Biochemical Reaction Rules with Constraints

Mathias John, Cédric Lhoussaine, Joachim Niehren, Cristian Versari

Biochemical Reaction Rules with Constraints

Mathias John[1,2], Cédric Lhoussaine[1,2],
Joachim Niehren[1,3], and Cristian Versari[4]

[1] BioComputing, Lifl (Cnrs Umr8022) & Iri (Cnrs Usr3078)
[2] University of Lille 1
[3] Inria, Lille
[4] University of Bologna

Abstract. We propose *React*(C), an expressive programming language for stochastic modeling and simulation in systems biology that is based on biochemical reactions with constraints. We prove that *React*(C) can express the stochastic π-calculus, in contrast to previous rule-based programming languages, and further illustrate the high expressiveness of *React*(C). We present a stochastic simulator for *React*(C) independently of the choice of the constraint language C. Our simulator decides for a given reaction rule whether it can be applied to the current biochemical solution. We show that this decision problem is NP-complete for arbitrary constraint systems C and that it can be solved in polynomial time for rules of bounded arity. In practice, we propose to solve this problem by constraint programming.

1 Introduction

The paradigm of chemical reactions is predominant in programming languages used for modeling and simulation in systems biology [6, 9, 3, 19, 1]. Chemical reactions are advantageous in that they can be given both, a continuous semantics in terms of ordinary differential equations (odes) as well as a stochastic semantics in terms of continuous time Markov chains (ctmcs). While odes describe deterministically the average dynamics of molecule populations, ctmcs describe the probabilities and speed of molecular interactions in an individual-based manner. The continuous semantics of a system of chemical reactions is an abstraction of its more precise stochastic semantics.

Biochemical reactions in the κ-calculus are widely accepted as a useful modeling language for systems biology [4, 12, 5, 6]. The underlying idea is to model biochemical reactions as graph rewrite rules. The following rewrite rule, for instance, states that a C-molecule with a free binding site 1 can be linked to an A-B complex by using the free binding site 1 of A (while the complex uses binding sites 2 of A and 1 of B).

$$A(1+2^y), B(1^y), C(1) \xrightarrow{4.5} (\nu x)A(1^x+2^y), B(1^y), C(1^x)$$

The stochastic rate 4.5 determines the distribution of the speed of this interactions according to the law of mass-action.

The alternative paradigm of agent-based modeling languages attracted also much interest for modeling and simulation in systems biology. It underlies the stochastic π-calculus and its many extensions, [18, 21, 14, 17, 23, 11], BioAmbients [20], BlenX or beta-binders [22], etc. The close relationship between modeling languages of both paradigms was first pointed out by Cardelli [2]. He identified the fragment of the stochastic π-calculus without ν-binders with systems of chemical reactions with the same CTMCs in order to obtain a continuous semantics of π-calculus processes in terms of ODEs. So far, however, there exists no positive result showing the expressiveness of the stochastic π-calculus for any language of chemical reactions. There exists a result for the π-calculus without stochastic semantics, which was shown equally expressive to the join calculus, a programming language based on chemical reaction rules, by Fournet et. al. [7]. Unfortunately, the encoding presented cannot be adapted to a stochastic setting in any obvious manner. Conversely, Danos and Laneve [6] showed that binary reaction rules of the κ-calculus without ν-binders on the left-hand side can be encoded in the π-calculus. The stochastic semantics is preserved as shown in [15]. A main limitation of the κ-calculus compared to the π-calculus is the restriction to graph rather than hypergraph rewriting.

In this paper we present *React*(C), a language of biochemical reaction rules with constraints. *React*(C) extends on the κ-calculus by hyperedge rewriting in particular. The graph rewrite rule from the κ-calculus above, for instance, can be written as follows, where `free` is a constant standing for a free binding site:

$$A(\texttt{free}; y), B(y), C(\texttt{free}) \xrightarrow{4.5} (\nu x)\ A(x; y), B(y), C(x)$$

Note that names of binding sites are identified by positions in *React*(C). The usefulness of hypergraph rewriting can be illustrated at modeling compartments, where the natural idea is to attribute each molecule by its compartment's name, i.e., to introduce a hyperedge per compartment that links all its molecules. One can then constrain reactions to happen only within a same compartment. For instance, consider a dimerization reaction between two A-molecules in the same compartment x:

$$A(x, \texttt{free}), A(x, \texttt{free}) \xrightarrow{3.1} (\nu z)A(x; z), A(x; z)$$

Variables with numeric values and arithmetic constraints are supported by *React*(C) too. These are useful for modeling dynamic volumes of compartments and to cope with alternative kinetics as those of the π-calculus. Simpler finite domain constraints were proposed for biological modeling in [13] and in BioCham. The following reaction rule for instance states that a polymerase bound at the *DNA* nucleotide with position x can advance to the next nucleotide at position $y = x + 1$ if x belongs to the finite domain $\{1, \ldots 47\} \setminus \{37\}$. The speed is given by the law of Mass action with a stochastic rate of 4.5.

$$\forall x \in \{1, \ldots 47\} \setminus \{37\}.\ Pol(x), \text{DNA}(y) \xrightarrow{\texttt{if } y{=}x{+}1 \texttt{ then } 4.5 \texttt{ else } 0} \text{DNA}(x), Pol(y)$$

Finally, *React*(C) enables general kinetics beside Mass action and Michaelis-Menten. Note that BioCham [3] and SBML [9] support both these kinetics but neither variables nor ν-binders.

Our main technical contribution is a proof that *React*(C) can indeed express the stochastic π-calculus, even if restricted to binary reaction rules with equality constraints and arithmetics on real numbers. This result is relevant since the π-calculus is the usual yardstick for the expressiveness of concurrent languages. Our result also illustrates that *React*(C) can express extensions of the stochastic π-calculus with constraints such as $\pi@$ [23] (without priorities) and the attributed π-calculus [11]. This means that all previous models in these languages carry over to *React*(C) in a systematic manner. Hyperedges and constraints thus provide the missing link between rule-based and agent-based modeling languages.

We present a stochastic simulation algorithm for *React*(C) that is independent of the choice of the constraint language C. Our simulator must decide for a given reaction rule whether it can be applied to the current biochemical solution. We show that this decision problem is NP-complete for arbitrary constraint systems C and that it can be solved in polynomial time for rules of bounded arity. In practice, we propose to solve this problem by constraint programming.

Our hardness proof relies on hypergraphs, so it does not apply to the κ-calculus. Indeed, the so called rigidity property of the κ-calculus (Lemma 3 of [5]) fails for *React*(C). Rigidity states that a matching of a connected pattern is entirely determined by matching only a single one of its molecules. It implies that the matching problem for the κ-calculus restricted to rules with connected patterns can be solved in P-time. The general case with multiple connected components remains open. We leave it also open whether the scalable simulation algorithm for the κ-calculus from [5] can be lifted to *React*(C) in any sense.

Outline. We start with a small language $\mathit{React}^{=}$ of biochemical reaction rules with equality constraints in Section 2 and show that it can express the stochastic π-calculus in Section 3 and 4. The main remaining problem not discussed so far, is to link stochastic mass-actions semantics with redex based stochastic semantics as in the stochastic π-calculus. The full language *React*(C) is presented in Section 5 and our simulation algorithm based on constraint programming in Section 6.

2 Reaction Rules with Equality Constraints

We present a small language of biochemical reaction rules with equality constraints $\mathit{React}^{=}$ that can express the stochastic π-calculus. We equip $\mathit{React}^{=}$ with a stochastic semantics that follows the usual law of Mass action.

We assume a signature $\mathcal{A}$ of molecule names $A \in \mathcal{A}$, each of which has a fixed arity in $ar(A) \in \mathbb{N}_0$. We also assume an infinite set $\mathcal{N}$ of (link) names ranged over by x, y and write $\tilde{x}$ for a possibly empty sequence of names $x_1; \ldots; x_n$. A molecule a is a term $A(\tilde{x})$ with $n = ar(A)$.

We define the biochemical solutions of $\mathit{React}^{=}$ in Fig. 2 as terms s that are constructed from molecules $A(\tilde{x})$, the composition operator s, s', and the empty solution $\mathbf{0}$. We often think of a biochemical solution as a (hyper-) graph of molecules that are linked by (hyper-) edges. For instance $A(x), B(x)$ describes the graph with two nodes $A(x)$ and $B(x)$ linked by a single edge named x. Such

Solutions	$s ::= A(\tilde{x}) \mid s, s' \mid \mathbf{0}$	where $A \in \mathcal{A},\ \tilde{x} \in \mathcal{N}$
Rate expressions	$e ::= \texttt{if } x_1{=}x_2 \texttt{ then } e_1 \texttt{ else } e_2$	where $x_1, x_2 \in \mathcal{N}$
	$\mid\ e_1 + e_2 \mid e_1 * e_2 \mid d$	and $d \in \mathbb{R}_0^+$
Reaction rules	$r ::= s \xrightarrow{e} (\nu\tilde{x})s'$	where $fn(r) = fn(s)$
Reactions	$\rho ::= s \xrightarrow{d} s'$	

Fig. 1. Syntax of reaction rules of language $React^{=}$

Precongruence	$(s_1, s_2), s_3 \approx s_1, (s_2, s_3) \qquad s_1, s_2 \approx s_2, s_1 \qquad s, \mathbf{0} \approx s$
Congruence	$\dfrac{s \approx s' \quad \sigma : \mathcal{N} \to \mathcal{N} \text{ injective}}{s \equiv s'\sigma}$

Fig. 2. Precongruence and congruence on solutions

graphs do neither depend on the order of molecules nor on the concrete choice of link names. For instance, the same graph is obtained by solutions $B(y), A(y)$ and $A(x), B(x)$.

