

# Stochastic Simulation of Multiple Process Calculi for Biology

Matthew R. Lakin<sup>a,b</sup>, Loïc Paulevé<sup>c</sup>, Andrew Phillips<sup>a,\*</sup>

<sup>a</sup>*Microsoft Research, Cambridge, United Kingdom*

<sup>b</sup>*Department of Computer Science, University of New Mexico, Albuquerque, NM, USA*

<sup>c</sup>*LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597*

*1 rue de la Noë - B.P. 92101 - 44321 Nantes Cedex 3, France*

---

## Abstract

Numerous programming languages based on process calculi have been developed for biological modelling, many of which can generate potentially unbounded numbers of molecular species and reactions. As a result, such languages cannot rely on standard reaction-based simulation methods, and are generally implemented using custom stochastic simulation algorithms. As an alternative, this paper proposes a generic abstract machine that can be instantiated to simulate a range of process calculi using a range of simulation methods. The abstract machine functions as a just-in-time compiler, which dynamically updates the set of possible reactions and chooses the next reaction in an iterative cycle. We instantiate the generic abstract machine with two Markovian simulation methods and provide encodings for four process calculi: the agent-based pi-calculus, the compartment-based bioambient calculus, the rule-based kappa calculus and the domain-specific DNA strand displacement calculus. We present a generic method for proving that the encoding of an arbitrary process calculus into the abstract machine is correct, and we use this method to prove the correctness of all four calculus encodings. Finally, we demonstrate how the generic abstract machine can be used to simulate heterogeneous models in which discrete communicating sub-models are written using different domain-specific languages and then simulated together. Our approach forms the basis of a multi-language environment for the simulation of heterogeneous biological models.

*Keywords:* generic abstract machine, stochastic simulation, pi-calculus, bioambient calculus, kappa calculus, correctness, implementation

---

## 1. Introduction

Biological systems typically involve large numbers of components with complex, highly parallel interactions and intrinsic stochasticity. To model this complexity, numerous programming languages based on process calculi have been developed, including variants of the stochastic pi-calculus [23, 27, 20], BlenX [8],

---

\*Corresponding author

*Email addresses:* mlakin@cs.unm.edu (Matthew R. Lakin),  
loic.pauleve@irccyn.ec-nantes.fr (Loïc Paulevé), andrew.phillips@microsoft.com  
(Andrew Phillips)

the kappa calculus [5], LBS [18], variants of the bioambient calculus [26, 19] and the DNA strand displacement calculus [21, 15]. Many of these calculi are expressive enough to generate potentially unbounded numbers of molecular species and reactions. As a result, they cannot rely on standard reaction-based simulation methods such as [12, 10], which require fixed numbers of species and reactions. Instead, a custom simulation algorithm is typically developed for each calculus. The choice of algorithm depends on the nature of the underlying biological system, such as whether exact simulation is required [11, 10], whether certain reactions operate at different timescales [12, 28], or whether non-Markovian reaction rates are needed [2, 16].

Rather than implementing custom stochastic simulation algorithms for each process calculus, we propose to use a generic abstract machine which can encode a range of process calculi and which can be instantiated to use a range of reaction-based simulation algorithms. The abstract machine functions as a just-in-time compiler, which dynamically updates the set of possible reactions and chooses the next reaction in an iterative cycle. Thus, the abstract machine computes only those species and reactions which are needed to proceed with the simulation. The abstract machine is instantiated to a particular calculus by defining two functions: one for converting a process of the calculus to a set of species and another for computing the set of possible reactions between species. The abstract machine is instantiated to a particular simulation algorithm by defining three functions: one for computing the next reaction, another for computing the reaction activity from an initial set of reactions and species populations, and a third for updating the reaction activity as the species populations change over time.

Although the idea of integrating different modelling and simulation methods within a common framework is not a new one [9], our approach is the first attempt to formally define a generic framework for simulating a broad range of process calculi with an arbitrary reaction-based simulation algorithm. Maintaining a clear separation between the simulation algorithm and the language specification allows us to readily instantiate the machine to different process calculi and to add new simulation algorithms, such as the non-Markovian simulation algorithm of [16], which can then be shared between calculi. This allows for rapid prototyping of novel domain-specific modelling languages by allowing substantial code re-use. Furthermore, the approach can be used to simulate multiple interacting calculi simultaneously, producing a multi-language environment for biological simulation.

There is another, perhaps more fundamental motivation for this work, which is to understand and formalise the relationship between multiple biological modelling languages. By defining a common abstract machine, we have essentially defined an underlying computational model to which different languages can be mapped. This is a first step towards a formal underpinning that encompasses the execution semantics of multiple biological modelling languages.

We first use our generic abstract machine to simulate a variant of stochastic pi-calculus [20], which supports a basic complexation primitive using bound output. We instantiate the abstract machine so that it stores complexes as a single species, allowing efficient simulation of systems involving complexation. We also show how variants of the bioambient calculus [19], the kappa calculus [7] and the DNA strand displacement calculus [15] may be encoded into the abstract machine. Each of these calculi is encoded by defining appropriate

functions for translating between species and processes of the calculus, and for generating the set of reactions between species. We define a generic proof of correctness for an arbitrary process calculus with respect to an arbitrary Markovian simulation algorithm, which is parameterised by the functions used to define that calculus. This approach can be re-used to quickly prove the correctness of language implementations that instantiate the generic abstract machine.

Simulating different domain-specific process calculi is important as it allows us to choose the language which is best-suited for modelling the system of interest. However, when we consider larger, more complex biological systems it may not be appropriate to use one language for the entire model, since large-scale biological systems tend to be composed of well-defined sub-units that can vary considerably in function. A natural approach to modelling such complex systems is to construct the model from communicating sub-models, each of which represents a well-defined functional unit of the system. Compositional modelling is good practise and has additional benefits, as the sub-models can be analysed independently and re-used in different settings. Our abstract machine is sufficiently powerful to support models that are recursively composed of sub-models written using different calculi.

The paper is structured as follows. In Sec. 2 we define the generic abstract machine, and in Sec. 3 we instantiate the simulation method of the abstract machine with the Direct Method [11] and the Next Reaction Method [10]. In Sec. 4 we instantiate the abstract machine to encode the pi-calculus, the bioambient calculus, the kappa calculus and the DNA strand displacement calculus. In Sec. 5 we present a generic method for proving the correctness of the abstract machine with respect to an arbitrary process calculus and a Markovian simulation algorithm. We then present specific instances of this generic proof for the various calculi. In Sec. 6 we show how multi-calculus models can be implemented in this framework.

## 2. Generic Abstract Machine

### 2.1. Preliminaries

We first define the main syntactic conventions that will be used in the remainder of the paper. We write  $\tilde{O}$  to denote a finite set  $\{O_1, \dots, O_N\}$  and  $\bar{I}$  to denote a finite multiset  $[I_1, \dots, I_N]$ . We also allow a multiset  $\bar{I}$  to be written as a set of pairs  $\{(I_1, i_1), \dots, (I_M, i_M)\}$ , where each pair  $(I, i)$  denotes an element  $I$  and its corresponding multiplicity  $i$ , with  $i > 0$ . We let  $\cup$  denote set union and  $\uplus$  denote multiset union, according to their standard definitions. We write  $I_1 \in \bar{I}$  as short for  $(I_1, i_1) \in \bar{I}$  in cases where we do not care about the multiplicity of  $I_1$ . We write  $\#\tilde{S}$  to denote the number of elements in the set  $\tilde{S}$ . We write  $\{E_i \mid C_1; \dots; C_N\}$  to denote the set of elements  $E_i$  that satisfy all of the conditions  $\{C_1, \dots, C_N\}$ . We write  $\{E_1 \mapsto v_1, \dots, E_N \mapsto v_N\}$  to denote a mapping from elements  $E_i$  to values  $v_i$ . For a given mapping  $M$ , we write  $\text{dom}(M)$  for the domain of  $M$  and  $M(E)$  for the value associated with element  $E$  in  $M$ . We also write  $M\{E \mapsto v\}$  for  $M$  updated so that  $E$  maps to  $v$ , and  $M \setminus E$  for  $M$  updated so that the mapping for  $E$  is removed. Finally, we write  $M\{M'\}$  for  $M$  updated so that  $E$  maps to  $v$  in  $M$  for each mapping  $E \mapsto v$  in  $M'$ .

---

$T ::= (t, S, R)$	Time $t$ , species map $S$ , reaction map $R$
$S ::= \{I_1 \mapsto i_1, \dots, I_N \mapsto i_N\}$	Map from a species $I$ to its population $i$
$R ::= \{O_1 \mapsto A_1, \dots, O_N \mapsto A_N\}$	Map from a reaction $O$ to its activity $A$
$O ::= (\bar{I}, r, \bar{I}')$	Reaction $\bar{I} \xrightarrow{r} \bar{I}'$ with rate $r$

---

**Definition 1.** Syntax of the generic abstract machine, where a term  $T$  consists of the current time  $t$ , a species map  $S$  and a reaction map  $R$ . We let  $\bar{I}$  denote a multiset of species  $[I_1, \dots, I_N]$ .

---

## 2.2. Syntax and Semantics

The syntax of the generic abstract machine is summarised in Definition 1. A machine term  $T$  is a triple  $(t, S, R)$ , where  $t$  is the current time,  $S$  is a map from a species  $I$  to its integer population  $i$ , and  $R$  is a map from a reaction  $O$  to its *activity*  $A$ , which is used to compute the next reaction. The data structure for the activity  $A$  will depend on the choice of simulation algorithm. Each reaction is represented as a tuple  $(\bar{I}, r, \bar{I}')$ , where  $\bar{I}$  denotes the multiset of reactant species,  $\bar{I}'$  denotes the multiset of product species and  $r$  denotes the reaction rate. The syntax of species  $I$  is specific to the choice of process calculus. The structure of a term of the abstract machine is summarised in tabular form as follows:

Machine term $T$				
Time $t$	Species map $S$		Reaction map $R$	
	Species	Population	Reaction	Activity
	$I_1$	$i_1$	$\bar{I}_1 \xrightarrow{r_1} \bar{I}'_1$	$A_1$
	$\dots$	$\dots$	$\dots$	$\dots$
	$I_N$	$i_N$	$\bar{I}_M \xrightarrow{r_M} \bar{I}'_M$	$A_M$

To instantiate the abstract machine with a given process calculus, we simply provide a function  $species(P)$  for transforming a process  $P$  of the calculus to a multiset of species, and a function  $reactions(I, \tilde{I}')$  for computing the multiset of reactions between a new species  $I$  and an existing set of species  $\tilde{I}'$ . The  $species$  function is used to initialise the abstract machine at the beginning of a simulation, while the  $reactions$  function is used to update the set of possible reactions dynamically. This is an important technical development that allows systems with potentially unbounded numbers of species and reactions to be simulated, which is not possible using standard stochastic simulation algorithms.

To instantiate the abstract machine with a given simulation method, we provide a function  $next(T)$  for choosing the next reaction from a term  $T$ , a function  $init(\bar{O}, T)$  for initialising a term with a multiset of reactions  $\bar{O}$ , and a function  $updates(I, T)$  for updating the activity of the reactions in a term  $T$  that are affected by a given species  $I$ . The abstract machine executes a given simulation method by repeated application of the following rule:

$$\frac{(\bar{I}, r, \bar{I}'), a, t' = next(t, S, R)}{(t, S, R) \xrightarrow{a, (\bar{I}, r, \bar{I}')} \bar{I}' \oplus ((t', S, R) \ominus \bar{I})} \quad (1)$$

This rule selects a reaction using the  $next$  function, which returns the chosen reaction, its *propensity*  $a$  and the new simulation time  $t'$ . The chosen reaction

---


$$\begin{aligned}
P \oplus T &\triangleq \text{species}(P) \oplus T \\
(I, i) \oplus (t, S, R) &\triangleq (t, S', R\{R'\}) \text{ if } \tilde{I}' = \text{dom}(S); I \notin \tilde{I}'; S' = S\{I \mapsto i\}; \\
&\quad \bar{O} = \text{reactions}(I, \tilde{I}'); R' = \text{init}(\bar{O}, (t, S', R)) \\
(I, i) \oplus (t, S, R) &\triangleq (t, S', R\{R'\}) \text{ if } S(I) = i'; S' = S\{I \mapsto i' + i\}; \\
&\quad R' = \text{updates}(I, (t, S', R)) \\
(t, S, R) \ominus (I, i) &\triangleq (t, S', R\{R'\}) \text{ if } S(I) = i'; i' \geq i; S' = S\{I \mapsto i' - i\}; \\
&\quad R' = \text{updates}(I, (t, S', R))
\end{aligned}$$


---

**Definition 2.** Adding and removing species in the generic abstract machine. We let  $\bar{O}$  denote a multiset of reactions  $[O_1, \dots, O_N]$ . If  $\bar{I}$  is a multiset  $\{(I_1, i_1), \dots, (I_N, i_N)\}$  we write  $\bar{I} \oplus T$  for  $(I_1, i_1) \oplus \dots \oplus (I_N, i_N) \oplus T$ , and  $T \ominus \bar{I}$  for  $T \ominus (I_1, i_1) \ominus \dots \ominus (I_N, i_N)$ .

---

is executed by removing the reactants  $\bar{I}$ , adding the products  $\bar{I}'$  and updating the current simulation time in the machine term.

Definitions for adding and removing species are summarised in Definition 2. A process  $P$  is added to a machine term  $T$  by computing the multiset of species  $[I_1, \dots, I_N]$  which correspond to  $P$  and then adding each of these species to the term. If a new species  $I$  is already present in the term then its population is incremented in  $S$  and the activity of the affected reactions is updated. If the species is not already present in the term then its population is initialised in  $S$  and new reactions for the species are computed, together with their activity. The operation  $T \ominus \bar{I}$  removes the species  $\bar{I}$  from the machine term  $T$ , by decreasing the corresponding species populations and updating the activity of the affected reactions.

For simulation of systems with large numbers of transient species, the machine can be modified so that species with zero population are garbage-collected. This can be achieved by modifying the definition of species removal so that species with zero population are removed from the species map, and reactions involving those species are removed from the reaction map, as follows:

$$\begin{aligned}
(t, S, R) \ominus (I, i) &\triangleq (t, S', R') \text{ if } S(I) = i'; i' = i; S' = S \setminus I; o_i = R(O_i); \\
&\quad R' = \{O_i \mapsto o_i \mid O_i = (\bar{J}, r, \bar{J}'); I \notin \bar{J}; O_i \in \text{dom}(R)\}
\end{aligned}$$

This definition implies that species are garbage-collected as soon as their population reaches zero. In practice, this approach will have significant benefits if a large number of species are used only once.

For Markovian simulation methods in which all reaction rates  $r$  are assumed to be exponentially distributed of the form  $\text{exp}(\lambda)$ , we can compute the Continuous Time Markov Chain (CTMC) of the abstract machine directly from (1). We first derive a CTMC semantics from the reduction relation  $T \xrightarrow{a, O} T'$ , using the following rule:

$$\frac{a = \left( \sum_{\{b, O \mid T \xrightarrow{b, O} T'\}} b \right) > 0}{T \xrightarrow{a} T'} \quad (2)$$

This rule sums the propensities  $b$  of all reactions  $T \xrightarrow{b, O} T'$  that give rise to the same term  $T'$ . The CTMC semantics therefore corresponds the set of initial

---


$$\begin{aligned}
next(t, S, R) &\triangleq (O_\mu, a_\mu, t + t') \text{ if } a_0 = \sum_{O_i \in dom(R)} R(O_i) > 0; \\
&\quad t' = \left(\frac{1}{a_0}\right) \ln\left(\frac{1}{n_1}\right); \quad \sum_{i=1}^{\mu-1} a_i < n_2 a_0 \leq \sum_{i=1}^{\mu} a_i \\
init(\bar{O}, (t, S, R)) &\triangleq \{O_i \mapsto propensity(O_i, S) \mid O_i \in merge(\bar{O})\} \\
merge(\bar{O}) &\triangleq \{(\bar{I}, exp(\lambda \times o_i), \bar{I}') \mid ((\bar{I}, exp(\lambda), \bar{I}'), o_i) \in \bar{O}\} \\
updates(I, (t, S, R)) &\triangleq \{O_i \mapsto propensity(O_i, S) \mid O_i \in dom(R); \\
&\quad O_i = (\bar{J}, r, \bar{J}') \mid I \in \bar{J}\}
\end{aligned}$$

$$propensity(\{(I_1, i_1), \dots, (I_N, i_N)\}, exp(\lambda), \bar{I}', S) \triangleq \lambda \times \binom{S(I_1)}{i_1} \times \dots \times \binom{S(I_N)}{i_N}$$

**Definition 3.** Instantiation of the generic abstract machine with the Direct Method [11], where  $n_1$  and  $n_2$  denote two random numbers from the standard uniform distribution  $U(0, 1)$ . The notation  $\binom{n}{k}$  denotes the binomial coefficient indexed by  $n$  and  $k$ , which computes the number of distinct  $k$ -element subsets that can be obtained from a set of size  $n$ . All reaction rates  $r$  are assumed to be exponentially distributed of the form  $exp(\lambda)$ , where  $\lambda$  is a real number.

