bind v m p) (cons p' l))
              else None
          | _ -> None )
    | _ -> None
```

The simulator repeatedly applies the `reduce` function to the list until no more reductions are possible, at which point the simulation terminates. After each

reduction step, the machine logs the time elapsed and the quantity of top-level inputs and outputs on each channel. The results are stored in a file as a comma separated list, which can be visualised using third party software. In order to improve the efficiency of the machine, the `reduce` function is be modified to keep track of the total number of inputs, outputs and mixed sums as described in Sect. 3.

5.5 Simulation Results

The implementation has been used to simulate the regulation of gene expression by positive feedback described in Sect. 2. As shown in Fig. 3 and in accordance with [4], higher levels of Protein A are observed in the presence of the TF gene and lower levels are observed when the TF gene is disabled.

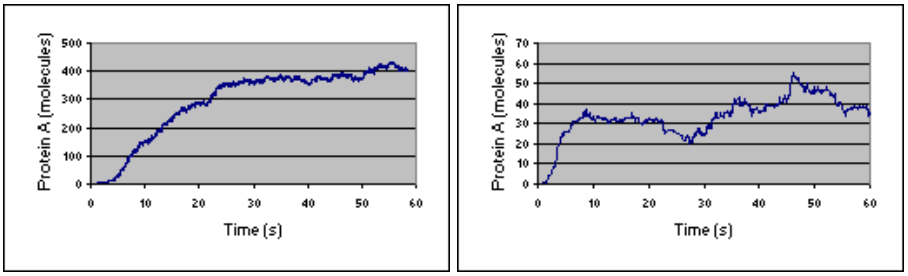


Fig. 3. Protein A molecules v.s. time in presence (left) and absence (right) of TF

The implementation has also been used to simulate a wide variety of chemical reactions and biological systems, including enzymatic reactions, a circadian clock, and a model of the cell cycle control in eucaryotes [8]. In addition, many of the benchmark examples that were used to validate the Gillespie algorithm [6] have been modelled as pi-calculus processes and correctly simulated in SPiM. Details of simulation results are available from [9], together with a stable release of SPiM for Windows, Linux and Mac OS X.

6 Related Work

The BioSPI system [4] is an existing implementation of a biochemical variant of the stochastic π-calculus. The system executes a process by compiling it to an FCP procedure, which is then executed by the FCP Logix platform [10]. Channel data structures are used to maintain stochastic information and synchronize send and receive requests, in accordance with the Gillespie algorithm.

Unlike SPiM, there is no formal definition of an abstract machine and the implementation is specific to FCP Logix. In contrast, SPiM is based on a general purpose abstract machine that is not tied to a particular platform. This allows for greater flexibility of implementation, and also results in increased efficiency. In some cases the SPiM simulator performs up to 7 times faster than BioSPI, such as for the circadian clock model described in [9]. In addition, according to [4] BioSPI calculates the activity of a channel x in P by $\mathrm{In}_x(P) * \mathrm{Out}_x(P)$. This assumes that there can never be both an input and an output on the same channel in the same summation. A special rate law is defined for homodimerization reactions of the form $\Sigma + x\langle n\rangle.P + x(m).Q \mid \Sigma' + x\langle n\rangle.P' + x(m).Q'$, but this does not account for arbitrary combinations of input and output. Furthermore, due to scope extrusion, it is not clear whether such arbitrary combinations can be avoided statically, without limiting the expressiveness of the calculus. SPiM addresses this issue by giving a more general definition of channel activity, which accounts for mixed inputs and outputs using $\mathrm{Mix}_x(P)$. Homodimerization reactions are also included in this definition, provided the reaction rate of the corresponding channel x is halved in the model. It is also worth noting that, unlike SPiM, the current BioSPI system does not implement a type system for channel communication. The BioSPI system has also been extended to handle membrane interactions [11].

Another implementation of the stochastic pi-calculus is the StoPi simulator described in [2], where fully general sums are supported. A stochastic calculus is formally defined, and the implementation architecture is also described in detail, but the paper does not include an abstract machine that maps readily to program code, or prove the correctness of the machine. An alternative stochastic simulator is the PEPA system [12], which can also be used to simulate biological processes. However, PEPA does not include a notion of name-passing, which is important for modelling chemical bonding and is one of the main features of the pi-calculus.