In Fig. 2, we define two congruence relations on solutions. The precongruence $\approx$ captures order independence. It is the least equivalence relation on solutions that renders the composition operator "," associative and commutative with the neutral element $\mathbf{0}$. We write $[s]_\approx$ for the equivalence class of a solution s. Clearly, we can identify precongruence classes with multisets of molecules. The weaker congruence relation $\equiv$ accounts for the irrelevance of concrete names in addition. It is defined such that $s \equiv s'$ if and only if there exists a solution $s'' \approx s$ and an injective function $\sigma : \mathcal{N} \to \mathcal{N}$ such that $s' = s''\sigma$, i.e., the term obtained by renaming all names x in s'' to $\sigma(x)$.

We write $fn(s)$ for the set of names occurring in s. As usual, we define iterated compositions $\prod_{i=1}^{0} s_i = \mathbf{0}$, $\prod_{i=1}^{n} s_i = (\prod_{i=1}^{n-1} s_i), s_n$, and $s^m = \prod_{i=1}^{m} s$. If $a_1, \ldots, a_n$ are pairwise distinct molecules then we define $\prod_{i=1}^{n} a_i^{m_i} = \prod_{i=1}^{n} a_i^{m_i}$. Modulo precongruence, this term stands for the multiset with m_i occurrences of molecule a_i.

Reactions ρ are terms of the form $s_1 \xrightarrow{d} s_2$. They can be applied to rewrite solutions congruent to s, s_1 to some solution congruent to s, s_2:

$$(\text{REACT})\ \frac{s_1' \equiv s, s_1 \quad s, s_2 \equiv s_2'}{s_1 \xrightarrow{d} s_2 \vdash s_1' \to s_2'}$$

Judgements $\rho \vdash s_1' \to s_2'$ capture the non-deterministic semantics of reactions. Note that the rate constant d is irrelevant here; it matters in the stochastic semantics only, see below.

Reaction rules r are terms of the form $s \xrightarrow{e} (\nu\tilde{x})s'$. They are to be understood as schemas that define sets of reactions, one reaction per substitution $\sigma : \mathcal{N} \to$

$$(\text{COND}_1)\ \frac{e_1 \Downarrow d_1}{\texttt{if } x{=}x \texttt{ then } e_1 \texttt{ else } e_2 \Downarrow d_1} \qquad (\text{COND}_2)\ \frac{x_1 \neq x_2 \quad e_2 \Downarrow d_2}{\texttt{if } x_1{=}x_2 \texttt{ then } e_1 \texttt{ else } e_2 \Downarrow d_2}$$

$$(\text{REALS})\ \frac{d \in \mathbb{R}_0^+}{d \Downarrow d} \qquad (+)\ \frac{e_1 \Downarrow d_1 \quad e_2 \Downarrow d_2}{e_1 + e_2 \Downarrow d_1 +^{\mathbb{R}} d_2} \qquad (*)\ \frac{e_1 \Downarrow d_1 \quad e_2 \Downarrow d_2}{e_1 * e_2 \Downarrow d_1 *^{\mathbb{R}} d_2}$$

Fig. 3. Big-step evaluator of rate expressions.

$$(\text{INST})\ \frac{\sigma : \mathit{fn}(s) \to N \quad \sigma' : \{\tilde{x}\} \to \mathcal{N} \backslash (N \cup \mathit{fn}(s)) \text{ injective} \quad e\sigma \Downarrow d}{s \xrightarrow{e} (\nu\tilde{x})s' \Downarrow_{\sigma,N} s\sigma \xrightarrow{d} s'\sigma'\sigma}$$

Fig. 4. Instantiation and evaluation of reactions rules to reactions.

$\mathcal{N}$. Reaction rules contain a rate expressions e. Substitutions σ instantiating the rule are applied to e before evaluation, yielding another rate expression that we denote by $e\sigma$. Before formalizing the semantics of reaction rules, we need to define the values of rate expressions.

A rate expression e is a term built from constants $d \in \mathbb{R}_0^+$, addition $e{+}e$, multiplication $e * e$, and rate-valued equality constraints $\texttt{if } x_1{=}x_2 \texttt{ then } e_1 \texttt{ else } e_2$. We write $\mathit{fn}(e)$ for the set of names occurring in e and *Exprs* for the set of all rate expressions. Usual Boolean-valued constraints are subsumed, as for instance the conjunctive equality and inequality constraints $x{=}y \wedge y{\neq}z$ by rate expression $\texttt{if } x{=}y \texttt{ then } (\texttt{if } y{=}z \texttt{ then } 0 \texttt{ else } 1) \texttt{ else } 0$. In Fig. 3, we define an evaluator for rate expressions $\Downarrow$: $\mathit{Exprs} \to \mathbb{R}_0^+$ as usual. Note that this evaluator always terminates: neither there exist program errors nor non-termination.

A reaction rule $r = s \xrightarrow{e} (\nu\tilde{x})s'$ uses the new operator $(\nu\tilde{x})$ that binds the names in $\tilde{x}$ with scope s' similarly to the new operator of the π-calculus. It requires the creation of new names $\tilde{x}$ with scope s'. Being new means to not occur in the current solution to which the rule is applied. The free names of $(\nu\tilde{x})s'$ are thus defined by $\mathit{fn}((\nu\tilde{x})s') = \mathit{fn}(s') \setminus \{\tilde{x}\}$, and the free names of the reaction rule by $\mathit{fn}(r) = \mathit{fn}(s) \cup \mathit{fn}(e) \cup \mathit{fn}((\nu\tilde{x})s')$.

The instantiation (INST) of a reaction rule $r = s \xrightarrow{e} (\nu\tilde{x})s'$ to some reaction is defined in Fig. 4. There, we assume that $N \subseteq \mathcal{N}$ is a finite set of names – this will be the set of names of the current chemical reaction – and that $\sigma : \mathit{fn}(s) \to N$ is a substitution. Even though the domain of σ is restricted to $\mathit{fn}(s)$ we can still apply σ to r, since $\mathit{fn}(r) = \mathit{fn}(s)$ is assumed in the syntax of $\mathit{React}^{=}$. As a consequence, there exist only finitely many such substitutions and thus only finitely many rule instances to be considered (for some fixed new-name generator). The application of σ to r is defined as follows. First, some new-name generator $\sigma' : \{\tilde{x}\} \to \mathcal{N} \setminus (N \cup \mathit{fn}(s))$ introduces new names fresh for N and $\mathit{fn}(s)$ on r.h.s. of r, second, substitution σ is applied to the resulting rule, third, the expression $e\sigma$ is evaluated to some real number d. In this case, we say that r can be instantiated ρ by σ and N, where $\rho = s\sigma \xrightarrow{d} s'\sigma'\sigma$, and write $r \Downarrow_{\sigma,N} \rho$.

$$(\text{COUNT})\ \frac{s \approx \prod_{i=1}^{n} a_i^{m_i} \quad s' \approx \prod_{i=1}^{n+m} a_i^{m'_i}}{count(s;s') = \prod_{i=1}^{n} \binom{m'_i}{m_i}} \qquad (\text{REACT}_{\text{MA}})\ \frac{s'_1 \approx s, s_1 \quad s, s_2 \equiv s'_2}{s_1 \xrightarrow{d} s_2 \vdash s'_1 \xrightarrow{d * count(s_1;s'_1)} s'_2}$$

$$(\text{RULES}_{\text{MA}})\ \frac{s_1 \equiv s'_1 \quad d = \sum_{r \in R} \sum_{\{(d',\sigma)\ \mid\ r \Downarrow_{\sigma,fn(s'_1)} \rho,\ \rho \vdash s'_1 \xrightarrow{d'} s_2\}} d'}{R \vdash s_1 \xrightarrow[\text{MA}]{d} s_2}$$

Fig. 5. Stochastic mass-action semantics of $React^{=}$.

The non-deterministic semantics of a reaction rule can now be defined by reduction to the non-deterministic semantics of reactions:

$$(\text{RULE})\ \frac{r \Downarrow_{\sigma,fn(s)} \rho \quad \rho \vdash s \to s'}{r \vdash s \to s'}$$

The set of free names of the current solution s is passed over to the instantiator $r \Downarrow_{\sigma,fn(s)} \rho$, in order to ensure that new-bound names are instantiated by fresh names for the current solution. Recall that only finitely many substitutions $\sigma : fn(r) \to fn(s)$ are to be considered. These are the possible matchings of the left hand side of the rule with the current solution.