---

transitions from a given term  $T$ . From this semantics we derive a corresponding CTMC by recursively enumerating the set of all transitions starting from  $T$ , as defined by the following recursive function:

$$CTMC(T) = flatten(\{T \xrightarrow{a} T'\} \cup CMTC(T') \mid T \xrightarrow{a} T') \quad (3)$$

where  $flatten(\{S_1, \dots, S_N\}) = S_1 \cup \dots \cup S_N$ . Essentially, for each transition  $T \xrightarrow{a} T'$  we recursively compute the transitions for the resulting process  $T'$ . As an example, consider the machine term  $T = (0, S, R)$  with  $S = \{A \mapsto 3\}$  and  $R = \{([A, A], r, [B]) \mapsto 3 \cdot r\}$ . This defines a machine term with a single reaction  $A + A \xrightarrow{r} B$  and with three copies of species  $A$ , written  $[A, A, A]$  for short. The corresponding CTMC for this machine term consists of the single transition  $[A, A, A] \xrightarrow{3 \cdot r} [A, B]$ , which results from the application of the reaction  $A + A \xrightarrow{r} B$ . Since there are three ways in which the reaction can be applied, the rate of the transition is given by  $3 \cdot r$ , which corresponds to the propensity of the reaction.

### 3. Instantiating the Abstract Machine with a Simulation Method

This section describes how the abstract machine can be instantiated with a chosen simulation method, by defining appropriate *next*, *init* and *updates* functions. We first present an instantiation with the Direct Method [11] followed by an instantiation with the Next Reaction Method [10]. In general, the abstract machine can be instantiated to a range of other reaction-based simulation algorithms, including algorithms for handling both Markovian and non-Markovian rates simultaneously, as shown in [16].

#### 3.1. Gillespie's Direct Method

An instantiation of the generic abstract machine with the Direct Method of [11] is outlined in Definition 3. Each reaction  $O_i$  in  $R$  is mapped to its

---


$$\begin{aligned}
next(t, S, R) &\triangleq (O, a, t') \text{ if } R(O) = (a, t'); a > 0; & (4) \\
& t' = \min\{t \mid R(O) = (a, t)\} \\
init(\overline{O}, (t, S, R)) &\triangleq \{O_i \mapsto (t', a) \mid O_i \in merge(\overline{O}); & (5) \\
& a = propensity(O_i, S); t' = t + delay(a)\} \\
updates(I, (t, S, R)) &\triangleq \{O \mapsto (t', a') \mid R(O) = (t'', a); O = (\overline{J}, r, \overline{J}'); & (6) \\
& I \in \overline{J}; a' = propensity(O, S); \\
& t' = t + (a/a')(t'' - t)\}
\end{aligned}$$

**Definition 4.** Instantiation of the generic abstract machine with the Next Reaction Method [10]. The activity of each reaction  $O$  is recorded as a pair  $A = (a, t)$ , where  $a$  denotes the reaction propensity and  $t$  denotes the putative time at which the reaction is scheduled to occur. The function  $delay(a)$  computes a time interval for an exponential reaction with propensity  $a$ .

---

activity, which in this case is simply the propensity  $a$  of the reaction. The function  $propensity((\overline{I}, r, \overline{I}'), S)$  computes the propensity of the reaction  $(\overline{I}, r, \overline{I}')$  by multiplying the number of distinct combinations of the reactants  $\overline{I}$  by the exponential rate  $\lambda$  of the reaction, assuming that all reaction rates are exponentially distributed of the form  $exp(\lambda)$ . The number of distinct combinations of the reactants is computed using the binomial coefficient, by looking up the population of each reactant in the species map  $S$ . The function  $init(\overline{O}, T)$  computes the propensity of each reaction in the multiset  $\overline{O}$  using the initial species populations in  $T$ . In order to merge multiple instances of the same reaction, the rate is multiplied by the number of occurrences of the reaction. Note that the merge is only applicable in the case of Markovian simulation methods such as the Direct Method, in which all of the reaction rates are assumed to be exponentially distributed. The function  $updates(I, T)$  recomputes the propensities of all the reactions in  $T$  for which  $I$  is a reactant. Note that, in general, this function should be defined in such a way that  $T \ominus (I_1, i_1) \ominus (I_2, i_2) = T \ominus (I_2, i_2) \ominus (I_1, i_1)$ , to ensure that the order in which species are removed has no effect on the corresponding propensities. Finally, the function  $next(T)$  chooses a reaction  $O_\mu$  from  $T$  with probability proportional to the reaction propensity  $a_\mu$ , and computes the corresponding duration  $t'$  of the reaction according to [11].

### 3.2. Next Reaction Method

An instantiation of the generic abstract machine with the Next Reaction Method (NRM) of [10] is outlined in Definition 4. Each reaction  $O_i$  in  $R$  is mapped to its activity, which is recorded as a pair  $(a, t)$ , where  $a$  is the propensity of the reaction and  $t$  is the putative time at which the reaction is scheduled to occur. The definitions of the  $propensity$  and  $merge$  functions are the same as in Definition 3. The next reaction is chosen to be the one with the smallest putative time, as defined by the function  $next(T)$ , which returns the chosen reaction  $(\overline{J}, r, \overline{J}')$  together with its putative time  $t'$  and its propensity  $a$  (4).

When a new reaction is created, the NRM computes the putative time of the reaction according to its propensity (5). This algorithm also provides a

way to update the putative times of Markovian reactions when their propensity changes, without generating a new random variable (6). When a new reaction is added to the machine, its propensity is computed and used to generate a random variable following the probability distribution of the reaction (5). Markovian reactions are updated by computing the new propensity and rescaling the putative time (6). It may be that the old propensity is 0, preventing direct use of the rescaling function. This case can be handled by keeping additional variables to register the last non-zero propensity and to rescale according to this old value (as discussed in note 11 of [10]). Similarly, if the new propensity is 0, the putative time is set to infinity.

#### 4. Instantiating the Abstract Machine with a Process Calculus

This section instantiates the generic abstract machine with four different process calculi, namely the stochastic pi-calculus, the bioambient calculus, the kappa calculus and the DNA strand displacement calculus, by defining appropriate *species* and *reactions* functions. The stochastic pi-calculus is an example of an agent-based modelling language, where the behaviour of an individual is described by a separate process. Here we focus on how multiple instances of process complexes can be grouped together for improved efficiency. The bioambient calculus is an example of an agent-based language with compartments. Here we focus on the aspects of the instantiation that relate specifically to the movement of compartments relative to each other. The kappa calculus is an example of a rule-based modelling language, in which the interactions between individuals are described as rules. Here we focus on the correspondence between rules and reactions. Finally, the DNA strand displacement calculus is an example of a domain-specific modelling language, tailored to a particular class of systems. Here we focus on demonstrating how a domain-specific calculus can be handled within the generic framework.

For each calculus, the semantics of the calculus itself is used to derive the corresponding *reactions* function, in contrast with [16]. Furthermore, a separate *process* function is defined for each calculus, in order to translate a species back to a process. This function will be used to prove the correctness of the instantiations in Sec. 5. In general, there are many different ways in which the abstract machine can be instantiated with a given calculus, with broad scope for calculus-specific optimisations. The abstract machine can also be instantiated to a broad range of calculi beyond the ones presented here.

##### 4.1. Stochastic Pi-Calculus

In this section we present an instantiation of the generic abstract machine with a variant of stochastic pi-calculus. The instantiation includes an optimisation for the simulation of process complexes. The instantiation is based on [16], with the key difference that the *reactions* function is now defined directly in terms of the calculus semantics.

###### 4.1.1. Calculus Syntax and Semantics

The syntax of the variant of stochastic pi-calculus used in this paper is given in Definition 5 and is based on [20]. A process  $P$  can be a choice of actions  $C$ , an instance  $X(\tilde{n})$  of a definition  $X$  with parameters  $\tilde{n}$ , a parallel composition

---

$P ::=$	$C$	Choice	$\pi ::=$	$\tau_r$	Delay
	$  X(\tilde{n})$	Instance		$  !x(\tilde{n})$	Send
	$  P_1   P_2$	Parallel		$  !x(\nu\tilde{m})$	Bind
	$  \nu x P$	Restriction		$  ?x(\tilde{m})$	Receive

$$\begin{aligned}
C &::= \pi_1^{i_1}.P_1 + \dots + \pi_N^{i_N}.P_N && \text{Actions} \\
E &::= X_1(\tilde{m}_1) \mapsto P_1, \dots, X_N(\tilde{m}_N) \mapsto P_N && \text{Environment}
\end{aligned}$$

**Definition 5.** Syntax of stochastic pi-calculus, where the empty choice denotes the null process  $\mathbf{0}$ . For each definition  $X(\tilde{m}) \mapsto P$  in the environment, we assume that  $\tilde{m} \subseteq \text{fn}(P)$ , where  $\text{fn}(P)$  denotes the free names of  $P$ . The restriction  $\nu x P$  binds the name  $x$  in  $P$  and both  $!x(\nu\tilde{m}).P$  and  $?x(\tilde{m}).P$  bind names  $\tilde{m}$  in  $P$ . We also assume that all recursive calls to a definition are *guarded* inside an action prefix  $\pi$ , such that for a given definition  $X(\tilde{m}) \mapsto P$ , any recursive call to  $X$  inside  $P$  can only occur after an action  $\pi$ . This prevents infinite expansion of process definitions.

---


$$P | \mathbf{0} \equiv P \tag{7}$$

$$P_1 | P_2 \equiv P_2 | P_1 \tag{8}$$

$$P_1 | (P_2 | P_3) \equiv (P_1 | P_2) | P_3 \tag{9}$$

$$\nu x \mathbf{0} \equiv \mathbf{0} \tag{10}$$

$$\nu x \nu y P \equiv \nu y \nu x P \tag{11}$$

$$\nu x (P_1 | P_2) \equiv P_1 | \nu x P_2 \text{ if } x \notin \text{fn}(P_1) \tag{12}$$

$$X(\tilde{n}) \equiv P_{\{\tilde{n}/\tilde{m}\}} \text{ if } E(X(\tilde{m})) = P \tag{13}$$

**Definition 6.** Structural congruence axioms in the stochastic pi-calculus, assuming a global environment  $E$ . Structural congruence is reflexive, symmetric and transitive, and holds in any context inside a process or a choice. Processes are assumed to be equal up to renaming of bound names and reordering of terms in a choice.

---


$$\tau_r^i.P + C \xrightarrow{r,i} P \tag{14}$$

$$!x(\tilde{n})^{i_1}.P_1 + C_1 | ?x(\tilde{m})^{i_2}.P_2 + C_2 \xrightarrow{\text{rate}(x), (i_1, i_2)} P_1 | P_2_{\{\tilde{n}/\tilde{m}\}} \tag{15}$$

$$!x(\nu\tilde{n})^{i_1}.P_1 + C_1 | ?x(\tilde{m})^{i_2}.P_2 + C_2 \xrightarrow{\text{rate}(x), (i_1, i_2)} \nu\tilde{n}(P_1 | P_2_{\{\tilde{n}/\tilde{m}\}}) \tag{16}$$

$$P \xrightarrow{r,w} P' \Rightarrow \nu x P \xrightarrow{r,w} \nu x P' \tag{17}$$

$$P \xrightarrow{r,w} P' \Rightarrow P | Q \xrightarrow{r,w} P' | Q \tag{18}$$

$$Q \equiv P \xrightarrow{r,w} P' \equiv Q' \Rightarrow Q \xrightarrow{r,w} Q' \tag{19}$$

**Definition 7.** Reduction in the stochastic pi-calculus, where a reduction identifier  $w$  can be either a single identifier  $i$  denoting a delay, or a pair of identifiers  $(i_1, i_2)$  denoting an interaction.

---

of processes  $P \mid Q$ , or a process  $\nu x P$  with a private channel  $x$ . A choice  $C$  consists of a competition between zero or more actions  $\pi^i.P$ , where  $\pi$  is the action that can be performed, after which process  $P$  is executed, and  $i$  is an index used for identifying individual actions. We take 0 to be the default index, and we abbreviate  $\pi^0$  to  $\pi$ . An action  $\pi$  can be a delay  $\tau_r$  of rate  $r$ , a send  $!x(\tilde{n})$  of values  $\tilde{n}$  on channel  $x$ , a send  $!x(\nu\tilde{n})$  of private values  $\tilde{n}$  on channel  $x$ , or a receive  $?x(\tilde{m})$  of values  $\tilde{m}$  on channel  $x$ . An environment  $E$  consists of a set of definitions  $X(\tilde{m}) \mapsto P$ , where  $X$  is the name of the definition,  $\tilde{m}$  are its parameters and  $P$  is the corresponding process.

The structural congruence axioms for the stochastic pi-calculus are summarised in Definition 6 in the standard way, and the reduction rules are summarised in Definition 7. The notation  $P \xrightarrow{r,w} P'$  states that the process  $P$  can reduce to  $P'$  by performing a reaction  $w$  at rate  $r$ . The reaction identifier  $w$  can be an index  $i$  denoting a particular delay  $\tau_r^i$ , or a pair of indices  $(i_1, i_2)$  denoting an interaction between two actions with indices  $i_1$  and  $i_2$ , respectively. A process can evolve on its own by executing a delay  $\tau_r$ . Two processes can evolve simultaneously by communicating or binding with each other. Communication between two processes occurs when one process sends values  $\tilde{n}$  on a channel  $x$ , denoted by  $!x(\tilde{n})$ , and a parallel process receives these values on the same channel  $x$ , denoted by  $?x(\tilde{m})$ . A binding between two processes can occur if one process sends private values  $\tilde{n}$  on a channel  $x$ , denoted by  $!x(\nu\tilde{n})$ , which are then shared only between the sender and receiver, representing the formation of a complex between the two. Note that we include an explicit notion of bound output  $!x(\nu\tilde{n})$  in order to directly model the formation of complexes in the calculus. This also allows us to convert a restricted choice  $\nu m !x(m)$  to a choice  $!x(\nu m)$ , where the scope of the binding is moved inside the choice. This in turn simplifies the treatment of complex formation, for example in rule (16).

Stochastic behaviour is introduced into the calculus by associating each delay  $\tau_r$  with a rate  $r$  and by associating each channel  $x$  with a corresponding rate given by  $rate(x)$ . Each channel  $x$  therefore denotes a pair  $(n, r)$ , where  $n$  denotes the channel name and  $r$  denotes the channel rate, such that  $rate((n, r)) = r$ . We assume that distinct channels have distinct names, and that renaming of channels preserves the rate. Each rate characterises a probability distribution. For exponentially distributed rates of the form  $exp(\lambda)$ , the probability of a reaction occurring within time  $t$  is given by  $F(t) = 1 - e^{-\lambda t}$ . The average duration of the reaction is given by the mean  $1/\lambda$  of this distribution.

We derive a CTMC semantics for the stochastic pi-calculus directly from its reduction relation, by first defining an *indexed form* for processes as follows, based on [19].

**Definition 8.** A process  $P$  is in *indexed form* if it is of the form  $\nu x_1 .. \nu x_M (C_1 \mid .. \mid C_N)$  and each unguarded action  $\pi^i$  is associated with a unique index  $i$ .

We can show that every process is structurally congruent to a process in indexed form, up to renaming of indices (the proof is straightforward). Note that the sole purpose of the indices is to uniquely identify each individual action, and that renaming these indices has no effect on the reductions that a given process can perform (see [19] for further details). Note that a process in indexed form does not necessarily remain in indexed form after a reduction takes place. For example, consider the process  $\tau_r^1.(X \mid X)$  with  $X \mapsto \tau_r^2$  in the environment. This process is in indexed form since each unguarded action has a unique index.

However, after a reduction we obtain the process  $(\tau_r^2 \mid \tau_r^2)$ , which is no longer in indexed form. In general, indexed form is used to uniquely identify each reduction by labelling it with the indices of the actions involved. Thus, it is important that processes are *not* equal up to renaming of indices, otherwise an infinite number of reactions could be generated. Moreover, we only need to convert a process to indexed form when we wish to count the number of distinct reductions, for example when computing the CTMC corresponding to a given process. The CTMC semantics is then given by the following rule, assuming process  $P$  is in indexed form, where the function  $index(P')$  converts process  $P'$  to indexed form.

$$\frac{a = \left( \sum_{\{\lambda, w \mid P \xrightarrow{exp(\lambda), w} P'\}} \lambda \right) > 0}{P \xrightarrow{a} index(P')} \quad (20)$$

The requirement that  $a > 0$  ensures that we can derive  $P \xrightarrow{a} P'$  precisely when there is a stochastic pi-calculus reduction from  $P$  to  $P'$ . We rely implicitly on the fact that  $\lambda > 0$  for all exponential rates  $\lambda$ . Using this semantics, we then derive a corresponding CTMC for a given process  $P$  by recursively enumerating the set of transitions  $P \xrightarrow{a} P'$  for each distinct process  $P'$ , according to (3). For the purposes of computing the CTMC we assume that processes are distinct up to structural congruence and injective renaming of indices. This ensures that we do not generate duplicate transitions  $P \xrightarrow{a} P'$  if  $P'$  is re-ordered or its indices are renamed.