7 Conclusion

We have described an abstract machine for a basic stochastic process calculus, and verified some of its properties. We hope that this will form a framework on which to design and build implementations of richer stochastic process calculi, and possibly of very different, biologically inspired calculi that share a stochastic architecture. We also plan to incorporate a graphical front-end to the current simulator, based on the graphical representation of the pi-calculus presented in this paper. It can be argued that improving the reliability of simulations and working toward a more user-friendly interface are two key objectives in the design and implementation of next-generation simulators for Systems Biologists.

References

1. Milner, R.: Communicating and Mobile Systems: the π-Calculus. (1999)

2. Bloch, A., Haagensen, B., Hoyer, M.K., Knudsen, S.U.: (The StoPi-calculus and Simulator) Article and runtime system available from http://www.cs.auc.dk/~steffen/dat4/stopi/.
3. Phillips, A., Yoshida, N., Eisenbach, S.: A distributed abstract machine for boxed ambient calculi. In: ESOP'04. LNCS, Springer-Verlag (2004)
4. 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)
5. Turner, D.N.: The Polymorphic Pi-Calculus: Theory and Implementation. PhD thesis (1996) CST-126-96 (also published as ECS-LFCS-96-345).
6. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81** (1977) 2340–2361
7. Priami, C.: Stochastic π-calculus. The Computer Journal **38** (1995) 578–589 Proceedings of PAPM'95.
8. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: a biospi model. In: BioConcur'03, ENTCS (2003)
9. Phillips, A.: (The Stochastic Pi-Machine) Available from http://www.doc.ic.ac.uk/~anp/spim/.
10. Silverman, W., Hirsch, M., Houri, A., Shapiro, E.: The logix system user manual, version 1.21. In Shapiro, E., ed.: Concurrent Prolog: Collected Papers (Volume II). MIT Press, London (1987) 46–77
11. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: Bioambients: An abstraction for biological compartments. In: Theoretical Computer Science, Special Issue on Computational Methods in Systems Biology, (Elsevier) To Appear.
12. Gilmore, S., Hillston, J.: The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In: Proceedings of MTTCPE. Number 794 in LNCS, Vienna, Springer-Verlag (1994) 353–368

A Proof Outline

The proofs in this appendix make use of the following notations:

$\overset{\text{IH}}{\Rightarrow} \triangleq$ By Induction Hypothesis

$\overset{(1)}{\Rightarrow} \triangleq$ By Rule (1)

$\overset{\text{lem}\,1}{\Rightarrow} \triangleq$ By Lemma 1

Lemma 2. *(Selection Safety)* $\forall A.A \in \text{SPiM} \wedge A \succ B \Rightarrow B \in \text{SPiM}$

Proof. By induction on Definition 11 of selection in SPiM. □