The stochastic semantics of $React(C)$ in Fig. 5 refines the non-deterministic semantics. The rate of a reaction rule now determines the probability and speed of its application according to the law of mass action. Inference rule (COUNT) defines $count(s;s')$ which is the number of occurrences of multiset $[s]_{\approx}$ in multiset $[s']_{\approx}$. Note that it has a unique value independently of the choice of s and s' in their congruence class, i.e. the ordering on the multiset elements imposed by s and s' does not affect the result. Furthermore, recall that $\binom{m}{m'} = 0$ if $m' > m$, so that $count(s;s') = 0$ if multiset $[s]_{\approx}$ is not contained in $[s']_{\approx}$. Inference rule (REACT_{MA}) states how to apply a reaction rule $s_1 \xrightarrow{d} s_2$ to a solution s'_1. This works as in the non-deterministic case (REACT) except that an application rate $d * count(s_1;s'_1)$ is computed. Inference rule (RULES_{MA}) describes applications of systems of chemical reactions R to a given solution s while producing s'. The situation is analoguous to the non-deterministic semantics in (RULE) except that we now have to sum up the application rates d' of all instantiations σ of reaction rules in R to reactions that can reduce s to s'.

The stochastic semantics of $React(C)$ defines a CTMC for each set of chemical reactions. The states of this Markov chain are equivalence classes $[s]_{\equiv}$ of solutions s modulo full congruence. Its state transitions are obtained by applying reactions rules according to rule (RULES_{MA}). That is, reaction rates of all instantiations of rules in R that lead to the same state $[s]_{\equiv}$ are summed up providing the rate of a single transition. Note that the precongruence must be used while counting (since different renamings should not be counted). Consider, e.g., solution $s = A(x), A(y)$ and rule $r = A(x_1), A(x_2) \xrightarrow{2.1} A(x_1)$. For r we obtain two possible instantiations $\sigma = \{(x_1,x),(x_2,y)\}$ and $\sigma' = \{(x_1,y),(x_2,x)\}$ both

$$(\text{RED})\ \frac{s_1 = \prod_{i=1}^{n} a_i \quad s_2 = \prod_{i=1}^{n'} a'_i}{\begin{array}{l} \mathit{redex}(s_1; s_2) = \{\ell : \{1,\ldots,n\} \to \{1,\ldots,n'\} \text{ injective } \mid \\ \qquad\qquad a'_{\ell(i)} = a_i \text{ for all } i \in \{1,\ldots,n\}\} \end{array}}$$

$$(\text{REACT}_{\text{RED}})\ \frac{s'_1 \approx s, s_1 \quad s, s_2 \equiv s'_2 \quad d \in \mathbb{R}^+ \quad \ell \in \mathit{redex}(s_1; s'_1)}{s_1 \xrightarrow{d} s_2 \vdash s'_1 \xrightarrow[\ell]{d} s'_2}$$

$$(\text{RULES}_{\text{RED}})\ \frac{s'_1 \equiv s_1 \quad d = \sum_{r \in R} \sum_{\{(d',\ell) \mid r \Downarrow_{\sigma,fn(s'_1)} \rho,\ \rho \vdash s'_1 \xrightarrow[\ell]{d'} s_2\}} d'}{R \vdash s_1 \xrightarrow[\text{RED}]{d} s_2}$$

Fig. 6. Stochastic redex semantics of language $\mathit{React}^{=}_{\text{RED}}$.

leading to the same state $[s']_{\equiv} = [s'']_{\equiv}$ where $s' = A(x)$ and $s'' = A(y)$. Thus, we obtain one transition $[s]_{\equiv} \xrightarrow{4.2} [s']_{\equiv}$.

3 An Alternative Stochastic Semantics

We provide an alternative stochastic semantics for systems of reaction rules in the small language that is based on redexes in analogy to the usual semantics of the stochastic π-calculus. We call the small language with the redex semantics $\mathit{React}^{=}_{\text{RED}}$ and show that $\mathit{React}^{=}_{\text{RED}}$ can be encoded into $\mathit{React}^{=}$ while preserving CTMCs. This encoding will provide the first part of our encoding of the stochastic π-calculus into $\mathit{React}^{=}$.

The particularity of a redex semantics is that it treats solutions as lists in some fixed order. It then enumerates all redexes by which a smaller list can be mapped into a larger list. A redex of a solution $s_1 = \prod_{i=1}^{n} a_i$ in a solution $s_2 = \prod_{i=1}^{n'} a'_i$ is an injective function $\ell : \{1,\ldots,n\} \to \{1,\ldots,n'\}$, such that $a'_{\ell(i)} = a_i$ for all $i \in \{1,\ldots,n\}$. Note that ℓ depends on the order of molecules in s_1 and s_2. For instance, solution A, A has two redexes in itself, $\ell_1 = \{(1,1),(2,2)\}$ and $\ell_2 = \{(1,2),(2,1)\}$, even though $\mathit{count}(A, A; A, A) = \binom{2}{2} = 1$. The reason is that the notion of redexes is order sensitive in contrast to the notion of multiset inclusion on which function *count* used in the Mass action semantics is based.

The stochastic redex semantics rules are given in Fig. 6. The language of reaction rules with this semantics is called $\mathit{React}^{=}_{\text{RED}}$. Definition (RED) introduces the set $\mathit{redex}(s_1; s_2)$ of all redexes by which s_1 matches s_2. Inference rule ($\text{REACT}_{\text{RED}}$) applies a reaction at a redex to a solution. Rule ($\text{RULES}_{\text{RED}}$) treats the application of all instances of reaction rules to a solution. Notice that in contrast to rule (RULES_{MA}) of $\mathit{React}^{=}$, rule ($\text{RULES}_{\text{RED}}$) does not consider substitutions to identify rule instances, since redexes can be used for this purpose equally well.

Lemma 1. *For all r, ℓ, and s there exists at most one σ, such that $r \Downarrow_{\sigma,\text{fn}(s)} \rho$ and $\rho \vdash s \xrightarrow[\ell]{d} s'$.*

$$
\begin{aligned}
&[\![s \xrightarrow{e} (\nu\tilde{x})s']\!] \;=_{\text{def}}\; s \xrightarrow{[\![e]\!]_s} (\nu\tilde{x})s' \\
&[\![e]\!]_{\prod_{i=1}^n A_i(\tilde{x}_i)} \;=_{\text{def}}\; e * \textstyle\prod_{i=1}^n \sum_{j=i}^n \mathtt{eq}(A_i(\tilde{x}_i); A_j(\tilde{x}_j)), \text{ with} \\
&\qquad \mathtt{eq}(A_1(\tilde{x}_1); A_2(\tilde{x}_2)) =_{\text{def}} \begin{cases} 0 & \text{if } A_1 \neq A_2 \\ \mathtt{eq}'(\tilde{x}_1; \tilde{x}_2) & \text{if } A_1 = A_2 \end{cases} \\
&\qquad \mathtt{eq}'((x_1; \tilde{x}_1); (x_2; \tilde{x}_2)) =_{\text{def}} \mathtt{if}\ x_1{=}x_1'\ \mathtt{then}\ \mathtt{eq}'(\tilde{x}_1; \tilde{x}_2)\ \mathtt{else}\ 0 \\
&\qquad \mathtt{eq}'((); ()) =_{\text{def}} 1
\end{aligned}
$$

Fig. 7. Encoding from $React^{=}_{\text{RED}}$ to $React^{=}$.

Proof. Let r be the rule $s_1 \xrightarrow{e} (\nu\tilde{x})s_2$. Since $r \Downarrow_{\sigma, fn(s)} \rho$, rule (INST) provides that $\sigma : fn(s_1) \to fn(s)$. By the definition of rule ($\text{REACT}_{\text{RED}}$), for all names $x \in fn(s_1)$ it holds that the values $\sigma(x)$ are uniquely determined by ℓ and S. □

In order to find an encoding from $React^{=}_{\text{RED}}$ to $React^{=}$, we need to quantify the discrepancy between $count(s_1; s_2)$ and the cardinality $\#redex(s_1, s_2)$. Since redexes are order sensitive, this is given by the difference between a combination without repetition (Mass action) and a variation without repetition (redexes). That is for each molecule a_i in a solution $\prod_{i=1}^n a_i$ the number of positions $i' > i$ need to be counted, where $a_i = a_{i'}$. In a solution $s \approx \prod_{i=1}^n a_i^{m_i}$ this number is given by $m_i!$ for molecule a_i.

Lemma 2. *For all solutions* $s \approx \prod_{i=1}^n a_i^{m_i}$, $s' \approx \prod_{i=1}^{n+m} a_i^{m_i'}$ *such that* $m_i \leq m_i'$ *for all* $i \in \{1, \ldots, n\}$*:*

$$\#redex(s; s') = \prod_{i=1}^{n} \binom{m_i'}{m_i} * m_i! = count(s; s') * \prod_{i=1}^{n} m_i!$$

Based on a claim that the number of redexes does not depend on the concrete order fixed by a solution, the proof is straightforward by induction on n.

We present the encoding $[\![\cdot]\!] : React^{=}_{\text{RED}} \to React^{=}$ in Fig. 7. The basic idea is to balance the difference between Mass action and redex quantification by counting the number of permutations of molecule places in solution lists according to the ideas above. Our encoding from $React^{=}_{\text{RED}}$ to $React^{=}$ is correct, in that it preserves CTMCs.