#### 4.1.2. Computing Species and Reactions from Calculus Processes

To instantiate the generic abstract machine with the stochastic pi-calculus, the first step is to define what constitutes a species. Here we assume that a species is either an instance  $X(\tilde{n})$  or a complex of instances  $\nu\tilde{n}(X_1(\tilde{n}_1) \mid \dots \mid X_M(\tilde{n}_M))$ , where each instance corresponds to a choice of actions. Our approach is motivated by the observation that a choice of actions is the basic unit of computation, where two parallel choices interact by communicating over shared channels. An alternative approach could be to assume that a species corresponds directly to a choice of actions, instead of using a named instance  $X(\tilde{n})$ . Our decision to use a named instance has the advantage that a species can be explicitly identified in a biological model by a meaningful name, and that the results of a simulation can be directly linked to the original model via this name. In order to formalise the notion of a species in stochastic pi-calculus, we define a normal form for processes (Definition 9), and show that all processes are structurally congruent to a normal form (Proposition 12).

**Proposition 12.** All processes of the stochastic pi-calculus are structurally congruent to a normal form according to Definition 9.

*Proof.* By induction on Definition 10. Using the structural congruence rules of Definition 6, we augment the environment such that all choices are defined separately (13), and we replace all instances that are not a choice with their corresponding process definition (13). Using the structural congruence rule for scoping (12), we modify the scope of a restriction such that a process is a parallel composition of species, where each species is either an instance or a complex.  $\square$

---

$P ::=$	$I_1 \mid \dots \mid I_N$	Species
$I ::=$	$X(\tilde{n})$	Instance
	$\nu \tilde{z} (X_1(\tilde{n}_1) \mid \dots \mid X_M(\tilde{n}_M))$	Complex
$C ::=$	$\pi_1^{i_1} . P_1 + \dots + \pi_N^{i_N} . P_N$	Choice
$E ::=$	$X_1(\tilde{m}_1) \mapsto C_1, \dots, X_N(\tilde{m}_N) \mapsto C_N$	Environment

---

**Definition 9.** Normal form of stochastic pi-calculus processes, where  $N \geq 0$  and  $M \geq 1$ . A process  $P$  is considered to be in normal form if it consists of a parallel composition of species  $I$ , where a species can be an instance  $X(\tilde{n})$  or a complex of instances  $\nu \tilde{z} (X_1(\tilde{n}_1) \mid \dots \mid X_M(\tilde{n}_M))$  and where every instance  $X(\tilde{n})$  corresponds to a choice of actions. We assume that  $\tilde{z} \cap \tilde{n}_1 \cap \dots \cap \tilde{n}_M \neq \emptyset$  and  $\tilde{z} \subseteq \tilde{n}_1 \cup \dots \cup \tilde{n}_M$ , so as to minimise the scope of restricted names.

---

$normal(\mathbf{0})$	$\triangleq$	$\mathbf{0}$	
$normal(X(\tilde{n}))$	$\triangleq$	$X(\tilde{n})$	<b>if</b> $E(X(\tilde{n})) = C$
$normal(X(\tilde{n}))$	$\triangleq$	$normal(P)$	<b>if</b> $E(X(\tilde{n})) = P \neq C$
$normal(P_1 \mid P_2)$	$\triangleq$	$normal(P_1) \mid normal(P_2)$	
$normal(C)$	$\triangleq$	$X(\tilde{n})$	<b>if</b> $E(X(\tilde{n})) = C$ <b>and</b> $X$ <i>fresh</i>
$normal(\nu x P)$	$\triangleq$	$insert(x, normal(P))$	
$insert(x, \prod_i I_i)$	$\triangleq$	$(\nu \tilde{z} \prod_k K_k) \mid \prod_j I_j$	<b>if</b> $I_k = \nu \tilde{z}_k K_k$ <b>and</b> $x \in \text{fn}(I_k), x \notin \text{fn}(I_j)$ <b>and</b> $\bigcap \tilde{z}_k = \emptyset, \tilde{z} = \{x\} \cup \bigcup \tilde{z}_k$ <b>and</b> $i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$ <b>and</b> $\mathcal{J} \cap \mathcal{K} = \emptyset, \mathcal{I} = \mathcal{J} \cup \mathcal{K}$

**Definition 10.** Computing the normal form of a stochastic pi-calculus process. We write  $\prod_i P_i$  as short for  $P_1 \mid \dots \mid P_N$ , assuming  $i \in \{1, \dots, N\}$ . We write  $E(X(\tilde{n})) = C$  as an abbreviation for  $E(X(\tilde{m})) = C'$  where  $C = C'_{\{\tilde{n}/\tilde{m}\}}$ . The case for  $normal(C)$  assumes that the environment contains a fresh definition  $X$  for the choice  $C$ , such that  $E(X(\tilde{n})) = C$  and  $X$  is not used elsewhere.

---

$species(P)$	$\triangleq$	$[I_1, \dots, I_N]$	<b>if</b> $normal(P) = (I_1 \mid \dots \mid I_N)$
$process([I_1, \dots, I_N])$	$\triangleq$	$(I_1 \mid \dots \mid I_N)$	
$reactions(I, \tilde{J})$	$\triangleq$	$mset(unary(I) \cup binary(I, (\tilde{J} \cup \{I\})))$	
$mset(\tilde{L})$	$\triangleq$	$\{((\bar{I}, r, \bar{I}'), k) \mid k = \#\{(\bar{I}, r, w, \bar{I}') \in \tilde{L}\}; (\bar{I}, r, w, \bar{I}') \in \tilde{L}\}$	
$unary(I)$	$\triangleq$	$\{([I], r, w, species(P)) \mid I \xrightarrow{r, w} P\}$	
$binary(I_1, \tilde{J})$	$\triangleq$	$\{([I_1, I_2], r, w, species(P)) \mid I_2 \in \tilde{J}; i_1 \in I_1; i_2 \in I_2;$ $w = (i_1, i_2); (I_1 \mid I_2) \xrightarrow{r, w} P\}$	

**Definition 11.** Generic abstract machine instantiated for the stochastic pi-calculus, using the definition of reduction in the calculus to derive the reactions. We assume a fixed global environment  $E$  containing all instance definitions. We write  $i \in I$  if identifier  $i$  is present within species  $I$ .

---

Using our normal form for processes (Definition 9), we now define the various functions that are needed to instantiate the generic abstract machine for stochastic pi-calculus (Definition 11). The function  $species(P)$  converts a process to a multiset of species, the  $process(\bar{I})$  function converts a multiset of species to a process, and the function  $reactions(I, \tilde{J})$  computes the multiset of reactions that the species  $I$  can perform with the set of species  $\tilde{J}$ . The function  $mset$  converts a set of reactions  $\tilde{L}$ , in which each reaction has a unique identifier  $w$ , to a multiset of reactions  $\bar{O}$  in which the identifiers are discarded. Note that the  $reactions$  function returns a multiset rather than a set since the same reaction can potentially be generated in multiple different ways, e.g. as in the process  $X \mapsto \tau_r.Y + \tau_r.Y$ . The function returns the multiset of unary reactions (delays) combined with the multiset of binary reactions (communications and bindings).

#### 4.1.3. Example

We illustrate the application of the generic abstract machine to the stochastic pi-calculus with the following simple example of complex formation.

$$\begin{aligned} A &= !x(\nu u).AB(u) \\ AB(u) &= !u.A \\ B &= ?x(u).BA(u) \\ BA(u) &= ?u.B \end{aligned}$$

Initially, 100 copies of processes  $A$  and  $B$  are added to the empty machine term, written  $(100 \cdot A \mid 100 \cdot B) \oplus (0, \emptyset, \emptyset)$ , where the notation  $100 \cdot X$  represents 100 parallel copies of the process  $X$ . This gives rise to the machine term  $(0, S, R)$ , where  $S$  and  $R$  are as follows:

$$\begin{aligned} S &= \{A \mapsto 100, B \mapsto 100\} \\ R &= [([A, B], rate(x), [\nu u(AB(u) \mid BA(u))]) \mapsto (10^4 \cdot (rate(x)), t_1)] \end{aligned}$$

The reaction involving species  $A$  and  $B$  is executed at time  $t_1$ , after which one copy of the species  $A$  and  $B$  are removed and one copy of the complex is added to the resulting machine term:

$$\nu u(AB(u) \mid BA(u)) \oplus ((t_1, S, R) \ominus [A, B])$$

This gives rise to the machine term  $(t_1, S_1, R_1)$ , where  $actions(\nu u(AB(u) \mid BA(u))) = \tau_u.(A \mid B)$  and  $S_1$  and  $R_1$  are as follows:

$$\begin{aligned} S_1 &= \{A \mapsto 99, B \mapsto 99, \nu u(AB(u) \mid BA(u)) \mapsto 1\} \\ R_1 &= [([A, B], rate(x), [\nu u(AB(u) \mid BA(u))]) \mapsto (9801 \cdot (rate(x)), t_3), \\ &\quad ([\nu u(AB(u) \mid BA(u))], rate(u), [A, B]) \mapsto ((rate(u)), t_2)] \end{aligned}$$

Note that existing simulation algorithms such as that from [20] handle  $N$  copies of the complex  $\nu u(AB(u) \mid BA(u))$  by creating a globally fresh name for each restricted channel  $u$ , as follows:

$$\nu u_1 \dots \nu u_N (AB(u_1) \mid BA(u_1) \mid \dots \mid AB(u_N) \mid BA(u_N))$$

In contrast, our approach treats these as  $N$  copies of the same complex  $\nu u(AB(u) \mid BA(u))$ , resulting in fewer species, fewer reactions and therefore a significantly

more efficient simulation. For the example described above, if there are  $N$  complexes we end up with a memory consumption of  $2N + 2$  species and  $N + 1$  reactions. In contrast, if we store complexes as species we end up with only 3 species and 2 reactions, resulting in a memory saving of order  $N$ . For many systems, the number of complexes can easily exceed 10000. If we use the next reaction method for simulation, the complexity scales with the logarithm of the number of reactions whose propensity is greater than zero [10]. For the above example with  $N = 10000$ , if we store complexes as species we obtain on the order of a 10000-fold saving in memory and a 10-fold speedup.

#### 4.2. Instantiation to the Bioambient Calculus

The bioambient calculus was first presented in [26] for modelling mobile compartments in biological processes. In this section, we instantiate the generic abstract machine to a version of the stochastic bioambient calculus. We provide full definitions for the bioambient calculus without the `merge` action, and briefly outline a straightforward extension for incorporating `merge`.

##### 4.2.1. Calculus Syntax and Semantics

The syntax and reduction rules of the stochastic bioambient calculus used in this section are presented in Definition 13 and are based on [19]. A process  $P$  can be a choice of actions  $C$ , an instance  $X(\tilde{n})$  of a definition  $X$  with parameters  $\tilde{n}$ , a parallel composition of processes  $P \mid Q$ , a process  $\nu x P$  with a private channel  $x$ , or an ambient  $\boxed{P}^a$  consisting of a process  $P$  inside the compartment named  $a$ . A choice  $C$  consists of a competition between zero or more actions  $\pi^i.P$ , as in the stochastic pi-calculus. An action  $\pi$  can be a delay  $\tau_r$ , a send  $\gamma!x(\tilde{n})$  of values  $\tilde{n}$  on channel  $x$ , or a receive  $\gamma?x(\tilde{m})$  of values  $\tilde{m}$  on channel  $x$ , where  $\gamma$  denotes the type of communication. This can be inside the same ambient (`local`), from one sibling ambient to another (`s2s`), from a child ambient to its parent (`c2p`) or from a parent ambient to a child (`p2c`). In addition, an action  $\pi$  can be a move  $\mu!x$  on channel  $x$  or an accept  $\mu?x$  on channel  $x$ , where  $\mu$  denotes the type of movement. This can be an ambient entering one of its siblings (`in`), a child ambient leaving its parent (`out`) or a merge of two sibling ambients (`merge`). An environment  $E$  consists of a set of definitions  $X(\tilde{m}) \mapsto P$ , as in the stochastic pi-calculus.

We derive a CTMC semantics for the bioambient calculus directly from its reduction relation, by first defining an *indexed form* for processes based on [19].

**Definition 14.** A process  $P$  is in *indexed form* if it is of the form  $\nu x_1 .. \nu x_M (C_1 \mid .. \mid C_N \mid \boxed{P_1}^{a_1} \mid .. \mid \boxed{P_K}^{a_K})$  where each process  $P_1, .., P_K$  is in indexed form, each ambient index  $a_1, .., a_K$  is unique and each unguarded action  $\pi^i$  is associated with a unique index  $i$ .

We can show that every process is structurally congruent to a process in indexed form, up to renaming of indices (the proof is straightforward [19]). The CTMC reduction semantics is then defined by the following rule, assuming process  $P$  is in indexed form, where the function  $index(P')$  converts process  $P'$  to indexed form:

$$a = \frac{\left( \sum_{\{\lambda, w \mid P^{exp(\lambda), w} P'\} } \lambda \right) > 0}{P \xrightarrow{a} index(P')} \quad (21)$$

---

$P, Q ::=$	$C \mid X(\tilde{n}) \mid P \mid Q \mid \nu x P \mid \boxed{P}^a$	Process
$C ::=$	$\pi_1^{i_1}.P_1 + \dots + \pi_N^{i_N}.P_N$	Choice
$E ::=$	$X_1(\tilde{m}_1) \mapsto P_1, \dots, X_N(\tilde{m}_N) \mapsto P_N$	Environment

$\pi ::=$	$\tau_r$ Delay	$\gamma ::=$	<b>local</b> Local
	$\mid \gamma!x(\tilde{n})$ Send		$\mid$ <b>s2s</b> Sibling
	$\mid \gamma?x(\tilde{m})$ Receive		$\mid$ <b>c2p</b> Parent
	$\mid \mu!x$ Move		$\mid$ <b>p2c</b> Child
	$\mid \mu?x$ Accept	$\mu ::=$	<b>in</b> Enter
			$\mid$ <b>out</b> Leave
$\lambda ::=$	$\gamma \mid \mu$ Binary actions		$\mid$ <b>merge</b> Merge

$$\begin{aligned}
& \tau_r^i.P + C \xrightarrow{r,i} P \\
& \text{local}!x(\tilde{n})^i.P + C \mid \text{local}?x(\tilde{m})^{i'}.P' + C' \xrightarrow{r_x,(i,i')} P \mid P'_{\{\tilde{n}/\tilde{m}\}} \\
& \boxed{Q \mid \text{c2p}!x(\tilde{n})^i.P + C}^a \mid Q' \mid \text{c2p}?x(\tilde{m})^{i'}.P' + C' \xrightarrow{r_x,(i,a,i')} \boxed{Q \mid P}^a \mid Q' \mid P'_{\{\tilde{n}/\tilde{m}\}} \\
& Q \mid \text{p2c}!x(\tilde{n})^i.P + C \mid \boxed{Q' \mid \text{p2c}?x(\tilde{m})^{i'}.P' + C'}^a \xrightarrow{r_x,(i,a,i')} Q \mid P \mid \boxed{Q' \mid P'_{\{\tilde{n}/\tilde{m}\}}}^a \\
& \boxed{Q \mid \text{s2s}!x(\tilde{n})^i.P + C}^a \mid \boxed{Q' \mid \text{s2s}?x(\tilde{m})^{i'}.P' + C'}^b \xrightarrow{r_x,(i,a,i',b)} \boxed{Q \mid P}^a \mid \boxed{Q' \mid P'_{\{\tilde{n}/\tilde{m}\}}}^b \\
& \boxed{Q \mid \text{in}!x^i.P + C}^a \mid \boxed{Q' \mid \text{in}?x^{i'}.P' + C'}^b \xrightarrow{r_x,(i,a,i',b)} \boxed{Q \mid P}^a \mid \boxed{Q' \mid P'}^b \\
& \boxed{Q \mid \text{out}!x^i.P + C}^a \mid Q' \mid \text{out}?x^{i'}.P' + C' \xrightarrow{r_x,(i,a,i',b)} \boxed{Q \mid P}^a \mid \boxed{Q' \mid P'}^b \\
& \boxed{Q \mid \text{merge}!x^i.P + C}^a \mid \boxed{Q' \mid \text{merge}?x^{i'}.P' + C'}^b \xrightarrow{r_x,(i,a,i',b)} \boxed{Q \mid P \mid Q' \mid P'}^b \\
& P \xrightarrow{r,w} P' \Rightarrow \boxed{P}^a \xrightarrow{r,w} \boxed{P'}^a \\
& P \xrightarrow{r,w} P' \Rightarrow \nu x P \xrightarrow{r,w} \nu x P' \\
& P \xrightarrow{r,w} P' \Rightarrow P \mid Q \xrightarrow{r,w} P' \mid Q \\
& Q \equiv P \xrightarrow{r,w} P' \equiv Q' \Rightarrow Q \xrightarrow{r,w} Q'
\end{aligned}$$

**Definition 13.** Syntax and core reduction rules of the stochastic bioambient calculus, based on [19]. For convenience, we write  $r_x$  as shorthand for  $rate(x)$ . Each reduction is labelled with its rate  $r$  and an index  $w$  which is a list of identifiers that denote the actions and ambients involved in a given reduction.