Lemma 3. *(Construction Safety)* $\forall V.\forall P.V \in \text{SPiM} \wedge P \in \text{SPi} \Rightarrow P : V \in \text{SPiM}$

Proof. By induction on Definition 8 of construction in SPiM:

(**30**) $x \notin fn(P)$
$\wedge(\nu x\, V) \in \text{SPiM} \wedge P \in \text{SPi}$
$\overset{(26)}{\Rightarrow}$ $V \in \text{SPiM}$
$\overset{\text{IH}}{\Rightarrow}$ $P : V \in \text{SPiM}$
$\overset{(26)}{\Rightarrow}$ $\nu x\,(P : V) \in \text{SPiM}$
$\overset{(30)}{\Rightarrow}$ $P : (\nu x\, V) \in \text{SPiM}$

(**31**) $A \in \text{SPiM} \wedge \mathbf{0} \in \text{SPi}$
$\overset{(31)}{\Rightarrow}$ $\mathbf{0} : A \in \text{SPiM}$

(**32**) $A \in \text{SPiM} \wedge (P \mid Q) \in \text{SPi}$
$\overset{(2)}{\Rightarrow}$ $P \in \text{SPi} \wedge Q \in \text{SPi}$
$\overset{\text{IH}}{\Rightarrow}$ $Q : A \in \text{SPiM}$
$\overset{\text{IH}}{\Rightarrow}$ $P : (Q : A) \in \text{SPiM}$
$\overset{(32)}{\Rightarrow}$ $(P \mid Q) : A \in \text{SPiM}$

(**33**) $x \notin fn(P, A)$
$\wedge A \in \text{SPiM} \wedge (\nu y\, P) \in \text{SPi}$
$\overset{(1)}{\Rightarrow}$ $P \in \text{SPi}$
$\Rightarrow$ $P_{\{x/y\}} \in \text{SPi}$
$\overset{\text{IH}}{\Rightarrow}$ $P_{\{x/y\}} : A \in \text{SPiM}$
$\overset{(26)}{\Rightarrow}$ $\nu x\,(P_{\{x/y\}} : A) \in \text{SPiM}$
$\overset{(33)}{\Rightarrow}$ $(\nu y\, P) : A \in \text{SPiM}$

(**34**) $A \in \text{SPiM} \wedge !\pi.P \in \text{SPi}$
$\overset{(16)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) \in \text{SPi}$
$\overset{(29)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) :: A \in \text{SPiM}$
$\overset{(34)}{\Rightarrow}$ $!\pi.P : A \in \text{SPiM}$

(**35**) $A \in \text{SPiM} \wedge (\pi.P + \Sigma) \in \text{SPi}$
$\overset{(29)}{\Rightarrow}$ $(\pi.P + \Sigma) :: A \in \text{SPiM}$
$\overset{(35)}{\Rightarrow}$ $(\pi.P + \Sigma) : A \in \text{SPiM}$ □

Theorem 5. *(Reduction Safety)* $\forall V.V \in \text{SPiM} \wedge V \xrightarrow{r} V' \Rightarrow V' \in \text{SPiM}$

Proof. By Lemma 2, Lemma 3 and by induction on Definition 10 of reduction in SPiM:

(**38**) $A \in \text{SPiM}$
$\wedge A \succ (x(m).P + \Sigma) :: A'$
$\wedge A' \succ (x\langle n\rangle.Q + \Sigma') :: A''$
$\wedge A \xrightarrow{r} P_{\{n/m\}} : Q : A''$
$\overset{\text{lem 2}}{\Rightarrow}$ $(x(m).P + \Sigma) :: A' \in \text{SPiM}$
$\overset{(29)}{\Rightarrow}$ $A' \in \text{SPiM} \wedge P \in \text{SPi}$
$\overset{\text{lem 2}}{\Rightarrow}$ $(x\langle n\rangle.Q + \Sigma') :: A'' \in \text{SPiM}$
$\overset{(29)}{\Rightarrow}$ $A'' \in \text{SPiM} \wedge Q \in \text{SPi}$
$\overset{\text{lem 3}}{\Rightarrow}$ $Q : A'' \in \text{SPiM}$
$\overset{\text{lem 3}}{\Rightarrow}$ $P_{\{n/m\}} : (Q : A'') \in \text{SPiM}$

(**37**) $\nu x\, V \in \text{SPiM}$