Proposition 1. *The encoding* $[\![\cdot]\!] : React^{=}_{\text{RED}} \to React^{=}$ *preserves the* CTMC*, in that for all rule sets* $R \in React^{=}_{\text{RED}}$ *and solutions* s *it holds* $R \vdash s \xrightarrow[\text{RED}]{d} s'$ *if and only if* $[\![R]\!] \vdash s \xrightarrow[\text{MA}]{d} s'$*.*

The proof basically proceeds by structural induction on the rules of the stochastic semantics of $React^{=}_{\text{RED}}$ and $React^{=}$. It is based on Lemma 1 and on another lemma that states that for all reaction rules $s_1 \xrightarrow{e} s_2$ of $React^{=}_{\text{RED}}$ and substitutions σ, it holds that if $s_1\sigma \approx \prod_{i=1}^m a_i^{m_i}$ and $e\sigma \Downarrow d \in \mathbb{R}^+$ then $[\![e]\!]_{s_1} \Downarrow d * \prod_{i=1}^m m_i!$, so that Lemma 2 can be applied.

Prefixes	$\pi ::= x?\tilde{y}$	receiver	where $x, \tilde{y}, \tilde{z} \in \mathcal{N}$
	$\mid\ x{:}d!\tilde{z}$	sender	and $d \in \mathbb{R}_0^+$
Sums	$M ::= \pi.P$	prefixed process	
	$\mid\ M_1 + M_2$	choice	
Processes	$P, Q, O ::= A(\tilde{x})$	defined process	where $A \in \mathcal{A}$
	$\mid\ P_1 \mid P_2$	parallel composition	
	$\mid\ (\nu x)P$	channel creation	
	$\mid\ \mathbf{0}$	idle process	
Definitions	$D ::= A(\tilde{x}) \triangleq M$	process definition	where $fn(M) \subseteq \{x\}$

Fig. 8. Syntax of the π-calculus.

4 Expressing the Stochastic π-Calculus

In this section, we propose an encoding of the stochastic π-calculus into $React^{=}_{\text{RED}}$ preserving the underlying CTMC, according to $React^{=}_{\text{RED}}$ alternative semantics of Sect. 3. The definition of the π-calculus we propose here corresponds to its "biochemical" variant: the bodies of parametric process definitions are sums of prefixed processes, possibly restricted. In order to simplify the presentation of the encoding, but not at the expense of the expressiveness, the syntax given in Fig. 8 excludes ν-operators over sums. Free names and structural congruence are defined as usual for π-calculus reduction semantics.

The stochastic semantics of the π-calculus as given in Fig. 9 refines the usual non-deterministic semantics. It is based on standard indexing of processes and prefixes, which allows the enumeration of all the pairs of prefixes that give rise to some reduction, as well as on the presence of *normal forms*, that allow the very compact expression of a process as a parallel composition of defined processes possibly preceded by restrictions.

In order to formalize the encoding from the π-calculus to reaction rules, we define a standard correspondence between (sets of) π-calculus process definitions and (sets of) rules. For the sake of readability, our encoding relies on the following assumptions:

- The set of names $\mathcal{N}$ of the π-calculus is that of $React^{=}_{\text{RED}}$ and all free names on the right hand side of a definition $A(\tilde{x}) \triangleq P$ are bound on the left, i.e. $fn(P) \subseteq \{\tilde{x}\}$.
- The set of molecule names $\mathcal{A}$ of the π-calculus is that of $React^{=}_{\text{RED}}$ and the number of names $|\tilde{x}|$ in definitions $A(\tilde{x}) \triangleq P$ must be equal to the *arity* of A fixed by $\mathcal{A}$. Furthermore all formal parameters in $\tilde{x}$ must be pairwise distinct and there exists no multiple definitions, that is for any pair of definitions $A_i(\tilde{x}_i) \triangleq P_i, i \in 1, 2$, it holds that $A_1 \neq A_2$.

$$(\text{COM}_{\text{S}\pi})\ \frac{P_1 = \sum_h \pi_h^1.P_h^1 \quad P_2 = \sum_h \pi_h^2.P_h^2 \quad \pi_l^1 = x{:}d!\tilde{z} \quad \pi_m^2 = x?\tilde{y} \quad |\tilde{y}| = |\tilde{z}|}{P_1 \mid P_2 \xrightarrow[(l,m)]{d} P_i^1 \mid P_j^2[\tilde{z}/\tilde{y}]}$$

$$(\text{DEF}_{\text{S}\pi})\ \frac{A_1(\tilde{x}_1) \triangleq M_1 \quad A_2(\tilde{x}_2) \triangleq M_2 \quad M_1[\tilde{y}_1/\tilde{x}_1] \mid M_2[\tilde{y}_2/\tilde{x}_2] \xrightarrow[(l,m)]{d} Q}{A_1(\tilde{y}_1) \mid A_2(\tilde{y}_2) \xrightarrow[(l,m)]{d} Q}$$

$$(\text{REDUCT}_{\text{S}\pi})\ \frac{\begin{array}{l} P = (\nu\tilde{y}) \prod_{h=1}^{n} A_h(\tilde{x}_h) \\ A_j(\tilde{x}_j) \mid A_k(\tilde{x}_k) \xrightarrow[(l,m)]{d} Q \end{array} \quad O \equiv (\nu\tilde{y})\big(Q \mid \prod_{h \in \{1,\ldots,n\} \setminus \{j,k\}} A_h(\tilde{x}_h)\big)}{P \xrightarrow[(j,l,k,m)]{d} O}$$

$$(\text{SUM}_{\text{S}\pi})\ \frac{P \equiv P' \quad d = \sum_{\{(d',(j,l,k,m)) \mid P' \xrightarrow[(j,l,k,m)]{d'} O\}} d'}{P \xrightarrow{r} O}$$

Fig. 9. Stochastic semantics for the π-calculus.

Definition 1 (Normal form). *Processes $P = (\nu\tilde{x})\Pi_i A_i(\tilde{y}_i)$ are said to be in normal form. The subset of $\mathcal{P}$ of processes in normal form is denoted as $\widehat{\mathcal{P}}$. Thus, in the following, $\hat{P}$ denotes a process in normal form.*

Modulo the usual structural congruence rules any process can be put into normal form:

Lemma 3 (Congruent normal form). *For every π-calculus process P, there exists $\widehat{P} \equiv P$ such that $\widehat{P}$ is in normal form.*

Each process may be put in several different normal forms. In order to define the encoding from processes to solutions, we need to choose a unique normal form $\phi(P)$ for each P. Of course, the associative and commutative properties of structural congruence, as well as α-renaming, allow several distinct normalization functions $\phi(\cdot)$ to be defined. For our purpose, the specific choice is not relevant, as long as the same $\phi(\cdot)$ is always selected hereinafter.

Lemma 4 (Normalization function). *There exists (at least) one surjective and total function $\phi : \mathcal{P} \to \widehat{\mathcal{P}}$ such that $\forall P \in \mathcal{P} : \phi(P) \equiv P$ and $\forall P \in \widehat{\mathcal{P}} : \phi(P) = P$.*

Normal forms are useful to define the stochastic semantics of the π-calculus, given in Fig. 9. This semantics relies on redexes which are here tuples that locate a pair of complementary prefixes. In the reduction

$$(\nu\tilde{x})(A_1(\tilde{x}_1), \ldots, A_n(\tilde{x}_n)) \xrightarrow[(j,l,k,m)]{d} Q$$

an interaction with rate d involves the l^{th} (output) prefix of process A_j and the m^{th} (input) prefix of process A_k. From those located interactions, rule ($\text{SUM}_{\text{S}\pi}$)

sums up the rates of the reductions leading to a common state, so that the specific pairs of complementary prefixes are forgotten.

Similarly to the stochastic semantics of the π-calculus, the encoding of the π-calculus in $React^{=}_{\text{RED}}$ relies on the correspondence between the redexes of these two languages. Such encoding consists in two parts: the first one allows the translation of parametric process definitions to reaction rules, the second one defines a tight correspondence between π-calculus processes and $React^{=}_{\text{RED}}$ solutions.