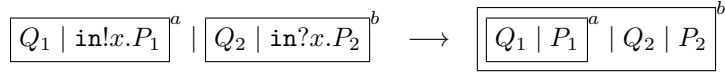
---

As in Sec. 4.1, we derive a corresponding CTMC for a given process  $P$  by recursively enumerating the set of transitions  $P \xrightarrow{a} P'$ , for each distinct process  $P'$ , according to (3). For the purposes of computing the CTMC we assume that processes are distinct up to structural congruence and injective renaming of indices.

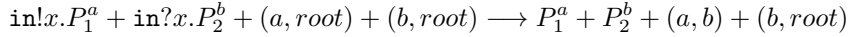
#### 4.2.2. Computing Species and Reactions from Processes

The most important feature of the bioambient calculus is the compartmentalising of processes into nested *ambients*, which prevents local reactions between processes from occurring across different ambients. We extract a flat set of reactions from a bioambient process by annotating all processes with the identifier of the ambient in which they are currently located. This assumes that each ambient is associated with a unique identifier, according to Definition 14. The hierarchical structure of ambients is specified by *location* species, where the location  $(a, b)$  means that the ambient  $a$  is a child of the ambient  $b$ . The identifier *root* is used to denote the top-level enclosing ambient. For example, the process  $\boxed{P_1 \mid \boxed{P_2}^a}^b$  is translated as the species multiset  $P_1^b, P_2^a, (a, b), (b, \text{root})$  where *root* is the identifier of the root ambient and  $a$  and  $b$  are ambient identifiers that are assumed to be globally unique. The assigning of locations to species in the definition of *species*( $P$ ) is formally presented in Definition 15, where the function *process*( $\bar{I}$ ) returns the process corresponding to the multiset of species  $\bar{I}$ .

The computation of reactions between species is defined in Definition 16. The reduction rules of the bioambient calculus are used directly to compute the set of reactions between process species  $I_1$  and  $I_2$  with their corresponding locations. For example,

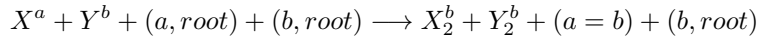


is encoded by the following reaction in the abstract machine:



Since the population of any location species is always either 0 or 1, applying such a reaction would disable reactions involving the old  $(a, \text{root})$  location of ambient  $a$ , and reactions involving the new  $(a, b)$  location would become possible instead.

Note that we do not explicitly include functions for supporting the **merge** operation in Definition 15. This can be achieved in a straightforward way by augmenting the location species with the notion of an *ambient alias* ( $a = b$ ), which states that  $a$  is merged into ambient  $b$ . Following the merge, every instance of ambient identifier  $a$  needs to be replaced with  $b$ . This is achieved by defining a *normal form* for species, which removes ambient aliases by substituting the corresponding ambient identifiers. For example, consider the process  $\boxed{A \mid X}^a \mid \boxed{B \mid Y}^b$  with  $X = \text{merge!}x^i.X_2$  and  $Y = \text{merge?}x^{i'}.Y_2$ . We have the following reaction:



After application of the reaction, the set of species is  $[X_2^b, Y_2^b, A^a, B^b, (a = b), (a, \text{root}), (b, \text{root})]$ , which is normalised to  $[X_2^b, Y_2^b, A^b, B^b, (b, \text{root})]$ .

---

$I ::=$	$X(\tilde{n})^a$	Process species
	$ $	$L$
$L ::=$	$(a, b)$	Ambient Location

$$\begin{aligned}
\text{species}(\mathbf{0}) &\triangleq \emptyset \\
\text{species}(P) &\triangleq \text{species}(P, \text{root}) \\
\text{species}(\boxed{P}^{a'}, a) &\triangleq \text{species}(P, a') \uplus [(a', a)] \\
\text{species}(X(\tilde{n}), a) &\triangleq [X(\tilde{n})^a] \text{ if } E(X(\tilde{n})) = C \\
\text{species}(X(\tilde{n}), a) &\triangleq \text{species}(P, a) \text{ if } E(X(\tilde{n})) = P \neq C \\
\text{species}(C) &\triangleq X(\tilde{n}) \text{ if } E(X(\tilde{n})) = C \text{ and } X \text{ fresh} \\
\text{species}(\nu x P, a) &\triangleq \text{species}(P_{\{y/x\}}, a) \text{ if fresh}(y) \\
\text{species}(P_1 | P_2, a) &\triangleq \text{species}(P_1, a) \uplus \text{species}(P_2, a)
\end{aligned}$$

$$\begin{aligned}
\text{process}(\bar{I}) &\triangleq \text{process}(\bar{I}, \text{root}) \\
\text{process}(\bar{I}, a) &\triangleq \text{parallel}(\bar{J} \uplus \{ \boxed{\text{process}(\bar{I} \setminus \bar{J}, b)}^b, 1 \} \mid (b, a) \in \bar{I} \} \\
&\quad \text{if } \bar{J} = \{ (X(\tilde{n}), k) \mid (X(\tilde{n})^a, k) \in \bar{I} \}
\end{aligned}$$

**Definition 15.** Functions for converting between species and processes. Note that the case for  $\text{species}(C)$  assumes that the environment contains a fresh definition  $X$  for the choice  $C$ , such that  $E(X(\tilde{n})) = C$  and  $X$  is not used elsewhere.

---

$$\begin{aligned}
\text{reactions}(I, \tilde{I}') &\triangleq \text{mset}(\text{unary}(I) \cup \text{nary}(I, (\tilde{I}' \cup \{I\}))) \\
\text{unary}(I) &\triangleq \{ ([I], r, w, \text{species}(P, a)) \mid I \xrightarrow{r, w} P; I = X(\tilde{n})^a \} \\
\text{nary}(I, \tilde{I}') &\triangleq \text{nary}2(I, \tilde{I}') \cup \text{nary}3(I, \tilde{I}') \cup \text{nary}4(I, \tilde{I}') \\
\text{nary}2(I_1, \tilde{I}') &\triangleq \{ (\bar{I}, r, w, \text{species}(P, a)) \mid \text{process}(\bar{I}) \xrightarrow{r, w} P; w = (i_1, i_2) \\
&\quad \bar{I} = [I_1, I_2] = [X(\tilde{n})^a, Y(\tilde{m})^a]; I_2 \in \tilde{I}' \} \\
\text{nary}3(I_1, \tilde{I}') &\triangleq \{ (\bar{I}, r, w, \text{species}(P, b)) \mid \text{process}(\bar{I}) \xrightarrow{r, w} P; w = (i_1, a, i_2); \\
&\quad \bar{I} = [I_1, I_2, I_3] = [X(\tilde{n})^a, Y(\tilde{m})^b, (a, b)]; a \neq b; [I_2, I_3] \subseteq \tilde{I}' \} \\
\text{nary}4(I_1, \tilde{I}') &\triangleq \{ (\bar{I}, r, w, \text{species}(P, c)) \mid \text{process}(\bar{I}) \xrightarrow{r, w} P; w = (i_1, a, i_2, c); \\
&\quad a \neq b; a \neq c; (d = b \text{ or } d = c); [I_2, I_3, I_4] \subseteq \tilde{I}'; \\
&\quad \bar{I} = [I_1, I_2, I_3, I_4] = [X(\tilde{n})^a, Y(\tilde{m})^b, (a, d), (b, c)] \}
\end{aligned}$$

**Definition 16.** Instantiation of the generic machine to the bioambient calculus. We write  $\text{parallel}(I_1, \dots, I_N)$  to stand for the process  $I_1 \mid \dots \mid I_N$  and  $\bar{I} \setminus \bar{J}$  for multiset difference. Note that in the cases for  $\text{nary}2, \text{nary}3$  and  $\text{nary}4$  we exploit the fact that the order of elements within a multiset is not fixed, for example in  $\text{nary}2$  it is possible that  $I_1 = Y(\tilde{m})^a$  and  $I_2 = X(\tilde{n})^a$ .

---

There is also broad scope for calculus-specific optimisations. For example, multiple distinct ambients containing the same process could be grouped together as a single species, and complexes in the bioambient calculus could be treated in a similar fashion to complexes in the stochastic pi-calculus, by modifying the *normal* function along the lines of Definition 10 to handle the hierarchical structure of bioambient processes. The treatment of complexes for bioambients is almost identical to that of stochastic pi-calculus, except that now complexes can potentially span multiple ambients. Having previously discussed how complexes can be optimised in Sec. 4.1, here we have focussed on how hierarchical compartments can be implemented in the generic abstract machine. The optimisation of complexes greatly increases the complexity of the definitions, and is not essential for defining a working implementation. For simplicity, we have therefore chosen to leave the optimisation of complexes for bioambients as future work. The ability to omit this complexity also highlights the flexibility of the generic abstract machine in supporting a range of implementations for a given calculus.

#### 4.2.3. Simple Example

Consider the following bioambient process definitions:

$$\begin{array}{ll}
E & = \text{in?}s.EL(s) + \text{in?}p.EL(p) & S & = \text{in!}s.X + \tau_r \\
EL(x) & = \text{out?}x.E & P & = \text{in!}p.X + \tau_r \\
& & X & = \text{out!}s.S + \text{out!}p.P
\end{array}$$

and the initial process  $\boxed{E}^a \mid \boxed{S|P}^b$ . After populating the machine, the population set  $S$  contains the species  $E^a, (a, \text{root}), S^b, P^b, (b, \text{root})$  and the related reactions are:

1.  $E^a + S^b + (a, \text{root}) + (b, \text{root}) \longrightarrow EL(s)^a + X^b + (a, \text{root}) + (b, a)$
2.  $S^b \longrightarrow \mathbf{0}$
3.  $E^a + P^b + (a, \text{root}) + (b, \text{root}) \longrightarrow EL(p)^a + X^b + (a, \text{root}) + (b, a)$
4.  $P^b \longrightarrow \mathbf{0}$

Suppose that reaction 1 is chosen. Then, the ambient at location  $(b, \text{root})$  moves to  $(b, a)$ . The species  $(b, \text{root}), E^a, S^b$  are set to zero population and new species  $EL(s)^a, X^b$  are created. The propensities of reactions 1, 2 and 3 are set to zero and a new reaction involving the added species is computed:

5.  $EL(s)^a + X^b + (a, \text{root}) + (b, a) \longrightarrow E^a + S^b + (a, \text{root}) + (b, \text{root})$ .

#### 4.3. Instantiation to the Kappa calculus

In this section we describe an instantiation of the generic abstract machine to simulate a variant of the kappa calculus [7].

##### 4.3.1. Calculus Syntax and Semantics

Kappa [6, 7] is a rule-based language suitable for modelling biological interactions. Kappa offers a compact formalism to describe the various interactions occurring between *agents* present in a solution. An agent is defined by its name and a set of sites it can use to interact with other agents. A site is either free or bound to one and only one other agent. A site may also have an internal state,

---

$P, G ::=$	$[a_1, \dots, a_M]$	Solution	$s ::=$	$x_\iota^\lambda$	Site
$a ::=$	$A(\sigma)$	Agent	$\iota ::=$	$\epsilon \mid m \in \mathbb{V}$	Internal state
$\sigma ::=$	$\{s_1, \dots, s_M\}$	Interface	$\lambda ::=$	$\epsilon \mid i \in \mathbb{N}$	Binding state
$E ::=$			$(G_1, r_1, G'_1), \dots, (G_N, r_N, G'_N)$ Rule set		

---

**Definition 17.** Syntax of kappa expressions, where  $r$  denotes the rate of a rule application,  $A$  is an agent name and  $x$  is a site name.

---

$$\begin{array}{llll}
x_{\iota_r}^{\lambda_r} / x_\iota^\lambda & \triangleq & x_{\iota_r}^{\lambda_r} & \\
x^{\lambda_r} / x_\iota^\lambda & \triangleq & x^{\lambda_r} & A(\sigma_r) / A(\sigma) \triangleq A(\sigma_r / \sigma) \\
\emptyset / \sigma & \triangleq & \sigma & \emptyset / G \triangleq G \\
s_r, \sigma_r / s, \sigma & \triangleq & s_r / s, \sigma_r / \sigma & a_r, G' / a, G \triangleq a_r / a, G' / G
\end{array}$$

**Definition 18.** Kappa solution replacement.

---

$\phi \in \text{embed}(G, P) : G \mapsto P$  is an embedding from a (partial) solution  $G$  to a solution  $P$  if for all  $a, b \in G$ :

$$\begin{array}{l}
\phi(a) = \phi(b) \Rightarrow a = b; \text{Name}(a) = \text{Name}(\phi(a)); \text{Site}(a) \subseteq \text{Site}(\phi(a)) \\
x_\iota^\lambda \in \text{Intf}(a) \Rightarrow x_{\iota'}^{\lambda'} \in \text{Intf}(\phi(a)) \textbf{ with } \lambda = \lambda' \textbf{ and } (\iota = \epsilon \textbf{ or } \iota = \iota')
\end{array}$$

**Definition 19.** Embedding between two solutions. *Name*, *Site*, and *Intf* retrieve respectively the name  $A$ , the site names  $\{x_1, \dots, x_M\}$ , and the interface  $\sigma$  of the given agent instance.

---

$$\frac{(G, r, G') \in E \quad \phi \in \text{embed}(G, P)}{P \xrightarrow{r, (\phi, G, r, G')} \phi(G') / P}$$

**Definition 20.** Kappa transition system, assuming a global fixed rule set  $E$ .

---

$$\begin{array}{ll}
I ::= & P \\
\text{process}([I_1, \dots, I_N]) & \triangleq I_1 \uplus^\alpha \dots \uplus^\alpha I_N \\
\text{species}(P) & \triangleq [I_1, \dots, I_N] \textbf{ with } \text{process}([I_1, \dots, I_N]) = P \textbf{ and} \\
& \forall I_k, \text{valid}(I_k) \textbf{ and } \forall I'_k \subset I_k, I'_k \neq I_k, \textbf{ not } \text{valid}(I'_k) \\
& \textbf{ with } \text{valid}([a_1, \dots, a_M]) \iff (x_\iota^i \in \text{Intf}(a_k) \Rightarrow \\
& \quad \exists \text{ a unique } k' \neq k, x_{\iota'}^i \in \text{Intf}(a_{k'})) \\
\text{reactions}(I_1, \tilde{I}') & \triangleq \text{mset}(\{(\bar{I}, r, w, \text{species}(P')) \mid \text{process}(\bar{I}) \xrightarrow{r, w} P'; \\
& \quad I_1 \in \bar{I}; \tilde{I}' \subseteq (\{I_1\} \cup \tilde{I}'); \#\bar{I} = \#G; w = (\phi, G, r, G')\})
\end{array}$$

**Definition 21.** Generic abstract machine instantiated for kappa, assuming a global fixed rule set  $E$ . A species  $I$  is a minimal valid solution.  $\uplus^\alpha$  stands for the multiset union where species binding states are renamed to ensure the global validity of the solution.  $\#G$  stands for the number of agents in  $G$ .

---

e.g. *phosphorylated* or *unphosphorylated*. The interactions between agents are specified by *rules*, which describe the transformation to apply on agents when a certain context matches. The context in which a rule is active consists of a multiset of *partial agents* describing the binding state and internal state of sites taking part in the transformation; all other sites should be discarded from the specification, giving a partial definition of an agent. For instance, the binding between two agents  $A(x, y), B(x, y)$  can be specified by the rule  $A(x), B(y) \rightarrow A(x^1), B(y^1)$  if the binding only depends on the sites  $x$  and  $y$  being free in  $A$  and  $B$ , respectively; 1 stands here for the binding state of the sites and is shared by precisely two sites in different agent instances. Definition 17 sums up the syntax of kappa. We denote by  $P$  a solution (multiset) of completely defined agents, and by  $G$  a solution of partially defined agents. Solutions are equal up to injective renaming of binding states.