$\wedge V \xrightarrow{r} V' \wedge \nu x\, V \xrightarrow{r} \nu x\, V'$
$\overset{(26)}{\Rightarrow}$ $V \in \text{SPiM}$
$\overset{\text{IH}}{\Rightarrow}$ $V' \in \text{SPiM}$
$\overset{(26)}{\Rightarrow}$ $\nu x\, V' \in \text{SPiM}$ □

Lemma 4. *(Decoding Soundness)* $\forall V.V \in \text{SPiM} \Rightarrow [\![V]\!] \in \text{SPi}$

Proof. By induction on Definition 13 of decoding in SPiM. □

Lemma 5. *(Selection Soundness)* $\forall A.A \in \text{SPiM} \wedge A \succ B \Rightarrow [\![A]\!] \equiv [\![B]\!]$

Proof. By induction on Definition 11 of selection in SPiM. □

Lemma 6. *(Construction Soundness)* $\forall V.\forall P.V \in \text{SPiM} \wedge P \in \text{SPi} \Rightarrow [\![P : V]\!] \equiv P \mid [\![V]\!]$

Proof. By induction on Definition 8 of construction in SPiM:

(**30**) $x \notin fn(P)$
$\wedge \nu x\, V \in \text{SPiM} \wedge P \in \text{SPi}$
$\overset{(26)}{\Rightarrow}$ $V \in \text{SPiM}$
$\overset{\text{IH}}{\Rightarrow}$ $[\![P : V]\!] \equiv P \mid [\![V]\!]$
$\Rightarrow$ $\nu x\, [\![P : V]\!] \equiv \nu x\, (P \mid [\![V]\!])$
$\overset{(21)}{\Rightarrow}$ $\nu x\, [\![P : V]\!] \equiv P \mid \nu x\, [\![V]\!]$
$\overset{(41)}{\Rightarrow}$ $[\![\nu x\, (P : V)]\!] \equiv P \mid [\![\nu x\, V]\!]$
$\overset{(30)}{\Rightarrow}$ $[\![P : (\nu x\, V)]\!] \equiv P \mid [\![\nu x\, V]\!]$

(**31**) $A \in \text{SPiM} \wedge \mathbf{0} \in \text{SPi}$
$\overset{(13)}{\Rightarrow}$ $[\![A]\!] \equiv \mathbf{0} \mid [\![A]\!]$
$\overset{(31)}{\Rightarrow}$ $[\![\mathbf{0} : A]\!] \equiv \mathbf{0} \mid [\![A]\!]$

(**32**) $A \in \text{SPiM} \wedge (P \mid Q) \in \text{SPi}$
$\overset{(2)}{\Rightarrow}$ $P \in \text{SPi} \wedge Q \in \text{SPi}$
$\overset{\text{IH}}{\Rightarrow}$ $[\![Q : A]\!] \equiv Q \mid [\![A]\!]$
$\overset{\text{IH}}{\Rightarrow}$ $[\![P : (Q : A)]\!] \equiv P \mid (Q \mid [\![A]\!])$
$\overset{(32)}{\Rightarrow}$ $[\![(P \mid Q) : A]\!] \equiv P \mid (Q \mid [\![A]\!])$
$\overset{(15)}{\Rightarrow}$ $[\![(P \mid Q) : A]\!] \equiv (P \mid Q) \mid [\![A]\!]$

(**33**) $x \notin fn(P, A)$
$\wedge A \in \text{SPiM} \wedge \nu y\, P \in \text{SPi}$
$\overset{(1)}{\Rightarrow}$ $P \in \text{SPi}$
$\Rightarrow$ $P_{\{x/y\}} \in \text{SPi}$
$\overset{\text{IH}}{\Rightarrow}$ $[\![P_{\{x/y\}} : A]\!] \equiv P_{\{x/y\}} \mid [\![A]\!]$
$\Rightarrow$ $\nu n\, [\![P_{\{x/y\}} : A]\!] \equiv \nu x\, (P_{\{x/y\}} \mid [\![A]\!])$
$\overset{(21)}{\Rightarrow}$ $\nu n\, [\![P_{\{x/y\}} : A]\!] \equiv \nu y\, P \mid [\![A]\!]$
$\overset{(41)}{\Rightarrow}$ $[\![\nu x\, (P_{\{x/y\}} : A)]\!] \equiv \nu y\, P \mid [\![A]\!]$
$\overset{(33)}{\Rightarrow}$ $[\![(\nu y\, P) : A]\!] \equiv \nu y\, P \mid [\![A]\!]$

(**34**) $A \in \text{SPiM} \wedge !\pi.P \in \text{SPi}$