The translation of parametric process definitions occurs in two steps: first, a rule is generated for each redex that locates a pair of complementary prefixes, which do not necessarily share the same subject name; then, the rates of identical rules are summed up. In order to illustrate these informal ideas, let us consider the following process definitions:

$$A(x,\dot{x}) \triangleq x{:}d!\dot{x}.\mathbf{0} + x{:}d!\dot{x}.\mathbf{0} + \dot{x}?z.B(x,\dot{x},z)$$
$$B(y,\dot{y},\ddot{y}) \triangleq y{:}d'!\ddot{y}.(\nu z)A(y,z) + \dot{y}{:}d''!\ddot{y}.B(y,\dot{y},\ddot{y})$$

Depending on how those definitions are instantiated, at most 4 interactions can occur. Those are given by redexes (j,l,k,m) identifying the l^{th} output prefix of the j^{th} definition and the m^{th} input prefix of the k^{th} definition [5]. For the above definition, those redexes are $(1,1,1,3)$, $(1,2,1,3)$, $(2,1,1,3)$ and $(2,2,1,3)$. A rule, constrained by the equality of the subject of the prefixes, corresponds to each redex:

$$(1,1,1,3) : A(x,\dot{x}), A(y,\dot{y}) \xrightarrow{\texttt{if } x=\dot{y} \texttt{ then } d \texttt{ else } 0} B(y,\dot{y},\dot{x})$$
$$(1,2,1,3) : A(x,\dot{x}), A(y,\dot{y}) \xrightarrow{\texttt{if } x=\dot{y} \texttt{ then } d \texttt{ else } 0} B(y,\dot{y},\dot{x})$$
$$(2,1,1,3) : B(y,\dot{y},\ddot{y}), A(x,\dot{x}) \xrightarrow{\texttt{if } y=\dot{x} \texttt{ then } d' \texttt{ else } 0} (\nu z)A(y,z), B(x,\dot{x},\ddot{y})$$
$$(2,2,1,3) : B(y,\dot{y},\ddot{y}), A(x,\dot{x}) \xrightarrow{\texttt{if } \dot{y}=\dot{x} \texttt{ then } d'' \texttt{ else } 0} B(y,\dot{y},\ddot{y}), B(x,\dot{x},\ddot{y})$$

We get rid of the redex indexing of rules by summing up the rates of identical rules, and thus we obtain the following rule-based model that corresponds to the above π-calculus definitions:

$$\left\{\begin{array}{l} A(x,\dot{x}), A(y,\dot{y}) \xrightarrow{2*\texttt{if } x=\dot{y} \texttt{ then } d \texttt{ else } 0} B(y,\dot{y},\dot{x}), \\ B(y,\dot{y},\ddot{y}), A(x,\dot{x}) \xrightarrow{\texttt{if } y=\dot{x} \texttt{ then } d' \texttt{ else } 0} (\nu z)A(y,z), B(x,\dot{x},\ddot{y}), \\ B(y,\dot{y},\ddot{y}), A(x,\dot{x}) \xrightarrow{\texttt{if } \dot{y}=\dot{x} \texttt{ then } d'' \texttt{ else } 0} B(y,\dot{y},\ddot{y}), B(x,\dot{x},\ddot{y}) \end{array}\right\}$$

We can now formalize the encoding. First we define the translation of π-calculus process definitions in reaction rules, then we formalize how to translate a process to a solution (and back to a process again).

Definition 2 (From process definitions to reaction rules). *Let $\mathcal{D}$ be a finite set of process definitions of π-calculus, $\mathcal{D} = \{\delta_s | \delta_s = A_s(\tilde{x}_s) \triangleq \sum_{t=1}^{n_s} \pi_s^t.P_s^t\}$ with cardinality $|\mathcal{D}|$. The tuple $R'_{\mathcal{D}}$ of reaction rules corresponding to $\mathcal{D}$ is defined as $R'_{\mathcal{D}} = \{(i, r_i) | r_i = s_i \xrightarrow{e_i} (\nu\tilde{x}_i)s'_i\}$, where i is a composite index $i = (j,l,k,m)$ for all j,l,k,m such that:*

[5] note that here we refer to j^{th} and k^{th} definitions while in the stochastic semantics the same redex refers to the j^{th} and k^{th} processes of the current state.

– *the* j^{th} *and* k^{th} *definitions in* $\mathcal{D}$ *are* $\delta_j = A_j(\tilde{x}_j) \triangleq \sum_{t=1}^{n_j} \pi_j^t.P_j^t$ *and* $\delta_k = A_k(\tilde{x}_k) \triangleq \sum_{t=1}^{n_k} \pi_k^t.P_k^t$ *if* $k \neq j$ *and* $\delta_k = A_j(\tilde{x}'_j) \triangleq \sum_{t=1}^{n_j} \pi_j^t.P_j^t[\tilde{x}'_j/\tilde{x}_j]$ *otherwise, for some fresh names* $\tilde{x}'_j$*;*
– $1 \leq l \leq n_j$ *and* $1 \leq m \leq n_k$
– $\pi_j^l = y_1{:}d!\tilde{z}_o \quad$ *and* $\quad \pi_k^m = y_2?\tilde{z}_i \quad$ *with* $|\tilde{z}_o| = |\tilde{z}_i|$*;*
– $s_i = A_j(\tilde{x}_j), A_k(\tilde{x}_k)$ *and* $s'_i = \phi(\ P_j^l \ \mid \ P_k^m[\tilde{z}_o/\tilde{z}_i]\ )$
– $e_i =$ `if` $y_1{=}y_2$ `then` d `else` 0.

Given a rule r*, the multiplicity* $\mathrm{M}(r)$ *of* r *in* $R'_{\mathcal{D}}$ *is defined as* $\mathrm{M}(r) = |\ \{i \mid (i,r) \in R'_{\mathcal{D}}\}\ |$*. The set* $R_{\mathcal{D}}$ *of reaction rules corresponding to* $\mathcal{D}$ *is defined as*

$$R_{\mathcal{D}} = \{s \xrightarrow{\mathrm{M}(r)*e} (\nu\tilde{x})s' \mid (i,r) \in R'_{\mathcal{D}} \text{ for some } i \text{ and } \ r = s \xrightarrow{e} (\nu\tilde{x})s'\}$$

In practice, the translation of a process in a solution removes the restrictions in front of the process and preserves the names of defined processes and of channels.

Definition 3 (From processes to solutions). *Let* P *be a* π*-calculus process, with* $\widehat{P} = (\nu\tilde{z})(A_1(\tilde{x}_1) \mid \cdots \mid A_n(\tilde{x}_n))$*. The solution* s_P *corresponding to* P *is defined as* $s_P = A_1(\tilde{x}_1), \ldots, A_n(\tilde{x}_n)$*.*

The reverse translation adds again restrictions in front of the process, by preserving all the names.

Definition 4 (From solutions to processes). *Let* s *be a solution, with* $s = A_1(\tilde{x}_1) \mid \cdots \mid A_n(\tilde{x}_n)$*. The process* P_s *corresponding to* s *is defined as* $P_s = (\nu\tilde{x}_1)\ldots(\nu\tilde{x}_n)P_s^{-(\nu)}$*, with* $P_s^{-(\nu)} = A_1(\tilde{x}_1), \ldots, A_n(\tilde{x}_n)$*.*

The remarkable expressiveness of $React^{=}_{\mathrm{RED}}$ allows it to provide a tight correspondence with the π-calculus: in fact, the state space generated by a π-calculus process P without free names is isomorphic to the one generated by its corresponding solution s_P. Moreover, the transition rate between any pair of states is preserved by the encoding, so that the CTMC associated with any π-calculus process P without free names is isomorphic to the one associated with its corresponding solution s_P. This important property is captured by the following theorem.

Theorem 1. *Let* $\mathcal{D}$ *be a finite set of process definitions of* π*-calculus,* $R_{\mathcal{D}}$ *be the set of reaction rules corresponding to* $\mathcal{D}$ *according to Def. 2. Let* P *be a* π*-calculus process with* $\mathrm{fn}(P) = \emptyset$*. Then:*

1. $P \xrightarrow{d} P' \quad \Rightarrow \quad R_{\mathcal{D}} \vdash s_P \xrightarrow[\mathrm{RED}]{d} s_{P'}$*;*
2. $R_{\mathcal{D}} \vdash s_P \xrightarrow[\mathrm{RED}]{d} s' \quad \Rightarrow \quad P \xrightarrow{d} P_{s'}$*.*

Surely, it is possible to relax the requirement of absence of free names for π-calculus processes at the price of losing isomorphism, since some process transitions are lost. Still it would be easy to identify again the isomorphic subchain of CTMC of the corresponding solution of $React^{=}_{\mathrm{RED}}$.

5 Biochemical Reaction Rules with General Constraints

In this section, we define a powerful language of biochemical reaction rules, $\mathit{React}(C)$, which besides others permits constraints in an arbitrary constraint system C, ν-binders on the left hand side, reflexivity, and general kinetics.

We define constraint languages like in higher-order logic in the simply typed call-by-value λ-calculus, extended by pairs, letrec expressions, case statements for matching molecules or solutions, and constants. We parametrize our λ-calculus by choice of base types, molecule constructors, and constants with a fixed semantics. Therefore, parameter C of $\mathit{React}(C)$ is assumed to be a tuple $C = (\mathcal{B}, \mathcal{A}, \mathcal{C}, [[.]])$ with the following properties:

- $\mathcal{B} = \{\iota, \ldots\}$ is a set of type constants such as $\mathtt{nat}_0$ (non zero natural numbers) and `real` for real numbers. Simple types build $\mathcal{B}$ and 3 further constants are defined in Fig. 10. They are ranged over by τ.
- $\mathcal{A} = \{A : \tilde{\tau}, \ldots\}$ is a set of typed molecule names, $\tilde{\tau}$ is a tuple of types.
- $\mathcal{C} = \{c : \tau, \ldots\}$ is a set of typed constants. If $\tau = \tau_1 \to \ldots \to \tau_n \to \tau'$ for some nonfunctional type τ' then we say that the arity of c is $ar(c) = n$. This set may contain constants for arithmetic functions such as $+ : \mathtt{nat}_0 \to \mathtt{nat}_0 \to \mathtt{nat}_0$.
- for every constant $c : \tau_1 \to \ldots \to \tau_n \to \tau''$ of arity n, there is a function $[[c : \tau]]_s : \mathit{Vals}(\tau_1) \times \ldots \times \mathit{Vals}(\tau_n) \to \mathit{Vals}(\tau'')$. Here, $\mathit{Vals}(\tau)$ is the set of values of type τ which are closed in that the only remaining variables are to type `link`, which is defined as usual for the simply typed λ-calculus (see Figs. 10 and 11).