Given a solution  $P$ , applying a rule  $(G, r, G')$  requires an *embedding*  $\phi : G \mapsto P$  (Definition 19) to map a partial agent specification to its specification in  $P$ . If such a mapping exists, the rule can be applied. By abuse of notation, the specification of agents in  $G'$  are extended using the same embedding, and are denoted by  $\phi(G')$ . As all agent instances present in  $G'$  are present in  $G$ ,  $\phi(G')$  is defined for all agents in  $G'$ , and the modifications applied on agents in  $G'$  override the mapped instances. The replacement of those agent instances in  $P$  is written  $\phi(G')/P$  (Definition 18). To ensure that the solution obtained is well-formed, binding states not present in  $G$  are mapped to fresh values by  $\phi$ . The transition system is stated in Definition 20, and a CTMC semantics can be derived from it using the following rule:

$$\frac{a = \left( \sum_{\{P^{r, \phi, (G, r, G') P'}\}} r \right) > 0}{P \xrightarrow{a} P'} \quad (22)$$

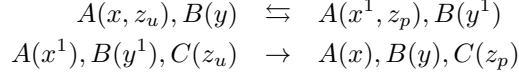
Note that creation and deletion of agents are not allowed in this setting, but the definitions can be readily extended to support this.

#### 4.3.2. Computing Species and Reactions from Processes

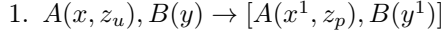
The instantiation of our generic abstract machine to kappa is given in Definition 21. A species  $I$  corresponds to a completely defined, minimal, valid solution (i.e. a kappa-species). A solution is valid if every binding state is shared by exactly two agents. Hence, extracting species from a solution splits the multi-set of agents into a multi-set of minimal valid solutions. Note that we may need to rename binding states to ensure that a given species always contains the same binding values between its agents. A solution can be recovered from a multiset of species by the multiset union where binding states are renamed to ensure the global validity of the solution. The multiset of reactions between a species  $I_1$  and a set of species  $\tilde{I}'$  is computed using the transition system from Definition 20: there is one reaction for each rule  $(G, r, G')$  for which there exists at least one embedding from a multiset of species in  $\{I_1\} \cup \tilde{I}'$  which contains  $I_1$  and which has the same number of agents as in  $G$ . The propensity of the obtained reaction  $(\bar{I}, r, \bar{J})$  corresponds to the number of possible different embeddings from  $G$  to  $\bar{I}$  which result in  $\bar{J}$  after application of the rule.

### 4.3.3. Simple Example

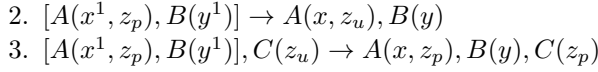
Consider the following kappa rules:



with initial solution  $E = 100 \cdot A(x, z_u), 100 \cdot B(y), 100 \cdot C(z_u)$ . Initially, the species map is  $\{A(x, z_u) \mapsto 100, B(y) \mapsto 100, C(z_u) \mapsto 100\}$  and one reaction involving the initial species is computed:



When applying this reaction, a new species  $[A(x^1, z_p), B(y^1)]$  is created with population 1. The populations of  $A(x, z_u)$  and  $B(y)$  are decreased by 1. The new species map is then  $\{A(x, z_u) \mapsto 99, B(y) \mapsto 99, C(z_u) \mapsto 100, [A(x^1, z_p), B(y^1)] \mapsto 1\}$ ; two reactions involving the new species are then computed:



If the same reaction is applied again, the population of the species  $[A(x^1, z_p), B(y^1)]$  is increased to 2 and the reaction multiset is conserved: the species map becomes  $\{A(x, z_u) \mapsto 98, B(y) \mapsto 98, C(z_u) \mapsto 100, [A(x^1, z_p), B(y^1)] \mapsto 2\}$ .

## 4.4. Instantiating the generic machine to the DSD calculus

The DNA strand displacement language (DSD) [21, 15] was developed to model DNA circuits that perform computation via strands of DNA displacing one another. The language has been used to model and analyse a broad range of circuits, some of which have been implemented experimentally, including a logic-based circuit for computing the square root of a binary number [24], and a collection of artificial neurons arranged to form an associative memory [25]. In this section we instantiate the generic abstract machine with a variant of the DSD calculus, based on [15] and [14].

### 4.4.1. Syntax and semantics of the DSD calculus

The syntax of the DSD calculus is presented in Definition 22 in terms of domains  $\mathbf{M}$ , sequences  $\mathbf{S}$ , strands  $\mathbf{A}$ , complexes  $\mathbf{G}$  and processes  $\mathbf{D}$ . A corresponding graphical representation is also given. Essentially, a process  $\mathbf{D}$  of the calculus is a collection of DNA species, where each species can be either a single strand  $\mathbf{A}$  or a complex  $\mathbf{G}$  of strands bound to each other. Each strand consists of a sequence  $\mathbf{S}$  of domains and has given orientation, where upper strands  $\langle \mathbf{S} \rangle$  are oriented to the right and lower strands  $\{\mathbf{S}\}$  are oriented to the left. Two single strands with opposing orientations can bind to each other along complementary sequences to form a double-stranded complex, with the upper strand on top and the lower strand on the bottom. A complex  $\{\mathbf{L}'\}\langle \mathbf{L} \rangle [\mathbf{S}] \langle \mathbf{R} \rangle \{\mathbf{R}'\}$  represents an upper strand  $\langle \mathbf{L} \mathbf{S} \mathbf{R} \rangle$  bound to a lower strand  $\{\mathbf{L}' \mathbf{S}^* \mathbf{R}'\}$  along the double-stranded region  $[\mathbf{S}]$ , which is formed by the sequence  $\mathbf{S}$  bound to its complementary sequence  $\mathbf{S}^*$ . The strands  $\langle \mathbf{L} \rangle, \{\mathbf{L}'\}$  and  $\langle \mathbf{R} \rangle, \{\mathbf{R}'\}$  represent overhanging upper and lower strands to the left and right of the double-stranded region, respectively. Some of these overhangs can potentially be empty, in which case they are omitted. Two complexes  $\mathbf{G}_1$  and  $\mathbf{G}_2$  can be joined along a common lower strand, written  $\mathbf{G}_1 : \mathbf{G}_2$ , or along a common upper strand, written  $\mathbf{G}_1 : : \mathbf{G}_2$ .

---

$ \begin{array}{l} D ::= \quad A \quad \text{Strand} \\ \quad \quad   \quad G \quad \text{Complex} \\ \quad \quad   \quad D1   D1 \quad \text{Composition} \\ A ::= \quad \langle S \rangle \quad \text{Upper strand} \\ \quad \quad \quad \overline{s} \\ \quad \quad   \quad \{S\} \quad \text{Lower strand} \\ \quad \quad \quad \overline{\overline{s}} \end{array} $	$ \begin{array}{l} M ::= \quad N \quad \text{Long domain} \\ \quad \quad   \quad \hat{N} \quad \text{Short domain} \\ S ::= \quad M \quad \text{Domain} \\ \quad \quad   \quad M^* \quad \text{Complement} \\ \quad \quad   \quad S1 \ S2 \quad \text{Concatenation} \\ L, R ::= \quad \emptyset \quad \text{Empty} \\ \quad \quad   \quad S \quad \text{Sequence} \end{array} $
$ \begin{array}{l} G ::= \quad \{L'\} \langle L \rangle [S] \langle R \rangle \{R'\} \\ \quad \quad \quad \begin{array}{c} \swarrow \quad \searrow \\ \langle \quad \quad \rangle \quad \langle \quad \quad \rangle \\ \quad \quad \quad \overline{s} \\ \quad \quad \quad \overline{\overline{s^*}} \\ \swarrow \quad \searrow \\ \langle \quad \quad \rangle \quad \langle \quad \quad \rangle \end{array} \\ \quad \quad   \quad G1 : G2 \quad \text{Complexes joined along lower strand} \\ \quad \quad   \quad G1 : : G2 \quad \text{Complexes joined along upper strand} \end{array} $	<p>Double stranded complex [S] with overhanging single strands <math>\langle L \rangle</math>, <math>\langle R \rangle</math> and <math>\{L'\}</math>, <math>\{R'\}</math></p>

---

**Definition 22.** Syntax of the DSD calculus, in terms of domains  $M$ , sequences  $S$ , strands  $A$ , complexes  $G$  and processes  $D$ . We omit empty upper strands  $\langle \emptyset \rangle$  and lower strands  $\{\emptyset\}$  as an abbreviation. Where present, the graphical representation below is equivalent to the program code above. We abbreviate a toehold  $\hat{N}$  to  $\mathbf{N}$  in the graphical representation, and use distinct colours for distinct toeholds.

---

In the graphical representation of complexes we omit the colons altogether and connect the strands, as shown for example in Definition 23.

Sequences  $S$  are divided into domains, where a domain  $M$  represents a nucleotide sequence with explicit information about its orientation. For example, the domain 5'-CACACA-3' denotes the nucleotide sequence CACACA oriented to the right, from its 5' end to its 3' end. This can also be written as 3'-ACACAC-5', which denotes the same nucleotide sequence oriented to the left. In general we assume that distinct domains are mapped to distinct, non-interfering nucleotide sequences. This allows us to abstract away from the underlying nucleotide sequences that occur in physical DNA strands. The complement  $M^*$  of a domain  $M$  is obtained by reversing its orientation and taking the Watson-Crick complement (C-G, T-A) of each nucleotide in the domain. For example, the complement of 5'-CACACA-3' is 3'-GTGTGT-5', such that complementary nucleotides stick together when a domain is placed on top of its complement in an opposing orientation. Similarly, the complement  $S^*$  of a sequence  $S$  is obtained by replacing each domain in  $S$  with its complement.

A domain can be either a long domain  $N$  or a short domain  $\hat{N}$ , also known as a *toehold*. The basic assumption is that toeholds are short enough to bind reversibly, while long domains are long enough to bind irreversibly. We also assume that two strands can only interact with each other via complementary toeholds. This ensures that all bindings are reversible, which limits deadlock interferences by allowing unintentional bindings to be undone. The assumption is enforced syntactically by a *well-formedness* constraint, which ensures that no long domain and its complement are simultaneously unbound anywhere in the

system. Additional details are provided in [15, 14].

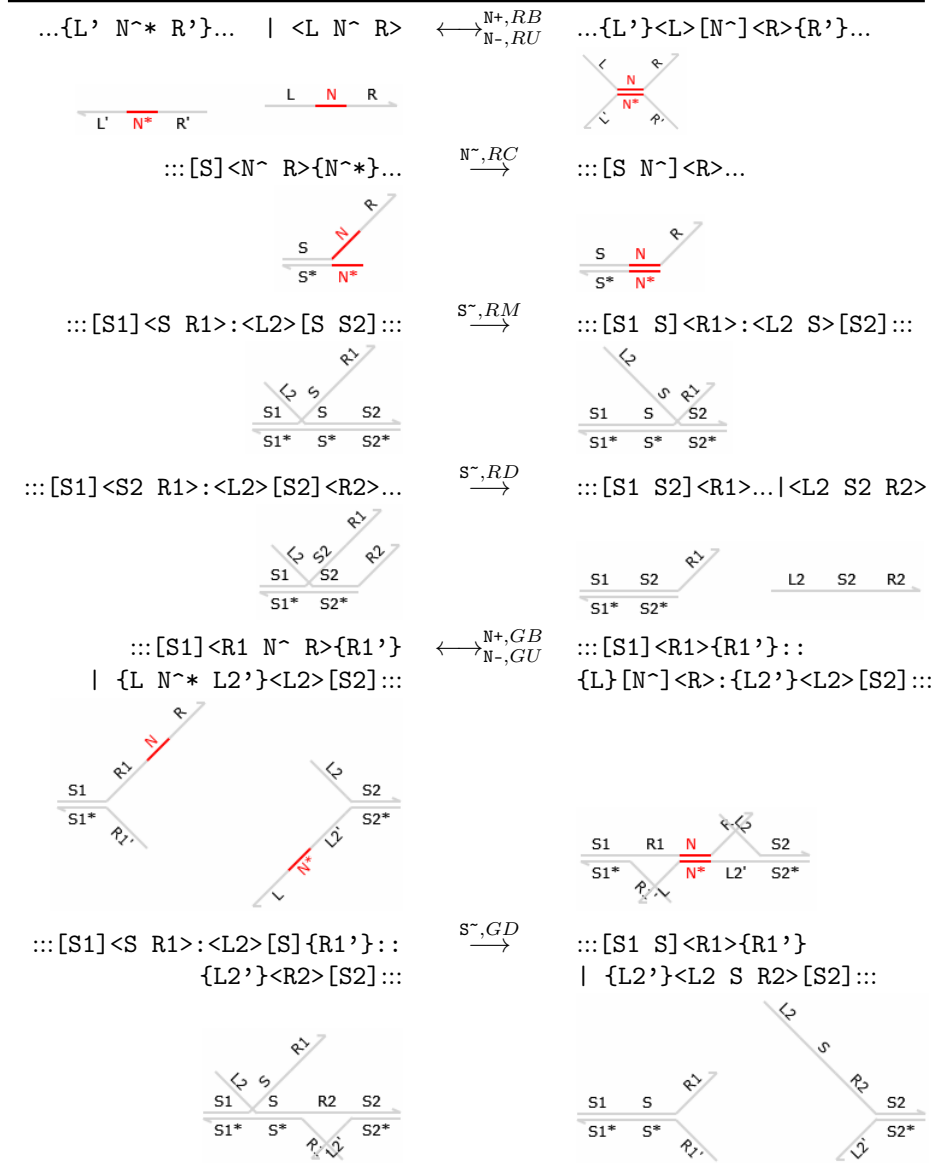
The reduction rules of the DSD calculus are presented in Definition 23. The rules are of the form  $D \xrightarrow{r,R} D'$ , which states that  $D$  can reduce to  $D'$  by performing an interaction with rate  $r$  according to rule  $R$ . The rules can be applied to strands and complexes inside larger contexts to the left and right, as described in Definition 23. Rules (RB) and (RU) define reversible toehold binding and unbinding reactions between an upper and lower strand. Rule (RC) allows complementary toeholds to bind if they are opposite each other in the same complex. Note that we do not provide versions of these three rules in which  $N$  is not a toehold, since our well-formedness assumption ensures that only complementary toeholds are exposed simultaneously. Rule (RM) defines a branch migration reaction, where a free overhanging strand partially replaces a bound strand in a complex. The sequence of the overhanging strand must match the sequence of the bound strand in order for the migration to take place. Rule (RD) can be thought of as a special case of (RM) in which the sequences match right to the end, such that the bound strand is completely displaced. Rule (GB) allows two complexes to bind on a shared toehold to form a longer complex, and rule (GU) allows the complex to break apart. Rule (GD) extends the strand displacement rule (RD) to the case where the displaced strand was previously holding two complexes together. Note that the ability to chain complexes together means that complexes can be of potentially unbounded length. As a result, it is infeasible to statically convert a process of the DSD calculus to a set of reactions for simulation. Instead, more sophisticated dynamic simulation techniques are required, which is a natural fit with the generic abstract machine presented in this paper.

In practice, since DNA strands and complexes can adopt multiple physical orientations in three-dimensional space, we define a set of equivalence rules between strands and complexes in Definition 25. The first two rules state that strands and complexes are equal up to rotation, while the next two rules account for the fact that the same physical complex can be written in multiple ways using the textual syntax. Additional reduction rules are presented in Definition 26, which allow a reduction to take place up to re-ordering of processes, and when gates and strands are reversed or complemented.

The CTMC semantics for the DSD calculus is given by the following rule, where the rate of a transition from process  $D$  to  $D'$  is given by the sum of the rates of all the distinct reactions by which this transition can occur:

$$\frac{a = \left( \sum_{\{\lambda, w \mid D \xrightarrow{\text{exp}(\lambda), w} D'\}} \lambda \right) > 0}{D \xrightarrow{a} D'} \quad (23)$$

The reaction identifier  $w$  is used to identify distinct reactions, and is obtained by first sorting the strands and complexes in process  $D$ , for example in lexicographic order, and then assigning a position  $i$  to each strand and complex. The reaction identifier  $w$  consists of the positions of the strands and complexes directly involved in the reduction, and is of the form  $(i, j)$  for binary reductions and  $(i)$  for unary reductions. As in Sec. 4.1, we use the CTMC semantics to derive a corresponding CTMC for a given process  $D$ . We recursively enumerate the set of transitions  $D \xrightarrow{a} D'$  for each distinct process  $D'$ , according to (3), where processes are considered to be distinct up to structural congruence.