$\overset{(16)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) \in \text{SPi}$
$\overset{(43)}{\Rightarrow}$ $[\![\pi.(P \mid !\pi.P) :: A]\!] = \pi.(P \mid !\pi.P) \mid [\![A]\!]$
$\overset{(16)}{\Rightarrow}$ $[\![\pi.(P \mid !\pi.P) :: A]\!] \equiv !\pi.P \mid [\![A]\!]$
$\overset{(34)}{\Rightarrow}$ $[\![!\pi.P : A]\!] \equiv !\pi.P \mid [\![A]\!]$

(**35**) $A \in \text{SPiM} \wedge \pi.P + \Sigma \in \text{SPi}$
$\overset{(29)}{\Rightarrow}$ $(\pi.P + \Sigma) :: A \in \text{SPiM}$
$\overset{(43)}{\Rightarrow}$ $[\![(\pi.P + \Sigma) :: A]\!] = (\pi.P + \Sigma) \mid [\![A]\!]$
$\overset{(35)}{\Rightarrow}$ $[\![(\pi.P + \Sigma) : A]\!] \equiv (\pi.P + \Sigma) \mid [\![A]\!]$

□

Lemma 7. *(Reduction Soundness)* $\forall V.V \in \text{SPiM} \wedge V \xrightarrow{r} V' \Rightarrow [\![V]\!] \xrightarrow{r} [\![V']\!]$

Proof. By Lemma 5, Lemma 6 and by induction on Definition 10 of reduction in SPiM:

$$
\begin{array}{ll}
(\mathbf{38}) & A \in \text{SPiM} \\
& \wedge A \succ (x(m).P + \Sigma) :: A' \\
& \wedge A' \succ (x\langle n\rangle.Q + \Sigma') :: A'' \\
& \wedge A \xrightarrow{r} P_{\{n/m\}} : Q : A'' \\
\overset{\text{lem } 5}{\Rightarrow} & [\![A]\!] \equiv [\![(x(m).P + \Sigma) :: A']\!] \\
\overset{\text{lem } 5}{\Rightarrow} & [\![A']\!] \equiv [\![(x\langle n\rangle.Q + \Sigma') :: A'']\!] \\
\overset{(43)}{\Rightarrow} & [\![A]\!] \equiv (x(m).P + \Sigma) \mid [\![A']\!] \\
\overset{(43)}{\Rightarrow} & [\![A']\!] \equiv (x\langle n\rangle.Q + \Sigma') \mid [\![A'']\!] \\
\Rightarrow & [\![A]\!] \equiv (x(m).P + \Sigma) \mid (x\langle n\rangle.Q + \Sigma') \mid [\![A'']\!] \\
\overset{(12,11,9)}{\Rightarrow} & [\![A]\!] \xrightarrow{r} P_{\{n/m\}} \mid Q \mid [\![A'']\!] \\
\overset{\text{lem } 6}{\Rightarrow} & [\![A]\!] \xrightarrow{r} P_{\{n/m\}} \mid [\![Q : A'']\!] \\
\overset{\text{lem } 6}{\Rightarrow} & [\![A]\!] \xrightarrow{r} [\![P_{\{n/m\}} : Q : A'']\!]
\end{array}
$$

$$
\begin{array}{ll}
(\mathbf{37}) & \nu x\, V \in \text{SPiM} \\
& \wedge V \xrightarrow{r} V' \\
& \wedge \nu x\, V \xrightarrow{r} \nu x\, V' \\
\overset{\text{IH}}{\Rightarrow} & [\![V]\!] \xrightarrow{r} [\![V']\!] \\
\overset{(10)}{\Rightarrow} & \nu x\, [\![V]\!] \xrightarrow{r} \nu x\, [\![V']\!] \\
\overset{(41)}{\Rightarrow} & [\![\nu x\, V]\!] \xrightarrow{r} [\![\nu x\, V']\!]
\end{array}
$$

□

Lemma 8. *(Structural Reduction)* $\forall V.V \in \text{SPiM} \wedge U \equiv V \wedge V \xrightarrow{r} V' \Rightarrow U \xrightarrow{r} \equiv V'$

Proof. By induction on Definition 14 of structural congruence in SPiM. □

Theorem 6. *(Completeness)* $\forall P.P \in \text{SPi} \wedge P \xrightarrow{r} P' \Rightarrow (\!|P \mid R|\!) \xrightarrow{r} \equiv (\!|P' \mid R|\!)$.