Note that simple types τ include, beside type constants in $\mathcal{B}$ and function types, two forms of molecule types, $A(\tilde{\tau})$ for molecules of species A with parameters of type $\tilde{\tau}$ and a constant `mol` which is the type of molecules. Furthermore, there is a type constant `sol` for solutions and a type `link` for link names.

Expressions as defined in Fig. 10 consist of λ-calculus terms extended with constants, molecule and solution data terms, and their respective matching constructs. Rules for their evaluation are provided in Fig. 12. A solution expression is a list of expressions $e_1, \ldots, e_n$ that the type system forces to evaluate to molecules. The special term `current_sol` evaluates to the current solution (that is the current state). In the matching `case_mol` e `of` $A(\tilde{x})$ `then` e_1 `else` e_2 variables $\tilde{x}$ scope over e_1. If e evaluates to a molecule $A(\tilde{v})$ then e_1 is evaluated with variables $\tilde{x}$ binding values $\tilde{v}$. Otherwise, e_2 is evaluated. Similarly, in the matching `case_sol` e `of` x^y, z `then` e_1 `else` e_2, variables x, y and z scope over e_1. If e evaluates to $A(\tilde{v}), s$ then e_1 is evaluated where x binds to $A(\tilde{v})$, y binds to the multiplicity of $A(\tilde{v})$ in solution $A(\tilde{v}), s$, and z binds to s with all occurrences of $A(\tilde{v})$ removed from s.Otherwise, that is when e evaluates to the empty solution, e_2 is evaluated. Values of this constraint language are standard.

Language $\mathit{React}(C)$ has full support for reflexivity, meaning that the current solution can always be reflected into a value of the language. This is a powerful feature, since it permits to express global constraints on the current solution,

Types	$\tau ::= \iota \mid \tau \to \tau \mid A(\tilde{\tau}) \mid \texttt{sol} \mid \texttt{mol} \mid \texttt{link}$	where $\iota \in \mathcal{B},\ A \in \mathcal{A}$
Expressions	$e \in \mathit{Exprs} ::= c \mid x \mid \lambda x.e \mid ee$	where $c \in \mathcal{C},\ x, y, z \in \mathcal{N}$
	$\mid\ \texttt{letrec}\ x = e\ \texttt{in}\ e \mid A(\tilde{e})$	
	$\mid\ \texttt{case\_mol}\ e\ \texttt{of}\ A(\tilde{x})\ \texttt{then}\ e\ \texttt{else}\ e \mid \mathbf{0} \mid e, e$	
	$\mid\ \texttt{current\_sol} \mid \texttt{case\_sol}\ e\ \texttt{of}\ x^y, z\ \texttt{then}\ e\ \texttt{else}\ e$	
Solution patterns	$p \in \mathit{Pats} ::= A(\tilde{x}) \mid p, p \mid \mathbf{0}$	
Solutions	$s \in \mathit{Sols} ::= A(\tilde{v}) \mid s, s \mid \mathbf{0}$	
Values	$v \in \mathit{Vals} ::= x \mid \mathbf{0} \mid s \mid \lambda x.e \mid c\ v_1 \dots v_k$	where $0 \leq k < ar(c)$ or $k = 0$
Reaction rules	$r ::= (\nu\tilde{x})p \xrightarrow{e} (\nu\tilde{y})s$	where $\mathit{fn}(r) \subseteq \mathit{fn}((\nu\tilde{x})p)$ and $\{\tilde{y}\} \cap (\mathit{fn}(r) \cup \mathit{fn}(e) \cup \{\tilde{x}\}) = \emptyset$

Fig. 10. Expressions and values of $\mathit{React}(C)$.

which do not only depend on the subsolution matching the left-hand-side of a rule.

In particular, we can rely on reflexivity in order to support arbitrary kinetics, as we illustrate by the following sequence of examples. There, we assume that C supports a constant $=: \texttt{mol} \to \texttt{mol} \to \texttt{nat}_0$ for the equality function on molecules and $+ : \texttt{nat}_0 \to \texttt{nat}_0 \to \texttt{nat}_0$ for the addition over natural numbers. We start with a function that counts all molecules of a solution.

$$\mathit{count\_mols} \triangleq \lambda s.\ \texttt{letrec}\ f = (\texttt{case\_sol}\ s\ \texttt{of}\ x^y, z\ \texttt{then}\ y + (fz)\ \texttt{else}\ 0)\ \texttt{in}\ fs$$

Similarly, we can count the number of A named molecules in a solution:

$$\mathit{count}_A \triangleq \lambda s.\ \texttt{letrec}\ f = \texttt{case\_sol}\ s\ \texttt{of}\ x^y, z\ \texttt{then}\ \texttt{case\_mol}\ x\ \texttt{of}\ A(x_1, \dots, x_k)\ \texttt{then}\ 1 + (fz)\ \texttt{else}\ (fz)\ \texttt{else}\ 0\ \texttt{in}\ fs$$

It is also possible to have a function that receives a molecule and a solution and counts the number of this molecule in a solution. For instance, $\mathit{count\_mol}\ A()\ A(), B(), A()$ is supposed to evaluate to 2.

$$\mathit{count\_mol} \triangleq \lambda m \lambda s.\ \texttt{letrec}\ f = \texttt{case\_sol}\ s\ \texttt{of}\ x^y, z\ \texttt{then}\ (\texttt{if}\ x = m\ \texttt{then}\ 1 + (fz)\ \texttt{else}\ (fz))\ \texttt{else}\ 0\ \texttt{in}\ fs$$

Our next objective is to define function *count* as needed to define the Mass action kinetics. Here we use the additional function constant $\mathit{binom} : \texttt{nat}_0 \to \texttt{nat}_0 \to \texttt{nat}_0$ that computes binomial coefficients. This means that we assume that the

$$(\text{T-VAR})\ \frac{x:\tau \in \Gamma}{\Gamma \vdash x:\tau_1} \quad (\text{T-FUN})\ \frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash \lambda x.e:\tau_1 \to \tau_2} \quad (\text{T-SPEC})\ \frac{A:\tilde{\tau} \in \mathcal{A} \quad \Gamma \vdash \tilde{e}:\tilde{\tau}}{\Gamma \vdash A(\tilde{e}):A(\tilde{\tau})}$$

$$(\text{T-APP})\ \frac{\Gamma \vdash e_1:\tau_1 \to \tau_2 \quad \Gamma \vdash e_2:\tau_1}{\Gamma \vdash e_1 e_2:\tau_2} \quad (\text{T-REC})\ \frac{\Gamma, x:\tau' \vdash e_1:\tau' \quad \Gamma, x:\tau' \vdash e_2:\tau}{\Gamma \vdash \texttt{letrec}\ x = e_1\ \texttt{in}\ e_2:\tau}$$

$$(\text{T-CONST})\ \frac{c:\tau \in \mathcal{C}}{\Gamma \vdash c:\tau} \quad (\text{T-MATCH})\ \frac{\Gamma \vdash e_1:\texttt{mol}A:\tilde{\tau'} \in \mathcal{A} \quad \Gamma, \tilde{x}:\tilde{\tau'} \vdash e_2:\tau \Gamma \vdash e_3:\tau}{\Gamma \vdash \texttt{case\_mol}\ e_1\ \texttt{of}\ A(\tilde{x})\ \texttt{then}\ e_2\ \texttt{else}\ e_3:\tau}$$

$$(\text{T-MOL})\ \frac{\Gamma \vdash e:A(\tilde{\tau})}{\Gamma \vdash e:\texttt{mol}} \quad (\text{T-SELF})\ \frac{}{\Gamma \vdash \texttt{current\_sol}:\texttt{sol}} \quad (\text{T-SOL-ELEM})\ \frac{\Gamma \vdash e:\texttt{mol}}{\Gamma \vdash e:\texttt{sol}}$$

$$(\text{T-SOL-APPEND})\ \frac{\Gamma \vdash e:\texttt{sol} \quad \Gamma \vdash e':\texttt{sol}}{\Gamma \vdash e, e':\texttt{sol}} \quad (\text{T-RULE-SET})\ \frac{\forall r \in R \quad \vdash r}{\vdash R}$$

$$(\text{T-MULT})\ \frac{\Gamma \vdash e_1:\texttt{sol} \quad \Gamma, x:\texttt{mol}, y:\texttt{nat}, z:\texttt{sol} \vdash e_2:\tau \quad \Gamma \vdash e_3:\tau}{\Gamma \vdash \texttt{case\_sol}\ e_1\ \texttt{of}\ x^y, z\ \texttt{then}\ e_2\ \texttt{else}\ e_3:\tau}$$

$$(\text{T-RULE})\ \frac{\begin{array}{c} dom(\Gamma) = (fn(P) \cup fn(e)) \backslash \{\tilde{x}\} \\ \Gamma, \tilde{x}:\texttt{link} \vdash p:\texttt{sol} \quad \Gamma, \tilde{x}:\texttt{link} \vdash e:\texttt{real} \quad \Gamma, \tilde{x}, \tilde{y}:\texttt{link} \vdash s:\texttt{sol} \end{array}}{\vdash (\nu\tilde{x})p \xrightarrow{e} (\nu\tilde{y})s}$$