**Definition 23.** Elementary reduction rules of the DSD calculus. For each rule, the graphical representation below is equivalent to the program code above. We let  $S^{\sim}$  denote the migration rate of a sequence  $S$ , and  $N^+$  and  $N^-$  denote the binding and unbinding rates of a toehold  $N^{\wedge}$ , respectively. We let  $\text{fst}(S)$  and  $\text{lst}(S)$  denote the first and last domain in a sequence  $S$ , respectively, and we assume that  $\text{fst}(R2) \neq \text{fst}(S2)$  for rule (RM). This ensures that branch migration is maximal along a given sequence and that rules (RM) and (RD) are mutually exclusive. We define syntax abbreviations for contexts, where  $H$  denotes a possibly empty complex, and  $(\circ)$  denotes either upper or lower concatenation. We write  $\dots [S]$  as an abbreviation for  $H1 \circ_1 \{L3\} \langle L3' \rangle [S]$  and  $[S] \dots$  for  $[S] \langle R3 \rangle \{R3'\} \circ_2 H2$ . We also write  $\dots \{S\} \dots$  for  $H1 : \{S\} : H2$  and  $G \dots$  for  $G : H2$ . We assume that concatenation with an empty complex has no effect.

---


$$\begin{array}{lcl}
\text{rotate}(\langle S \rangle) & \triangleq & \{\text{rev}(S)\} \\
\text{rotate}(\{S\}) & \triangleq & \langle \text{rev}(S) \rangle \\
\text{rotate}(\{L'\} \langle L \rangle [S] \langle R \rangle \{R'\}) & \triangleq & \{\text{rev}(R)\} \langle \text{rev}(R') \rangle [S^*] \\
& & \langle \text{rev}(L') \rangle \{\text{rev}(L)\} \\
\text{rotate}(G1 : G2) & \triangleq & \text{rotate}(G2) : : \text{rotate}(G1) \\
\text{rotate}(G1 : : G2) & \triangleq & \text{rotate}(G2) : \text{rotate}(G1) \\
\text{rev}(\langle S \rangle) & \triangleq & \langle \text{rev}(S) \rangle \\
\text{rev}(\{S\}) & \triangleq & \{\text{rev}(S)\} \\
\text{rev}(\{L'\} \langle L \rangle [S] \langle R \rangle \{R'\}) & \triangleq & \{\text{rev}(R')\} \langle \text{rev}(R) \rangle [\text{rev}(S)] \\
& & \langle \text{rev}(L) \rangle \{\text{rev}(L')\} \\
\text{rev}(G1 : G2) & \triangleq & \text{rev}(G2) : : \text{rev}(G1) \\
\text{rev}(G1 : : G2) & \triangleq & \text{rev}(G2) : \text{rev}(G1)
\end{array}$$

**Definition 24.** Rotating and reversing strands and complexes in the DSD calculus, where  $\text{rev}(S)$  reverses the order of domains in the sequence  $S$ .

---

$$\begin{array}{lcl}
G & \equiv & \text{rotate}(G) \\
A & \equiv & \text{rotate}(A) \\
::: [S1] \langle R1 \rangle \{R\} S : \{L\} \langle L2 \rangle [S2] ::: & \equiv & ::: [S1] \langle R1 \rangle \{R\} : \{S\} L \langle L2 \rangle [S2] ::: \\
::: [S1] \langle R\ S \rangle \{R1\} : : \{L2\} \langle L \rangle [S2] ::: & \equiv & ::: [S1] \langle R \rangle \{R1\} : : \{L2\} \langle S\ L \rangle [S2] ::: \\
D \equiv D' \Rightarrow D | D2 & \equiv & D' | D2
\end{array}$$

**Definition 25.** Structural congruence rules of the DSD calculus, which rely on the definitions of contexts from Definition 23. Parallel composition (1) is assumed to be commutative and associative, and structural congruence is assumed to be reflexive, symmetric and transitive.

---

$$\begin{array}{lcl}
D \xrightarrow{r,R} D' \Rightarrow \text{rev}(D) \xrightarrow{r,R} \text{rev}(D') \\
D \xrightarrow{r,R} D' \Rightarrow \text{com}(D) \xrightarrow{r,R} \text{com}(D') \\
D \xrightarrow{r,R} D' \Rightarrow D | D2 \xrightarrow{r,R} D' | D2 \\
D2 \equiv D \xrightarrow{r,R} D' \equiv D2' \Rightarrow D2 \xrightarrow{r,R} D2'
\end{array}$$

**Definition 26.** Inductive reduction rules of the DSD calculus. We assume that  $\text{rev}(D)$  reverses all of the strands and complexes in process  $D$ , while  $\text{com}(D)$  complements all of the sequences in process  $D$ .

---

#### 4.4.2. Computing Species and Reactions from Processes

Functions for converting between species and processes in the DSD calculus are given in Definition 27, where a species is a strand or complex, and a process is a parallel composition of species.

#### 4.4.3. Example

We illustrate the application of the generic abstract machine to the DSD calculus with a simple example of a logical AND gate made of DNA. In this example, two inputs  $\langle 1\ 2 \rangle$  and  $\langle 3\ 4 \rangle$  cooperate to displace the output  $\langle 2$

---


$$\begin{aligned} \text{species}(I_1 \mid \dots \mid I_M) &\triangleq [I_1, \dots, I_M] \\ \text{process}([I_1, \dots, I_M]) &\triangleq I_1 \mid \dots \mid I_M \end{aligned}$$


---

**Definition 27.** Converting between processes and species in DSD, where a species  $I$  is a strand  $A$  or a complex  $G$ .

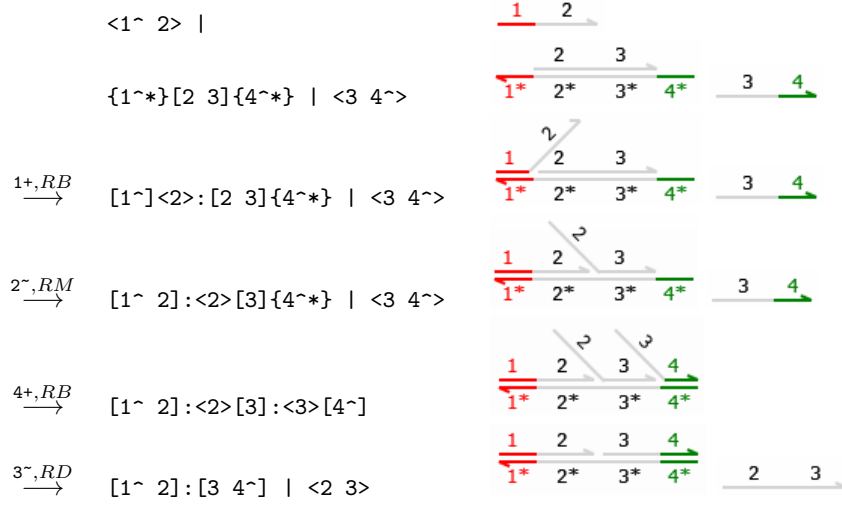
---

$$\begin{aligned} \text{reactions}(I, \tilde{I}') &\triangleq \text{unary}(I) \cup \text{binary}(I, \tilde{I}') \\ \text{unary}(I) &\triangleq \{([I], r, \bar{I}') \mid [I] \xrightarrow{r, R} \bar{I}'\} \\ \text{binary}(I_1, \tilde{I}') &\triangleq \{([I_1, I_2], r, \bar{I}'') \mid I_2 \in \tilde{I}'; [I_1, I_2] \xrightarrow{r, R} \bar{I}''; R \in \{RB, GB\}\} \end{aligned}$$

**Definition 28.** Compiling species to reactions in DSD. We write  $\bar{I} \xrightarrow{r, R} \bar{I}'$  if  $(I_1 \mid \dots \mid I_M) \xrightarrow{r, R} (I'_1 \mid \dots \mid I'_N)$  holds, where  $\bar{I} = [I_1, \dots, I_M]$  and  $\bar{I}' = [I'_1, \dots, I'_N]$ .

---

3> from the initial gate  $\{1^{\wedge*}\} [2 \ 3] \{4^{\wedge*}\}$ . A possible sequence of reductions is shown below, starting from an initial process with one copy of each input and one gate:



If we start with 100 copies of each input and 100 gates we obtain the machine term  $(0, S, R)$ , where  $S$  and  $R$  are as follows and where  $I_1 = \langle 1^{\wedge} \ 2 \rangle$ ,  $I_2 = \langle 3 \ 4^{\wedge} \rangle$ ,  $O = \langle 2 \ 3 \rangle$ ,  $G_6 = \{1^{\wedge*}\} [2 \ 3] \{4^{\wedge*}\} : \langle 3 \rangle [4^{\wedge}]$  and  $G_1, \dots, G_5$  correspond to the complexes from the above sequence of reductions, in order of appearance:

$$\begin{aligned} S &= \{I_1 \mapsto 100, I_2 \mapsto 100, G_1 \mapsto 100\} \\ R &= [(\{I_1, G_1\}, 1^+, [G_2]) \mapsto (10^4 \cdot 1^+, t_1) \\ &\quad , (\{I_2, G_1\}, 4^+, [G_6]) \mapsto (10^4 \cdot 4^+, t_2)] \end{aligned}$$

By the end of the simulation all of the inputs are consumed and converted into the output, with  $S = \{G_5 \mapsto 100, O \mapsto 100\}$  and  $R = []$ , assuming that reactions with propensity zero are garbage collected.

---


$$\begin{aligned} \llbracket P \rrbracket_t &\triangleq P \oplus (t, \emptyset, \emptyset) \\ \llbracket t, S, R \rrbracket &\triangleq \text{process}(S) \end{aligned}$$

**Definition 29.** Translating between calculus processes  $P$  and machine terms  $T$ . The function  $\llbracket P \rrbracket_t$  encodes a process  $P$  to a corresponding term at a given time  $t$ . The function  $\llbracket T \rrbracket$  decodes a term  $T$  to a corresponding process, and is parameterised by the calculus-specific function  $\text{process}(S)$ . For a given species map  $S = \{I_1 \mapsto i_1, \dots, I_N \mapsto i_N\}$ , with a slight abuse of notation we also allow  $S$  to stand for the multiset of species  $\{(I_1, i_1), \dots, (I_N, i_N)\}$ .

---

$$\begin{aligned} \text{reactionset}(S, S') &\triangleq \{(\bar{I}, r, \bar{I}') \mid (\bar{I}, r, \bar{I}') \in \text{reactionset}(S); S' = (S \setminus \bar{I}) \uplus \bar{I}'\} \\ \text{reactionset}(S) &\triangleq \text{merge}(\{(O_i, k_i) \mid (O_i, k_i) \in \text{reactions}(I, \tilde{I}'); \text{set}(S) = \{I\} \cup \tilde{I}'\}) \end{aligned}$$

**Definition 30.** Additional functions used for the proofs. We write  $S$  as syntactic sugar for the multiset of species corresponding to  $S$ , as described above. The  $\text{merge}$  function for combining multiple identical reactions is as defined in Definition 3.

---

## 5. Correctness

In this section we prove the correctness of the generic abstract machine for process calculi with Markovian semantics and exact mass-action simulation methods. The proof is given in terms of equivalence between Continuous Time Markov Chains (CTMCs), following the approach outlined in [4]. It is sufficient to show that the CTMC generated by the calculus semantics is the same as the one generated by the abstract machine. Furthermore, as discussed for example in [10], all exact stochastic simulation algorithms with mass action kinetics select reactions and times according to the correct probability distributions, so that the probability of generating a given trajectory with the simulation algorithm is exactly the probability that would come out of the solution of the Master Equation. Thus, it is sufficient to prove the correspondence between the CTMC of a given process calculus and the CTMC generated by a given simulation method with mass action kinetics. In this case we use the Direct Method.

### 5.1. Generic Statement of Correctness Theorems

We define a function  $\llbracket P \rrbracket_t$  which encodes a process  $P$  in the calculus into a corresponding term in the abstract machine at a given simulation time  $t$ . We also define a function  $\llbracket T \rrbracket$  which decodes a term  $T$  in the abstract machine into a corresponding process in the calculus. The encoding and decoding functions between the calculus  $\mathcal{C}$  and the machine  $\mathcal{CM}$  are stated in Definition 29. The definition of decoding requires that the user defines an additional  $\text{process}$  function for their calculus, which translates a multiset of species back into a process. This can be thought of as an inverse to the  $\text{species}$  function—indeed, we must establish this fact for a particular encoding to be correct.

The correctness of an encoding is established by demonstrating a reduction equivalence between the calculus and the machine. In order to preserve the cor-

respondence, we assume a notion of structural congruence for machine terms, where terms are structurally congruent up to renaming of definitions, garbage-collection of unused definitions and structural congruence of processes. We also assume that the structural congruence on machine terms allows additional species with population 0 and additional reactions with propensity 0. As before, we assume a top-level environment  $E$  of definitions throughout. It suffices to prove correctness with respect to the Direct Method of stochastic simulation (Definition 3), as the Direct Method and the Next Reaction Method are equivalent and are known to be correct.

In this section we present a generic technique for proving correctness of some calculus  $C$ , which comes equipped with *species*, *reactions* and *process* functions which induce an abstract machine instantiation  $CM$ . In order to exploit our generic proof one must prove the following propositions.

**Proposition 31** (Species correctness).  $\forall P, \bar{I}$ :  $process(species(P)) = P$  and  $species(process(\bar{I})) = \bar{I}$ .

**Proposition 32** (Reaction correctness).  $\forall S, S', a$ :  $process(S) \xrightarrow{a} process(S')$  iff  $a = \sum_{\{O \in reactionset(S, S')\}} propensity(O, S)$  and  $a > 0$ .

Assuming that these propositions all hold, we now present general proofs of soundness and completeness for calculus encodings, which relate the CTMC semantics of the abstract machine (3) and the CTMC semantics of the calculus. As mentioned in Definition 30 we write  $S$  not only for a species population map but also for the corresponding multiset of species. We say that an abstract machine term is *well-formed*, and write  $wellformed(T)$ , if  $T = (t, S, R)$  and  $dom(R) = reactionset(S)$ . The structural congruence on abstract machine terms described above allows us to ignore any additional reactions  $O$  for which  $propensity(O, S) = 0$ . It is straightforward to show that  $(P)_t$  is well-formed for all processes  $P$  and that well-formedness is preserved by abstract machine reductions. The generic correctness theorems are now as follows.

**Theorem 33** (Generic soundness).  $\forall T, T', a$ : if  $T \in CM$  and  $wellformed(T)$  and  $T \xrightarrow{a} T'$  then  $\llbracket T \rrbracket \xrightarrow{a} \llbracket T' \rrbracket$ .

*Proof.* Assume that  $T \xrightarrow{a} T'$  with  $T = (t, S, R)$  and  $T' = (t', S', R')$ . By definition of abstract machine CTMC reduction (3) we have that  $a = \sum_{\{b, O | T \xrightarrow{b, O} T'\}} b$  with  $a > 0$ .

If  $T \xrightarrow{b, O} T'$  with  $O = (\bar{I}, r, \bar{I}')$  then by definition of abstract machine reduction (1) we have that  $T' = \bar{I}' \oplus ((t', S, R) \ominus \bar{I})$  for some  $t'$ . By definition of species addition ( $\oplus$ ) and removal ( $\ominus$ ) we have that  $S' = (S \setminus \bar{I}) \uplus \bar{I}'$ . By definition of the *next* function for the Direct Method (Definition 3) we also have that  $O \in dom(R)$  and  $b = propensity(O, S) > 0$ . Since  $T$  is well-formed, by definition of *wellformed* we get that  $dom(R) = reactionset(S)$  and therefore  $O \in reactionset(S)$ . Since  $S' = (S \setminus \bar{I}) \uplus \bar{I}'$ , by definition of  $reactionset(S, S')$  we have that  $O \in reactionset(S, S')$ . Therefore  $a = \sum_{\{O \in reactionset(S, S')\}} propensity(O, S)$  with  $a > 0$ . Therefore by Proposition 32 we have that  $process(S) \xrightarrow{a} process(S')$ .

By definition of  $\llbracket \cdot \rrbracket$  we have that  $\llbracket T \rrbracket = \llbracket (t, S, R) \rrbracket = process(S)$  and  $\llbracket T' \rrbracket = \llbracket (t', S', R') \rrbracket = process(S')$ . Therefore  $\llbracket T \rrbracket \xrightarrow{a} \llbracket T' \rrbracket$  holds as required.  $\square$

**Theorem 34** (Generic completeness).  $\forall P, P', a$ : if  $P \in C$  and  $P \xrightarrow{a} P'$  then  $(P)_t \xrightarrow{a} (P')_{t'}$  for some  $t, t'$ .

*Proof.* Assume that  $P \xrightarrow{a} P'$  with  $S = \text{species}(P)$  and  $S' = \text{species}(P')$ . By Proposition 31 we have that  $\text{process}(S) \xrightarrow{a} \text{process}(S')$ . Therefore by Proposition 32 we have that  $a = \sum_{\{O \in \text{reactionset}(S, S')\}} \text{propensity}(O, S)$ .