Proof. By Lemma 8, Lemma 9 and by induction on Definition 2 of reduction in SPi:

$$
\begin{array}{ll}
(\mathbf{10}) & m \notin fn(P, R) \wedge P \xrightarrow{r} P' \\
& \wedge (\nu n\, P) \xrightarrow{r} (\nu n\, P') \\
\Rightarrow & P_{\{m/n\}} \xrightarrow{r} P'_{\{m/n\}} \\
\overset{\text{IH}}{\Rightarrow} & (\!|P_{\{m/n\}} \mid R|\!) \xrightarrow{r} \equiv (\!|P'_{\{m/n\}} \mid R|\!) \\
\overset{(37)}{\Rightarrow} & \nu m\, (\!|P_{\{m/n\}} \mid R|\!) \xrightarrow{r} \equiv \nu m\, (\!|P'_{\{m/n\}} \mid R|\!) \\
\overset{(33)}{\Rightarrow} & (\nu n\, P) : R : [] \xrightarrow{r} \equiv (\nu n\, P') : R : [] \\
\overset{(32)}{\Rightarrow} & ((\nu n\, P) \mid R) : [] \xrightarrow{r} \equiv ((\nu n\, P') \mid R) : []
\end{array}
$$

$$
\begin{array}{ll}
(\mathbf{9}) & Q \xrightarrow{r} Q' \wedge Q \equiv P \xrightarrow{r} P' \equiv Q' \\
\overset{\text{IH}}{\Rightarrow} & (\!|P|\!) \xrightarrow{r} \equiv (\!|P'|\!) \\
\overset{\text{lem } 9}{\Rightarrow} & (\!|P|\!) \equiv (\!|Q|\!) \wedge (\!|P'|\!) \equiv (\!|Q'|\!) \\
\overset{\text{lem } 8}{\Rightarrow} & (\!|Q|\!) \xrightarrow{r} \equiv (\!|Q'|\!)
\end{array}
$$

$$
\begin{array}{ll}
(\mathbf{11}) & P \xrightarrow{r} P' \wedge (P \mid Q) \xrightarrow{r} (P' \mid Q) \\
\overset{\text{IH}}{\Rightarrow} & (\!|P \mid Q|\!) \xrightarrow{r} (\!|P' \mid Q|\!)
\end{array}
$$

$$
\begin{array}{ll}
(\mathbf{12}) & (x\langle n\rangle.P + \Sigma) \mid (x(m).Q + \Sigma') \xrightarrow{r} P \mid Q_{\{n/m\}} \\
\overset{(37,38)}{\Rightarrow} & \nu\tilde{z}\, ((x\langle n\rangle.P + \Sigma) :: (x(m).Q + \Sigma') :: C) \xrightarrow{r} \nu\tilde{z}\, (P : Q_{\{n/m\}} : C) \\
\overset{\text{def } 8}{\Rightarrow} & (x\langle n\rangle.P + \Sigma) : (x(m).Q + \Sigma') : R : [] \xrightarrow{r} P : Q_{\{n/m\}} : R : [] \\
\overset{(32)}{\Rightarrow} & ((x\langle n\rangle.P + \Sigma) \mid (x(m).Q + \Sigma') \mid R) : [] \xrightarrow{r} (P \mid Q_{\{n/m\}} \mid R) : []
\end{array}
$$

□

Lemma 9. *(Structural Completeness)* $P \equiv Q \Rightarrow [\![P]\!] \equiv [\![Q]\!]$

Proof. By induction on Definition 3 of structural congruence in SPi, where $[\![P]\!] = P : []$

(13) $\mathbf{0} \mid P \equiv P$
$\overset{(31)}{\Rightarrow}$ $\mathbf{0} : (P : []) \equiv P : []$
$\overset{(32)}{\Rightarrow}$ $(\mathbf{0} \mid P) : [] \equiv P : []$

(14) $P \mid Q \equiv Q \mid P$
$\overset{(45,46,47)}{\Rightarrow}$ $\nu\tilde{x}\,\nu\tilde{y}\,A@B \equiv \nu\tilde{y}\,\nu\tilde{x}\,B@A$
$\overset{\text{def } 8}{\Rightarrow}$ $P : Q : [] \equiv Q : P : []$