Fig. 11. Type system for expressions and rules.

constraint language C provides this constant, such that for all natural numbers n and m and solutions s it holds that $[[binom]]_s(n,m) = \binom{n}{m}$.

$$\begin{array}{rl} count \triangleq \lambda s_1 \lambda s_2.\ \texttt{letrec}\ f = & \texttt{case\_sol}\ s_1\ \texttt{of}\ x^y, z\ \texttt{then} \\ & (binom\ (count\_mol\ x\ s_2) \\ & \quad (count\_mol\ x\ s_1)) * (f z) \\ & \texttt{else}\ 0\ \texttt{in}\ f s \end{array}$$

Reaction rules are enriched with ν-binders on the left-hand side, similarly to the κ-calculus. They have thus the form $(\nu\tilde{x})p \xrightarrow{e} (\nu\tilde{y})s$. ν-bound variables on the right hand side should not occur elsewhere in the rule. ν-bound variables on the left hand side must match link names that are entirely removed from the solution by rule application, see Fig. 13. Typing rule (T-RULE) for rules ensures that ν-bound variables have `link` type, that reactant and product patterns are of `sol` type and that e has `real` type, see Fig. 11. For a well-typed rule set R and a solution s, typing is preserved by reduction.

Proposition 2 (Subject reduction). *Let R be a rule set and s be a solution, if $\Gamma \vdash s$ for some typing context Γ, and $\vdash R$, and $R \vdash s \xrightarrow{d} s'$ then $\Gamma \vdash s'$.*

6 Stochastic Simulator

We propose a stochastic simulator that applies to both $React^{=}$ and $React(C)$ and is independent of the choice of constraint system C. We also discuss the

$$\frac{}{v \Downarrow_s v} \qquad \frac{e_1 \Downarrow_s \lambda x.e \quad e_2 \Downarrow_s v' \quad e[v'/x] \Downarrow_s v}{e_1 e_2 \Downarrow_s v} \qquad \frac{e_1 \Downarrow_s v_1 \neq \lambda x.e \quad e_2 \Downarrow_s v_2 \quad v_1 v_2 \Downarrow_s v}{e_1 e_2 \Downarrow_s v}$$

$$\frac{c : \tau \in \mathcal{C} \quad ar(x) = n \quad e_1 \Downarrow_s v_1 \quad \ldots \quad e_n \Downarrow_s v_n}{c\ e_1 \ldots e_n \Downarrow_s [[c]]_s(v_1, \ldots, v_n)} \qquad \frac{e_2[e_1/x] \Downarrow_s v}{\texttt{letrec}\ x = e_1\ \texttt{in}\ e_2 \Downarrow_s v}$$

$$\frac{\tilde{e} \Downarrow_s \tilde{v}}{A(\tilde{e}) \Downarrow_s A(\tilde{v})} \qquad \frac{e_1 \Downarrow_s A(\tilde{v}') \quad e_2[\tilde{v'}/\tilde{x}] \Downarrow_s v}{\texttt{case\_mol}\ e_1\ \texttt{of}\ A(\tilde{x})\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow_s v}$$

$$\frac{e_1 \Downarrow_s B(\tilde{v}') \quad A \neq B \quad e_3 \Downarrow_s v}{\texttt{case\_mol}\ e_1\ \texttt{of}\ A(\tilde{x})\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow_s v}$$

$$\frac{e_1 \Downarrow_s s_1 \quad s_1 \equiv \prod_{i=1}^{n} a_i^{m_i} \quad s_1 = a_1, s_1' \quad s_2 \equiv \prod_{i=2}^{n} a_i^{m_i} \quad e_2[a/x, m_1/y, s_2/z] \Downarrow_s v}{\texttt{case\_sol}\ e_1\ \texttt{of}\ x^y, z\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow_s v}$$

$$\frac{e_1 \Downarrow_s \mathbf{0} \quad e_3 \Downarrow_s v}{\texttt{case\_sol}\ e_1\ \texttt{of}\ x^y, z\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Downarrow_s v} \qquad \frac{e \Downarrow_s s \quad e' \Downarrow_s s'}{e, e' \Downarrow_s s, s'} \qquad \frac{}{\texttt{current\_sol} \Downarrow_s s}$$

Fig. 12. Big-step evaluation of expressions.

$$\text{(INST)}\ \frac{\begin{array}{c}\sigma : fn((\nu\tilde{x})p) \cup fn(e) \to Vals \text{ type preserving} \quad e\sigma'\sigma \Downarrow d \\ \sigma' : \{\tilde{x}\} \to N \backslash N' \text{ injective} \quad \sigma'' : \{\tilde{x}'\} \to \mathcal{N} \backslash (N \cup fn((\nu\tilde{x})p) \cup fn(e)) \text{ injective}\end{array}}{(\nu\tilde{x})p \xrightarrow{e} (\nu\tilde{x}')s \Downarrow_{\sigma,\sigma',N,N'} p\sigma'\sigma \xrightarrow{d} s\sigma''\sigma}$$

$$\text{(REACT)}\ \frac{s_1' \approx s, s_1 \quad s, s_2 \equiv s_2' \quad d \in \mathbb{R}^+}{s_1 \xrightarrow{d} s_2 \vdash s_1' \xrightarrow{d} s_2'}$$

$$\text{(SUM)}\ \frac{s_1' \equiv s_1 \quad d = \sum_{r \in R} \sum_{\{(d', \sigma, \sigma') \mid r \Downarrow_{\sigma,\sigma',fn(s_1'),fn(s_2)} \rho,\ \rho \vdash s_1' \xrightarrow{d'} s_2\}} d'}{R \vdash s_1 \xrightarrow[\text{MA}]{d} s_2}$$

Fig. 13. Stochastic mass-action semantics of $React(C)$.

algorithmic complexity of a single simulation step. It should be noted that the efficient simulation algorithm for the κ-calculus [5], which updates matches of rules dynamically, cannot be generalized in any obvious manner, since hyper-edges spoil the principle of rigidity (unique matches for connected patterns).

A stochastic simulator allows to execute a system of chemical reaction rules R on a biochemical solution s. It then computes traces by repeatedly applying the reaction rules in R to the current solution, with s as the initial solution. Note that these traces also contain the time delays Δ for every step. Our simulator is given in Fig. 14. It may compute infinite traces but the overall simulation time could easily be limited. Given the current solution s, the rule set R, and the current time point t, our simulator computes the set of applicable reactions (l, r, s') of R on s with their rates r and selects one of them non-deterministically by Gillespie's SSA algorithm [8] and also returns its time delay Δ. Note that

```
simulate(s,t)  // with system of reaction rules R
  let  Reacts = {(d,(l,r,s')) | r ∈ R, r ⇓_l ρ, ρ ⊢ s -d-> s'}  // compute all potential
       reaction steps
  let  (d,(l,r,s'),Δ) = SSA(Reacts)  // choose transition and time delay by SSA
  output(d,(l,r,s'),Δ)  // trace the chosen reaction
  simulate(s',t+Δ)
```

Fig. 14. Stochastic Simulator for $React^{=}$ and $React(C)$.

label l is some pair $(\sigma, fn(s'))$ for $React^{=}$ and some pair $(\sigma, \sigma', fn(s), fn(s'))$ for $React(C)$ where σ' takes care of the ν-binders on the left-hand side of r. The algorithm then outputs the selected step, its rate, and its delay and continues with s' at time point $t + \Delta$.

Algorithmically, the main problem to be solved by the simulator is to compute the set of applicable reactions for a given system R of rules and a solution s. The following proposition states that every step of the simulator can be done in polynomial time under the assumption that the maximal arity n of reaction rules is bounded. This result is relevant, since the encoding of definitions of the stochastic π-calculus produces only reactions rules of arity 2. We define the size $|r|$ of a rule as the number of its symbols, and similarly the size $|s|$ of a solution.

Proposition 3. *Let r be a reaction rule $(\nu\tilde{x})\prod_{i=1}^{n} A_i(\tilde{x}_i) \xrightarrow{e} (\nu\tilde{x}')p'$ and s a solution $\prod_{j=1}^{m} A'_j(\tilde{v}'_j)$ then the set of possible instantiations $\{l \mid r \Downarrow_l \rho,\ \rho \vdash s \xrightarrow{d} s'\}$ can be computed in time $O(|r| + |s| + m^n)$.*

Proof. We enumerate all injective functions $\ell : \{1, \ldots, n\} \to \{1, \ldots, m\}$ and tests whether they define a redex. There are m^n many of such functions and testing whether ℓ is a redex costs time $O(|s| + |r|)$. Again this is enough, since all free variables of r occur freely on the left hand side. □

Note that a naive approach that enumerates all possible assignments of pattern variables to values in s leads to an algorithm in $O(|r|^{|s|})$ which is exponential in the solution size and thus unfeasible.