Let  $\llbracket P \rrbracket_t = (t, S, R) = T$  and  $\llbracket P' \rrbracket_{t'} = (t', S', R') = T'$ . By definitions of  $\llbracket \cdot \rrbracket_t$  and the addition function ( $\oplus$ ) and the *init* function for the Direct Method (Definition 3) we have that  $\text{dom}(R) = \text{reactionset}(S)$  and  $R(O) = \text{propensity}(O, S)$  for each reaction  $O$  in  $\text{dom}(R)$ . Similarly, we have  $\text{dom}(R') = \text{reactionset}(S')$  and  $R'(O') = \text{propensity}(O', S')$  for each reaction  $O'$  in  $\text{dom}(R')$ .

If  $O \in \text{reactionset}(S, S')$  with  $O = (\bar{I}, r, \bar{I}')$  then by definition of *reactionset* we have that  $S' = (S \setminus \bar{I}) \uplus \bar{I}'$ . Therefore by definition of species addition ( $\oplus$ ) and removal ( $\ominus$ ) we have that  $T' = \bar{I}' \oplus ((t', S, R) \ominus \bar{I})$ . Therefore by definition of abstract machine reduction (1) we have that  $T \xrightarrow{b, O} T'$  with  $b = \text{propensity}(O, S)$ . Therefore  $a = \sum_{\{b, O \mid T \xrightarrow{b, O} T'\}} b$  with  $a > 0$ . Therefore by abstract machine CTMC reduction (3) we have that  $T \xrightarrow{a} T'$ . Therefore  $\llbracket P \rrbracket_t \xrightarrow{a} \llbracket P' \rrbracket_{t'}$  holds for some  $t, t'$ , as required.  $\square$

We now prove correctness results for the stochastic pi-calculus, the bioambient calculus, the kappa calculus, and the DSD calculus, by proving Proposition 31 and Proposition 32 for each calculus in turn.

## 5.2. Stochastic Pi-Calculus

To prove that the encoding of stochastic pi-calculus is correct we rely on the fact that every process is structurally congruent to its normal form, as shown in Proposition 12. Given the definition of *species*( $P$ ) in Definition 11, it follows that if  $P \equiv P'$  then  $\text{species}(P) \equiv \text{species}(P')$ . This is important as it shows that we can convert back and forth between a species and its normal form representation without losing any information on the corresponding species in the generic abstract machine.

*Proof of species correctness.* It is trivial to see that Proposition 31 holds for stochastic pi-calculus because the *species* function simply turns a parallel composition of species (from the normal form) into the corresponding multiset, whereas the *process* function simply turns the multiset back into a parallel composition which is structurally congruent to the original process.  $\square$

*Proof of reaction correctness.* We know that  $\text{process}(S) \xrightarrow{a} \text{process}(S')$  holds iff  $a = \sum_{\{\lambda, w \mid \text{process}(S) \xrightarrow{\exp(\lambda), w} \text{process}(S')\}} \lambda$  with  $a > 0$ .

We will write  $\text{indices}(P, P')$  for the set  $\{w \mid \exists r. P \xrightarrow{r, w} P'\}$ . In the stochastic pi-calculus, indices  $w$  can either be a single index  $i$  for a unary delay or a pair index  $(i_1, i_2)$  for a binary communication or binding. Given an index  $w$ , we define helper functions  $\text{src}(w)$ ,  $\text{tgt}(w)$  and  $\text{rate}(w)$  as follows, where we write  $i \in I$  to mean that the action index  $i$  appears in the choice corresponding to species  $I$  (this is well-defined since each index only appears once).

$$\begin{aligned} \text{src}(w) &\triangleq \begin{cases} I & \text{if } w = i \text{ and } i \in I \\ I_1 \mid I_2 & \text{if } w = (i_1, i_2) \text{ and } i_1 \in I_1 \text{ and } i_2 \in I_2 \end{cases} \\ \text{tgt}(w) &\triangleq P' \text{ if } \text{src}(w) \xrightarrow{\exp(\lambda), w} P' \\ \text{rate}(w) &\triangleq \exp(\lambda) \text{ if } \text{src}(w) \xrightarrow{\exp(\lambda), w} \text{tgt}(P') \end{aligned}$$

We say that  $w \approx w'$  holds iff  $src(w) = src(w')$ ,  $tgt(w) = tgt(w')$  and  $rate(w) = rate(w')$ . We write  $indexsets(P, P')$  for the set of  $\approx$ -equivalence classes in  $indices(P, P')$ . We will write  $src(\tilde{w})$ ,  $tgt(\tilde{w})$  and  $rate(\tilde{w})$  to mean  $src(w)$ ,  $tgt(w)$  and  $rate(w)$  for some/any  $w \in \tilde{w}$ .

Now, we observe that  $process(S) \xrightarrow{\exp(\lambda), w} process(S')$  holds iff  $src(w) \xrightarrow{rate(w), w} tgt(w)$ , where  $S' = (S \setminus species(src(w))) \uplus species(tgt(w))$ . If we let  $reaction(\tilde{w}) = (species(src(\tilde{w})), rate(\tilde{w}), species(tgt(\tilde{w})))$  then it is clear from these definitions and the definitions from Definition 11 that

$$\{reaction(\tilde{w}) \mid \tilde{w} \in indexsets(process(S), process(S'))\} = reactionset(S, S')$$

i.e. that every  $\approx$ -equivalence class of indices in  $indices(process(S), process(S'))$  corresponds to a particular reaction in the abstract machine. Writing  $p(S)$  to abbreviate  $process(S)$  and  $size(\tilde{w})$  for the number of indices in  $\tilde{w}$ , we get that

$$\begin{aligned} & \sum_{\{\lambda, w \mid p(S) \xrightarrow{\exp(\lambda), w} p(S')\}} \\ &= \sum_{w \in indices(p(S), p(S'))} rate(w) \\ &= \sum_{\tilde{w} \in indexsets(p(S), p(S'))} (rate(\tilde{w}) \times size(\tilde{w})) \\ &= \sum_{\{reaction(\tilde{w}) \mid \tilde{w} \in indexsets(p(S), p(S'))\}} propensity(reaction(\tilde{w}), S) \\ &= \sum_{O \in reactionset(S, S')} propensity(O, S). \end{aligned}$$

Hence we get that  $a = \sum_{O \in reactionset(S, S')} propensity(O, S)$  with  $a > 0$ .  $\square$

### 5.3. Bioambient Calculus

In the case of bioambient calculus, the proof of Proposition 31 is less straightforward, because the definitions of the *species* and *process* functions must take care to handle ambient locations correctly. Although the results in this section apply to the bioambient calculus without the **merge** action, they can be readily extended to incorporate this action.

*Proof of species correctness.* By induction on the definitions of the *species* and *process* functions in Definition 15 and Definition 16. The most delicate case is for ambients, as we must show that the ambient locations and the structure of the ambient tree are preserved. In computing  $species(P, a)$  the  $a$  identifier is added to all instances at the current position in the hierarchy to indicate their location. When an ambient  $\boxed{P'}^{a'}$  is encountered, the location species  $(a', a)$  is created and we use the location  $a'$  instead of  $a$  as we recurse within  $P'$ . When we convert back to processes, the definition of  $process(\bar{I}, a)$  collects all instances at the current location in the ambient hierarchy, then uses the location species to recursively rebuild the child trees within appropriately labelled ambients.  $\square$

*Proof of reaction correctness.* This proof is similar to the proof of reaction correctness for the stochastic pi-calculus outlined above. The main difference is that the definition of the *src* and *tgt* functions must reflect the fact that indices on bioambient reactions are different and more complicated than those for the stochastic pi-calculus. In particular, we add an additional argument  $\tilde{L}$  to *src*

and  $tgt$  which is the set of all location species present in the system:

$$src(w, \tilde{L}) \triangleq \begin{cases} I & \text{if } w = i \text{ and } i \in I \\ I_1 \mid I_2 & \text{if } w = (i_1, i_2) \text{ and } i_1 \in I_1 \text{ and } i_2 \in I_2 \\ I_1 \mid I_2 \mid (a, b) & \text{if } w = (i_1, a, i_2) \text{ and } i_1 \in X(\tilde{n})^a \\ & \text{and } i_2 \in Y(\tilde{m})^b \text{ and } a \neq b \text{ and } (a, b) \in \tilde{L} \\ I_1 \mid I_2 \mid (a, d) \mid (b, c) & \text{if } w = (i_1, a, i_2, b) \text{ and } i_1 \in X(\tilde{n})^a \\ & \text{and } i_2 \in Y(\tilde{m})^b \text{ and } \{(a, d), (b, c)\} \subseteq \tilde{L} \\ & \text{and } a \neq b \text{ and } a \neq c \text{ and } (d = b \text{ or } d = c) \end{cases}$$

$$tgt(w, \tilde{L}) \triangleq P' \text{ if } src(w, \tilde{L}) \xrightarrow{\exp(\lambda), w} P'$$

In the final two cases for  $src(w, \tilde{L})$ , we also allow for the symmetric case where  $i_1 \in Y(\tilde{m})^b$  and  $i_2 \in X(\tilde{n})^a$ . In order to reconstruct a process from  $tgt(w, \tilde{L})$  we also need the correct ambient identifier to pass to the *species* function. To this end we introduce a new function  $loc(w, \tilde{L})$  which computes the correct location for a given index  $w$ , and is defined as follows:

$$loc(w, \tilde{L}) \triangleq \begin{cases} a & \text{if } w = i \text{ and } i \in X(\tilde{n})^a \\ a & \text{if } w = (i_1, i_2) \text{ and } i_1 \in X(\tilde{n})^a \text{ and } i_2 \in Y(\tilde{m})^a \\ b & \text{if } w = (i_1, a, i_2) \text{ and } i_1 \in X(\tilde{n})^a \\ & \text{and } i_2 \in Y(\tilde{m})^b \text{ and } a \neq b \text{ and } (a, b) \in \tilde{L} \\ c & \text{if } w = (i_1, a, i_2, b) \text{ and } i_1 \in X(\tilde{n})^a \\ & \text{and } i_2 \in Y(\tilde{m})^b \text{ and } \{(a, d), (b, c)\} \subseteq \tilde{L} \\ & \text{and } a \neq b \text{ and } a \neq c \text{ and } (d = b \text{ or } d = c) \end{cases}$$

As before, we also allow for the symmetric case where  $i_1 \in Y(\tilde{m})^b$  and  $i_2 \in X(\tilde{n})^a$  in the final two cases here. When we come to translate  $tgt(w, \tilde{L})$  back into species we actually compute  $species(tgt(w, \tilde{L}), loc(w, \tilde{L}))$ , using the definition of  $species(P, a)$  from Definition 15, which ensures that the instances are labelled with the correct ambient identifier. These definitions follow those from Definition 16 and we handle merged reactions by summing over all indices  $w$ . Hence we can use a proof sketch similar to that for reaction correctness in the stochastic pi-calculus presented above, to show that  $process(S) \xrightarrow{a} process(S')$  iff  $a = \sum_{O \in \text{reactionset}(S, S')} propensity(O, S)$  with  $a > 0$ .  $\square$

#### 5.4. Kappa Calculus

We present proofs of the requisite correctness lemmas for the kappa instantiation.

*Proof of species correctness.* This follows directly from *species* and *process* specification in Definition 21.  $\square$

*Proof of reaction correctness.* We know that  $process(S) \xrightarrow{a} process(S')$  iff  $a = \sum_{\{process(S) \xrightarrow{r, \phi, (G, r, G')} process(S')\}} r$  with  $a > 0$ . Given a rule  $(G, r, G')$ , every different embedding  $\phi \in \text{embed}(G, process(S))$  contributes to the propensity of

the obtained reaction. Let us define the multiset of reactions  $F$ , where one reaction is present for each different rule-embedding pair which generates it:

$$F = mset(\{(species(\phi(G)), r, w, species(\phi(G')/\phi(G))) \mid (G, r, G') \in E; \\ \phi \in embed(G, process(S)); \phi(G')/process(S) = process(S'); \\ w = (\phi, G, r, G')\})$$

For all  $((\bar{I}, r, \bar{I}'), k) \in F$ , we know that  $(\bar{I}, r, \bar{I}') \in reactionset(S, S')$ . Given a rule  $(G, r, G')$  and a minimal valid solution  $G^a \subseteq G$ , we write  $\{\phi_1, \dots, \phi_L\} \subseteq embed(G, process(S))$  for the set of embeddings that are different only in the mapping of the species  $G^a$ , and are such that  $\phi_1(G) = \dots = \phi_L(G) = process(\bar{I})$ . We get that  $species(\phi_1(G^a)) = \dots = species(\phi_L(G^a)) = I_a$  with  $I_a \in \bar{I}$ ; therefore  $L = S(I_a)$ . By induction, we obtain  $r \times k = propensity((\bar{I}, r, \bar{I}'), S)$ . Hence it follows that  $\sum_{((\bar{I}, r, \bar{I}'), k) \in F} (r \times k) = \sum_{\{process(S)^{r, \phi, (G, r, G')}, process(S')\}} r$ , which is equivalent to  $a = \sum_{\{O \in reactionset(S, S')\}} propensity(O, S)$  with  $a > 0$ .  $\square$

### 5.5. DSD Calculus

Finally, we outline proofs of the requisite correctness lemmas for the DSD instantiation.

*Proof of species correctness.* This follows directly from the definitions of the *species* and *process* functions in Definition 27.  $\square$

*Proof of reaction correctness.* This follows directly the definitions of the *reactions* function in Definition 28 and the CTMC semantics in (23). Essentially, the transitions derived by the CTMC semantics for a given process correspond exactly to the reactions derived by the reactions function, such that the transition rates correspond to the propensities of the reactions.  $\square$

## 6. Multi-Calculus Models

Up to now we have discussed models where the behaviour of the entire system is derived using the rules of a single process calculus. However, the generic abstract machine defined in Sec. 2 is general enough to allow multi-calculus (heterogeneous) models to be constructed from components written using multiple different calculi. This approach allows us to choose the most appropriate domain-specific language to formalise each different aspect of the system.

### 6.1. Defining a Multi-Calculus Model

For a calculus  $C$  we will write  $\mathbb{P}_C$  for the type of processes in that calculus and  $\mathbb{I}_C$  for the corresponding type of species. The definition of reactions as a tuple  $(\bar{I}, r, \bar{I}')$  induces a type  $\mathbb{R}_C$  of reactions for calculus  $C$  in terms of the associated species type  $\mathbb{I}_C$ . Writing  $\mathcal{P}_{\text{fin}}(X)$  for the set of all finite subsets of  $X$ , the types of the *species* and *reactions* functions for calculus  $C$  can be summarised as follows.

$$\begin{aligned} species_C & : \mathbb{P}_C \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{I}_C) \\ reactions_C & : \mathbb{I}_C \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{I}_C) \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{R}_C) \end{aligned}$$

Now suppose that we wish to define a heterogeneous model using the finite set of sub-calculi  $\overline{C} \equiv \{C_1, \dots, C_n\}$ , each of which is defined as above. Note that we will refer to  $\overline{C}$  as a calculus in the same way as  $C_i$ . Assuming that the process types and species types for the various sub-calculi are all disjoint, we define process and species types for  $\overline{C}$  in terms of those for the sub-calculi, as follows.

$$\begin{aligned} \mathbb{I}_{\overline{C}} &\triangleq \mathbb{I}_{C_1} \uplus \dots \uplus \mathbb{I}_{C_n} \\ \mathbb{P}_{\overline{C}} &\triangleq \mathbb{P}_{C_1} \times \dots \times \mathbb{P}_{C_n} \times \mathcal{P}_{\text{fin}}(\mathbb{R}_{\overline{C}}) \end{aligned}$$

A species in calculus  $\overline{C}$  is simply a species in one of its sub-calculi. Note that since the species types are assumed to be disjoint, we can always tell which calculus a particular species came from. A process in calculus  $\overline{C}$  consists of a process in each of the sub-calculi  $C_i$  along with a set of reactions in the  $\overline{C}$  calculus. Since the species in the  $\overline{C}$  calculus could be from any of the sub-calculi, these reactions are the only way that species from different calculi can interact (the rules of each sub-calculus can, by definition, only produce species from that calculus). We refer to these reactions as *glue reactions* as they provide the interface to link the various components together. The modeller must supply the glue reactions up front, but this is not a problem in practise as the glue reactions should arise naturally from the structure of the heterogeneous model.