$\overset{(32)}{\Rightarrow}$ $(P \mid Q) : [] \equiv (Q \mid P) : []$

(15) $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
$\Rightarrow$ $P : Q : R : [] = P : Q : R : []$
$\overset{(32)}{\Rightarrow}$ $P : (Q \mid R) : [] \equiv (P \mid Q) : R : []$
$\overset{(32)}{\Rightarrow}$ $(P \mid (Q \mid R)) : [] \equiv ((P \mid Q) \mid R) : []$

(16) $!\pi.P \equiv \pi.(P \mid !\pi.P)$
$\overset{(34)}{\Rightarrow}$ $!\pi.P : [] = \pi.(P \mid !\pi.P) :: []$
$\overset{(35)}{\Rightarrow}$ $!\pi.P : [] = \pi.(P \mid !\pi.P) : []$

(17) $\nu x\,\mathbf{0} \equiv \mathbf{0}$
$\overset{(44)}{\Rightarrow}$ $\nu x\,[] \equiv []$
$\overset{(31)}{\Rightarrow}$ $\nu x\,(\mathbf{0} : []) \equiv \mathbf{0} : []$
$\overset{(33)}{\Rightarrow}$ $(\nu x\,\mathbf{0}) : [] \equiv \mathbf{0} : []$

(18) $\nu x\,\nu y\,P \equiv \nu y\,\nu x\,P$
$\overset{(45)}{\Rightarrow}$ $\nu x\,\nu y\,(P : []) \equiv \nu y\,\nu x\,(P : [])$
$\overset{(33)}{\Rightarrow}$ $(\nu x\,\nu y\,P) : [] \equiv (\nu y\,\nu x\,P) : []$

(19, 20, 25) $\Sigma \equiv \Sigma'$
$\overset{(49)}{\Rightarrow}$ $\Sigma :: [] \equiv \Sigma' :: []$
$\overset{(35)}{\Rightarrow}$ $\Sigma : [] \equiv \Sigma' : []$

(21) $x \notin fn(P) \wedge \nu x\,(P \mid Q) \equiv P \mid \nu x\,Q$
$\overset{(32)}{\Rightarrow}$ $\nu x\,((P \mid Q) : []) \equiv \nu x\,(P : Q : [])$
$\overset{(30)}{\Rightarrow}$ $\nu x\,((P \mid Q) : []) \equiv P : \nu x\,(Q : [])$
$\overset{(33)}{\Rightarrow}$ $(\nu x\,(P \mid Q)) : [] \equiv P : (\nu x\,Q) : []$
$\overset{(32)}{\Rightarrow}$ $(\nu x\,(P \mid Q)) : [] \equiv (P \mid \nu x\,Q) : []$

(22) $P \equiv P' \wedge \nu x\,P \equiv \nu x\,P'$
$\overset{\text{IH}}{\Rightarrow}$ $P : [] \equiv P' : []$
$\overset{(47)}{\Rightarrow}$ $\nu x\,(P : []) \equiv \nu x\,(P : [])$
$\overset{(33)}{\Rightarrow}$ $(\nu x\,P) : [] \equiv (\nu x\,P) : []$

(23) $P \equiv P' \wedge A \equiv A' \wedge P \mid Q \equiv P' \mid Q$
$\overset{\text{IH}}{\Rightarrow}$ $P : [] \equiv P' : []$
$\overset{(46,47)}{\Rightarrow}$ $\nu\tilde{x}\,\nu\tilde{y}\,A@B \equiv \nu\tilde{x}\,\nu\tilde{y}\,A'@B$
$\overset{\text{def } 8}{\Rightarrow}$ $P : Q : [] \equiv P' : Q : []$
$\overset{(32)}{\Rightarrow}$ $(P \mid Q) : [] \equiv (P' \mid Q) : []$

(24) $P \equiv P' \wedge !\pi.P \equiv !\pi.P'$
$\overset{(25,23)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) \equiv \pi.(P' \mid !\pi.P')$
$\overset{(49)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) :: [] \equiv \pi.(P' \mid !\pi.P') :: []$
$\overset{(35)}{\Rightarrow}$ $\pi.(P \mid !\pi.P) : [] \equiv \pi.(P' \mid !\pi.P') : []$

□