The next proposition shows that we cannot obtain a simulator with steps in polynomial time, neither for $React(C)$ nor for $React^{=}$, without imposing additional restrictions such as a bound on the maximal arity of reaction rules. The input is a reaction rule r and a solution s and the output is "yes" if and only if r is applicable to s, i.e., if there exists a substitution σ such that $r \Downarrow_{\sigma, fn(s)} \rho$ and $\rho \vdash s \xrightarrow{d} s'$.

Proposition 4. *The reaction-applicability problem is* NP*-complete for both $React^{=}$ and $React(C)$.*

Proof. The generate and test algorithm in the Proof of Proposition 3 can be run in non-deterministic polynomial time, so reaction applicability is in NP. In

order to prove NP-hardness, we show that 3SAT can be reduced to reaction-applicability in polynomial time. We illustrate the ideas of our encoding at the following two 3SAT clauses as an example.

$$(b_1 \vee \overline{b_2}) \wedge (b_1 \vee b_2 \vee \overline{b_3})$$

We now express theses clauses by a reaction rule $p \xrightarrow{d} s'$ that is supposed to match a solution s. For each of the two clauses C_i where $1 \leq i \leq 2$, we fix a variable x_i which will match either of the three Boolean variable b_1, b_2, b_3 and that z_i matches the value of this Boolean variable in variable assignments satisfying the clauses. We use molecule names in $\mathcal{A} = \{C_i, E_{ij} \mid 1 \leq i, j \leq 2\}$.

1. The first clause is expressed by adding a reactant $C_1(x_1, z_1)$ to pattern p and molecules $C_1(b_1, 1), C_1(b_2, 0)$ to the solution s.
2. The second clause is expressed by adding a reactant $C_2(x_2, z_2)$ to pattern p and molecules $C_2(b_1, 1), C_2(b_2, 1), C_2(b_3, 0)$ to the solution s.
3. In order to express that z_i must match the Boolean that is assigned to the Boolean variable matching x_i, we encode condition $x_i = x_j \Rightarrow z_i = z_j$ for all $1 \leq i < j \leq 2$. This can be done by adding the reactants $E_{ij}(x_i, x_j, z_i, z_j)$ to pattern p and the following molecules to solution s:
$$\prod_{\beta,\beta' \in \mathbb{B}} \prod_{1 \leq k \neq l \leq 3} E_{ij}(b_k, b_k, \beta, \beta), E_{ij}(b_k, b_l, \beta, \beta')$$
So if x_i and x_j match the same b_k then z_i and z_j must match the same Boolean β. Otherwise, there is no restriction.

Pattern p grows linearly in the size of the clauses, while solution s grows both, quadratically with the number of clauses and quadratically with the number of Boolean variables, and thus polynomially in the size of the clauses. □

Computing matching redexes by constraint programming. We propose to use constraint programming in order to find an algorithm that computes the set of applicable reactions for a given system R of reaction rules and a biochemical solution s with a complexity less than the worst complexity $O(|r| + |s| + m^n)$. This is relevant, since this algorithm will always need quadratic time for each step of binary rules, while one would hope for linear time in many cases.

Rather than generating all redex candidates ℓ and then testing whether ℓ is indeed a redex of r and s, we define a constraint that states whether a redex candidate for a solution is indeed a redex and then solve this constraint by constraint programming, i.e. by propagating and splitting rather than generating and testing. For a given solution $s = \prod_{i=1}^{n} a_i^{m_i}$ and a reaction $\prod_{i=1}^{m} p_i \xrightarrow{e} s'$, the constraint $\psi(r, s)$ is defined as follows:

$$\psi(\textstyle\prod_{i=1}^{m} p_i \xrightarrow{e} s',\ \prod_{i=1}^{n} a_i^{m_i}) \ = \ e \Downarrow d \in \mathbb{R}^+ \ \wedge_{i=1}^{m} I_i \in \{1, \ldots, n\} \wedge p_i = a_{I_i} \wedge \wedge_{i=1}^{m} \#\{j \mid I_i = I_j\} \leq m_i$$

We use finite domain variables I_i and so called element constraints for expressing $\wedge_{i=1}^{m} I_i \in \{1, \ldots, n\} \wedge p_i = a_{I_i}$ which states that all p_i match a_{I_i} (line 2). Strong propagators for element constraints are provided by all current constraint programming libraries. Additional requirements are that the number of patterns matched to the same molecule must not exceed the number of that molecule in the solution (line 3) and that e evaluates to a successful value (line 1).

7 Conclusion

We introduced a new language of biochemical reaction rules with constraints *React*(C) that is highly expressive. We sowed that with equality constraints and hyperedges the missing features for subsuming the expressiveness of the stochastic π-calculus are provided. Besides constraints *React*(C) supports reflexivity, which enables modelers to define arbitrary kinetics.

We presented a simulator for *React*(C) that computes steps in polynomial time, under the assumption that the arity of reaction rules is bounded. We showed that efficient simulation is impossible without this assumption. A constraint programming solution that may often avoid the higher polynomials in the worst case was presented. An implementation is under way.

In future work, we would like to show that the attributed π-calculus $\pi(C)$ can be encoded in *React*(C) restricted to binary rules. Furthermore, we conjecture that the imperative π-calculus can be encoded into *React*(C) restricted to ternary rules. This would prove that *React*(C) subsumes BioAmbients as well. The relationship of *React*(C) to Bigraphs is also to be elaborated.

References

1. ML. Blinov, JR. Faeder, B. Goldstein, and WS. Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.
2. L. Cardelli. From processes to odes by chemistry. In *IFIP TCS*, volume 273 of *IFIP*, pages 261–281. 2008.
3. N. Chabrier-Rivier, F. Fages, and S. Soliman. The biochemical abstract machine biocham. In *Computational Methods in Systems Biology, International Conference CMSB 2004*, volume 3082 of *LNCS*, pages 172–191. 2005.
4. V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Abstracting the differential semantics of rule-based models: Exact and automated model reduction. In *25th LICS*, pages 362–381. IEEE Press, 2010.
5. V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks. In *Programming Languages and Systems, 5th Asian Symposium*, volume 4807 of *LNCS*, pages 139–157. 2007.
6. V. Danos and C. Laneve. Formal molecular biology. *TCS*, 325(1):69–110, 2004.
7. C. Fournet and G. Gonthier. The reflexive cham and the join-calculus. In *POPL*, pages 372–385, ACM press, 1996.
8. DT. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403 – 434, 1976.
9. SW. Gilroy and MD. Harrison. SBML: a user interface mark-up language based on interaction style. *Int. J. Web Eng. Technol.*, 4(2):207–234, 2008.
10. M. John, C. Lhoussaine, and J. Niehren. Dynamic compartments in the imperative pi calculus. In *Computational Methods in Systems Biology, 7th International Conference*, volume 5688 of *LNCS*, pages 235–250. 2009.
11. M. John, C. Lhoussaine, J. Niehren, and A. Uhrmacher. The attributed pi calculus with priorities. *Trans. on Computational Systems Biology*, 5945(XII):13–76, 2010.

12. J. Krivine, V. Danos, and A. Benecke. Modelling epigenetic information maintenance: A kappa tutorial. In *21st CAV*, volume 5643 of *LNCS*, pages 17–32. 2009.
13. C. Kuttler, C. Lhoussaine, and M. Nebut. Rule-based modeling of transcriptional attenuation at the tryptophan operon. *Trans. on Computational Systems Biology*, 5945(XII):199–228, 2010.
14. C. Kuttler, C. Lhoussaine, and J. Niehren. A stochastic pi calculus for concurrent objects. In *Second International Conference on Algebraic Biology*, volume 4545 of *LNCS*, pages 232–246. 2007.
15. C. Laneve, S. Pradalier, and G. Zavattaro. From biochemistry to stochastic processes. *ENTCS*, 253(3):167–185, 2009.
16. N. Papanikolaou. The space and motion of communicating agents author: Robin Milner *SIGACT News*, 41(3):51–55, 2010.
17. A. Phillips and L. Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *Computational Methods in Systems Biology, International Conference*, volume 4695 of *LNCS*, pages 184–199. 2007.
18. C. Priami. Stochastic pi-calculus. *Comput. J.*, 38(7):578–589, 1995.
19. S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: Stochastic simulation of large-scale genetic regulatory networks. *J. Bioinformatics and Computational Biology*, 3(2):415–436, 2005.
20. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Y. Shapiro. Bioambients: an abstraction for biological compartments. *TCS*, 325(1):141–167, 2004.
21. A. Regev and E. Shapiro. Cells as Computation. *Nature*, 419:343, 2002.
22. A. Romanel and C. Priami. On the computational power of BlenX. *TCS*, 411(2):542–565, 2010.
23. C. Versari. A core calculus for a comparative analysis of bio-inspired calculi. In *ESOP*, volume 4421 of *LNCS*, pages 411–425. 2007.