Before we proceed, we need to extract the species for a given calculus from the multi-calculus species type. If  $\tilde{I} \in \mathbb{I}_{\overline{C}}$  then define  $\pi_{C_i}(\tilde{I}) = \{I \mid I \in \tilde{I}; I \in \mathbb{I}_{C_i}\}$ . It follows that  $\pi_{C_i}(\tilde{I}) \in \mathbb{I}_{C_i}$  for all  $i$  and furthermore that  $\{\pi_{C_i}(\tilde{I}) \mid i \in \{1, \dots, n\}\}$  partitions  $\tilde{I}$ . Now, let  $P \in \mathbb{P}_{\overline{C}}$  stand for a multi-calculus process such that  $P \equiv (P_{C_1}, \dots, P_{C_n}, G)$  where  $G$  is the (user-specified) set of glue reactions. Then, the *species* and *reactions* functions for the  $\overline{C}$  calculus are defined as follows.

$$\begin{aligned} \text{species}_{\overline{C}}(P) &\triangleq \text{species}_{C_1}(P_1) \uplus \dots \uplus \text{species}_{C_n}(P_n) \\ \text{reactions}_{\overline{C}}(I, \tilde{I}) &\triangleq \text{reactions}_{C_i}(I, \pi_{C_i}(\tilde{I})) \uplus \text{glue}(I, \tilde{I}, O) \text{ if } I \in \mathbb{I}_{C_i} \end{aligned}$$

where the *glue* function is defined as follows.

$$\text{glue}(I, \tilde{I}, G) \triangleq \{(O, 1) \mid O \in G; I \in \text{reactants}(O); \text{reactants}(O) \subseteq \tilde{I} \cup \{I\}\}$$

We get the starting species for the  $\overline{C}$  calculus by simply taking the starting species in each individual sub-calculus. To compute the reactions between species  $I$  and the set of existing species  $\tilde{I}$  we first use the *reactions* function from the appropriate sub-calculus to compute all possible reactions within that calculus. However, we must also include inter-calculus reactions—we compute these by looking through the set  $G$  of glue reactions to find any reactions for which we know about all of the reactants, and where the new species  $I$  is one of the reactants. (The second criterion ensures that each glue reaction is only added once because each species  $I$  is only considered once by the compiler during a simulation run.) These inter-calculus reactions allow the sub-models written in different domain-specific languages to interact with each other across the boundaries of their respective species types. We can now use the techniques described above to derive a simulator for multi-calculus models containing components written in different calculi.

### 6.2. Simple Example

We now consider a simple example of a multi-calculus model, which will use components written in the stochastic pi-calculus and the kappa calculus. Our process definitions in the stochastic pi-calculus are as follows.

$$A \triangleq !x.C \quad B \triangleq ?x.\mathbf{0} \quad C \triangleq ?z.\mathbf{0}$$

The use of the  $z$  channel, which does not appear elsewhere, ensures that the process  $C$  cannot take part in any subsequent pi-calculus reactions. Our initial stochastic pi-calculus process will be 100 copies of  $A$  and 100 copies of  $B$ . Now, our kappa rule is as follows.

$$X(a), Y(a) \rightarrow X(a^1), Y(a^1)$$

Our initial kappa solution will consist of 100 copies of  $Y(a)$ . Finally, we include the following glue reaction in our definition of the multi-calculus model.

$$C \longrightarrow X(a)$$

This glue reaction links the stochastic pi-calculus processes and kappa agents together to create a working multi-calculus model. Starting with 100 copies each of  $A$ ,  $B$  and  $Y(a)$ , the first interaction will be between  $A$  and  $B$ , which will lead to the creation of  $C$ . We know that  $C$  cannot take part in any stochastic pi-calculus interactions but it can trigger the glue reaction, producing the kappa agent  $X(a)$ . This then activates the kappa part of our multi-calculus compiler definition, since  $X(a)$  and  $Y(a)$  can bind to each other on the site  $a$ , replacing them with the agents  $X(a^1), Y(a^1)$ , which cannot take part in any reactions at all. This demonstrates how glue reactions allow communication between the separate components of a multi-calculus model.

### 6.3. Biological Example

The previous example used the generic abstract machine to integrate biological models written using different modelling paradigms, namely process-based and rule-based. Here we show how the generic abstract machine can also integrate domain-specific languages, such as DSD, with more general-purpose languages, such as pi-calculus. Although DSD was developed specifically to model computation performed by DNA strand displacement systems, such systems will ultimately operate within the context of living cells. For example, [1] proposed a system which could take as input RNA strands representing known cancer markers, and produce as output a DNA strand which was a known anti-cancer drug. Although this particular system required additional restriction enzymes to function, it is possible to envisage a strand-displacement variant. Furthermore, since RNA and DNA can form complexes and are structurally quite similar, we can use DSD to model both the RNA and DNA strands. An example system is  $\{a^* \} [A \ b^*] : [B \ c^*] \langle C \rangle$ , represented graphically as:



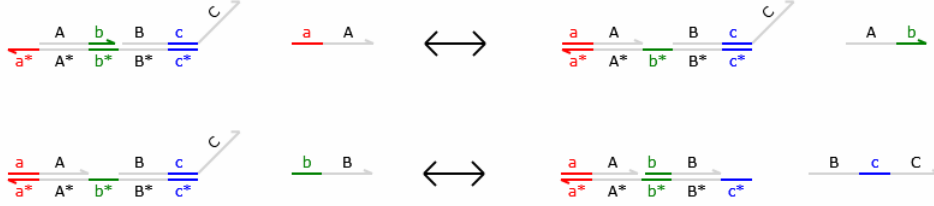
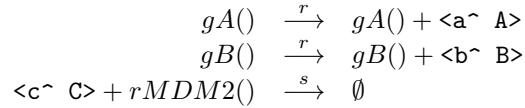


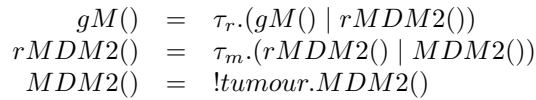
Figure 1: Execution of a strand displacement logic circuit, which produces an output  $\langle c^{\wedge} C \rangle$  if both inputs  $\langle a^{\wedge} A \rangle$  and  $\langle b^{\wedge} B \rangle$  are present. Each execution step represents a merged sequence of strand binding, branch migration and strand displacement reactions.

This DNA complex takes as input two RNA strands  $\langle a^{\wedge} A \rangle$  and  $\langle b^{\wedge} B \rangle$ , and produces an output DNA strand  $\langle c^{\wedge} C \rangle$  if both inputs are present. Thus, the complex acts as a logical AND gate. The gate is somewhat more complex than the one presented in 4.4, since it completely decouples the output strand from the inputs. An execution trace for the gate in the presence of both inputs is shown in Fig. 1. The input RNA strands could represent known cancer markers, while the output strand could represent a known cancer antidote, such as Vitravene, which functions by suppressing the production of the oncogene MDM2, as outlined in [1].

We model the production and consumption of the RNA strands using the following glue reactions, which allow species from both calculi to interact, where names beginning with  $g$  and  $r$  denote genes and messenger RNA in the pi-calculus model, respectively:



We then define a partial model of the host cell machinery in pi-calculus, focussing on the production of the MDM2 protein via transcription and translation mechanisms, together with the machinery for tumour formation.



In general, the DSD circuit will be significantly more complex than the one outlined above, as will the model of the host cell physiology. However, the basic motivation remains the same - we can use a suitable domain-specific language to model the computation involving DNA strand displacement systems, with all its advantages, and use a more general-purpose language to model the behaviour of the host cell.

#### 6.4. Recursive Models

Note that there is no reason why the sub-calculi mentioned here could not themselves be heterogeneous calculi whose species and reactions are computed recursively using a similar procedure. This means that we can define hierarchical models, where the sub-components could themselves contain sub-components

written using various different domain-specific languages, using our generic abstract machine as a common implementation layer. This could be a promising approach for modelling larger biological systems as these are often inherently hierarchical and composed of many different kinds of functional unit, while the fundamental mode of interaction is still via chemical reactions.

The ability to define complex, heterogeneous and recursive models in the same framework as simple single-calculus models demonstrates the expressive power of the generic abstract machine presented in this paper.

## 7. Discussion

In this paper we have presented a generic abstract machine for the simulation of process calculi with potentially unbounded numbers of species and reactions. We have instantiated the abstract machine with two Markovian simulation methods and four process calculi, namely the stochastic pi-calculus, the bioambient calculus, the kappa calculus and the DNA strand displacement calculus. We have demonstrated the correctness of Markovian abstract machine instantiations for these four calculi by means of a generic proof method. Finally, we have defined a general method for simulating multiple process calculi simultaneously.

This paper is a significantly revised and extended version of two conference papers, [16] and [22]. In [16] we defined a generic abstract machine together with its instantiation to the stochastic pi-calculus. We also summarised an instantiation to the bioambient calculus in an appendix. Here we provide full details on the instantiation to bioambients and we present significantly revised encodings for both calculi, which define the *reactions* functions directly in terms of the operational semantics of the calculus. This greatly simplifies the instantiations and also simplifies the corresponding proofs of correctness. We also instantiate the abstract machine to the kappa calculus and the DSD calculus, to demonstrate the broader applicability of our approach. In [16] we briefly outlined a proof of correctness of the generic abstract machine with respect to the stochastic pi-calculus with general distributions. In this paper we prove generic correctness theorems for an arbitrary process calculus with Markovian rates. These theorems are parameterised by the *species*, *process* and *reactions* functions for that calculus. We provide instances of this proof for the stochastic pi-calculus, the bioambient calculus, the kappa calculus and the DSD calculus. The idea of multi-calculus simulation within our generic framework was originally proposed in [22], whereas in this paper we provide full details of the approach.

For each of the four calculi presented in this paper, the calculus-specific *reactions* function is derived from the underlying reduction semantics of the calculus. This approach relies on the existence of an appropriate *process* function from (multisets of) species back to processes and on the fact that processes can be extracted from their context and still perform reductions. This suggests that it may be possible to automatically derive an instantiation of many other process calculi directly from their reduction semantics. It would be interesting to characterise the collection of process calculi for which this is possible.

In addition to defining the *species* and *reactions* functions, the user must also decide what constitutes a “species” for their calculus of interest. For example, in our instantiation to the stochastic pi-calculus we allow a species to be either an instance (which is expanded out to a choice by consulting the environment) or a

complex. This allows us to optimise the treatment of complexes in the simulator, which provides efficiency gains. Similar optimisations may be possible for our instantiation of the bioambient calculus, but these refinements are left for future work.

In future we might consider generalising the proof method to non-Markovian rates, to reconcile this work with our non-Markovian simulation algorithm [16]. This would require defining non-Markovian semantics for each calculus, and considering the correspondence between transitions of the calculus and the abstract machine.

The use of garbage collection techniques to remove obsolete species and their associated reactions from the abstract machine may be necessary in order to make tractable the simulation of systems generating a large number of species that are used only once. In Sec. 2 we briefly outlined how the abstract machine could be adapted to garbage collect species with zero populations. However, in cases where species are continually switching between zero and non-zero populations, more advanced garbage collection heuristics could be used. For example, a memory model could be introduced such that all species with zero population are garbage collected once the total number of species exceeds a given threshold. Alternatively, a species could be garbage collected if its population remains zero beyond a given number of simulation steps.

Custom simulation engines often rely on language-specific optimisations to improve simulation efficiency. In this paper we have demonstrated how optimisations for some of these languages can be implemented within the generic abstract machine. Examples include aggregation of complexes in the stochastic pi-calculus, and computation of propensities for connected components in the kappa calculus. Future work could investigate whether further optimisations, such as the one outlined in [6] for connected components in kappa, could be achieved by modifying the calculus instantiations in Sec. 4, or by generalising the abstract machine definitions in Sec. 2.

The current abstract machine relies on a finite set of glue reactions, which allows species from multiple calculi to interact with each other. An alternative approach would be to define a notion of aliasing, which identifies species from one calculus with species from another. Future work is needed to investigate alternative methods for implementing interoperability between calculi.

Our generic abstract machine aims to simulate a broad range of process calculi. This includes process calculi capable of  $n$ -ary interactions, and reactions with arbitrary stoichiometric coefficients. To highlight the flexibility of our approach, to date we have used the abstract machine to implement the DNA Strand Displacement (DSD) calculus for modelling DNA circuits [21], the Genetic Engineering of Cells (GEC) calculus for modelling of genetic devices [17], and the Stochastic Pi Machine (SPiM) calculus for general modelling of biological systems [29], by defining appropriate *species* and *reactions* functions for each calculus. Simulators for these three calculi are available online at <http://research.microsoft.com/dna>, <http://research.microsoft.com/gec> and <http://research.microsoft.com/spim>, respectively. A comparison with the previous SPiM implementation is outlined in the main text. Note that the DSD implementation is based directly on the generic abstract machine, and as such there is no prior implementation to compare with. Nevertheless, the generic abstract machine greatly simplified the implementation of this calculus by allowing significant code re-use. We have left the implementations of the kappa

and bioambient calculi for future work. In our experience, the approach outlined in this paper has greatly accelerated the development of these programming languages, by reducing the overhead for implementing custom stochastic simulation algorithms and allowing code re-use between projects. We are currently investigating implementations of other process calculi such as the brane calculus [3] and a calculus based on statecharts [13].

**Acknowledgements.** We thank Filippo Polo for his work developing the SPiM user interface and visualisations, and Kathy Gray together with the anonymous reviewers for valuable comments.

## References

- [1] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429(6990):423–429, May 2004.
- [2] D. Bratsun, D. Volfson, L. S. Tsimring, and J. Hasty. Delay-induced stochastic oscillations in gene regulation. *Proceedings of the National Academy of Sciences of the United States of America*, 102(41):14593–14598, 2005.
- [3] L. Cardelli. Brane calculi. In *Computational Methods in Systems Biology*, pages 257–278, 2004.
- [4] L. Cardelli. On process rate semantics. *Theor. Comput. Sci.*, 391(3):190–215, 2008.
- [5] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. *CONCUR 2007 - Concurrency Theory*, chapter Rule-Based Modelling of Cellular Signalling, pages 17–41. 2007.
- [6] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks, invited paper. In Z. Shao, editor, *Proceedings of the Fifth Asian Symposium on Programming Systems, APLAS '2007, Singapore*, volume 4807 of *Lecture Notes in Computer Science*, pages 139–157, Singapore, 2007. Springer, Berlin, Germany.
- [7] V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract interpretation of cellular signalling networks. In F. Logozzo, D. Peled, and L. Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 4905 of *Lecture Notes in Computer Science*, pages 83–97. Springer Berlin / Heidelberg, 2008.
- [8] L. Dematté, C. Priami, and A. Romanel. Modelling and simulation of biological processes in BlenX. *SIGMETRICS Performance Evaluation Review*, 35(4):32–39, 2008.
- [9] R. Ewald, J. Himmelspach, M. Jeschke, S. Leye, and A. M. Uhrmacher. Flexible experimentation in the modeling and simulation framework JAMES II – implications for computational systems biology. *Brief Bioinform*, Jan 2010.

- [10] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.
- [11] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.
- [12] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115:1716–1733, 2001.
- [13] D. Harel. Statecharts : A Visual Formalism for Complex Systems. *Sci. Comput. Prog.*, 8:231–274, 1987.
- [14] M. R. Lakin and A. Phillips. Modelling, simulating and verifying turing-powerful strand displacement systems. In L. Cardelli and W. M. Shih, editors, *DNA*, volume 6937 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2011.
- [15] M. R. Lakin, S. Youssef, L. Cardelli, and A. Phillips. Abstractions for DNA circuit design. *Journal of the Royal Society Interface*, July 2011. Published online, doi:10.1098/rsif.2011.0343.
- [16] L. Paulevé, S. Youssef, M. R. Lakin, and A. Phillips. A generic abstract machine for stochastic process calculi. In *CMSB '10: Proceedings of the 8th International Conference on Computational Methods in Systems Biology*, pages 43–54, New York, NY, USA, 2010. ACM.
- [17] M. Pedersen and A. Phillips. Towards programming languages for genetic engineering of living cells. *Journal of the Royal Society Interface*, 6(S4):437–450, August 2009.
- [18] M. Pedersen and G. Plotkin. A language for biochemical systems. In *Computational Methods in Systems Biology*, volume 5307 of *LNCS*, pages 63–82. Springer, 2008.
- [19] A. Phillips. An abstract machine for the stochastic bioambient calculus. *Electronic Notes in Theoretical Computer Science*, 227:143–159, January 2009.
- [20] A. Phillips and L. Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *Computational Methods in Systems Biology*, volume 4695 of *LNCS*, pages 184–199. Springer, September 2007.
- [21] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 6(S4):419–436, August 2009.
- [22] A. Phillips, M. Lakin, and L. Paulevé. Stochastic simulation of process calculi for biology. In G. Ciobanu and M. Koutny, editors, *MeCBIC*, volume 40 of *EPTCS*, pages 1–5, 2010.
- [23] C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80:25–31, 2001.

- [24] L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, Jun 2011.
- [25] L. Qian, E. Winfree, and J. Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, Jul 2011.
- [26] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Y. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.
- [27] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, Singapore, 2001. World Scientific Press.
- [28] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *J. Chem. Phys.*, 121:10356–10364, 2004.
- [29] D. Y. Wang, L. Cardelli, A. Phillips, N. Piterman, and J. Fisher. Computational modeling of the egfr network elucidates control mechanisms regulating signal dynamics. *BMC Systems Biology*, 3(118), December 2009.