A Rule-Based and Imperative Language for Biochemical Modeling and Simulation

Đurica Nikolić[1,2], Corrado Priami[1,3], and Roberto Zunino[1,3]

[1] The Microsoft Research - University of Trento Centre for Computational and Systems Biology
[2] University of Verona, Italy
[3] University of Trento, Italy

Abstract. We present COSBI LAB Language ($\mathcal{L}$ for short), a simple modeling language for biochemical systems. $\mathcal{L}$ features stochastic multiset rewriting, defined in part through rewriting rules, and in part through imperative code.

We provide a continuous-time Markov chain semantics for $\mathcal{L}$ at three different abstraction levels, linked by Galois connections. We then describe a simulation algorithm for the most concrete semantics, which is then adapted to work at higher abstract levels while improving space and time performance. Doing so results in the well-known Gillespie's Direct Method, as well as in a further optimized algorithm.

1 Introduction

In this paper we present a computer language, called $\mathcal{L}$, for modeling and simulating biochemical systems. In such setting, we are concerned with the modeling of the kinds of behaviour leading to the creation of *biomolecular complexes* and their mutual interaction. Complexes are to be thought as an aggregation of smaller molecules, kept together by chemical bonds on specific zones called *interaction sites*.

Different kinds of mathematical structures have been used to model such entities. Often, these take inspiration from *graphs* and their generalizations, e.g., hypergraphs. Here, the smaller molecules are represented by graph nodes, which are taken as primitive stateful entities. We shall name these stateful nodes "*boxes*". Boxes also have a list of sites, from which they can be connected to other boxes by (undirected) edges. In this framework, complexes are just the connected components of the graphs. System evolution is then typically modeled via a stochastic transition system, hence providing a semantics based on continuous-time Markov chains; this is done to capture the inherent uncertainty of the biological phenomena, which are "noisy" in their nature. The actual definition of the transition system depends on the modeling language at hand. For instance, BlenX [5] is a language which uses a graph-like representation of complexes, whose boxes are equipped with a process in a *stochastic process algebra*. There, the stochastic operational semantics of the processes inside the boxes form the basis for defining the transition system for graphs. Kappa [2] instead resorts to stochastic *rewriting rules*, borrowing from graph rewriting techniques. These rules can be used to express in an intuitive way how the graph is affected by biochemical reactions.

Often, however, the precise graph structure of complexes, their chemical bonds, and the nature of the interaction sites is still unknown to researchers in biology. Modelers

G. Eleftherakis, M. Hinchey, and M. Holcombe (Eds.): SEFM 2012, LNCS 7504, pp. 16–32, 2012.

wishing to use graph-based modeling languages are then asked to provide more data than those available. In such cases, it would be better to use a less detailed structure to represent complexes. In our $\mathcal{L}$ language we use box *multisets* in lieu of box graphs. Consequently, interaction sites are no longer represented, as well as exact chemical bonds. Rather, we just represent the fact that two molecules belong to the same complexes (or to different ones). This approach seem to be closer to the actual knowledge available to researchers. As a bonus, we also get some performance improvements in simulation, since e.g. graph isomorphism tests (untractable, but quadratic time under certain assumptions [7]) are now replaced by multiset equality tests (linear time with most representations).

In $\mathcal{L}$, we use stochastic multiset rewriting rules to define the evolution of a system. Three kind of rules are used: association rules (assoc) which merge two complexes, dissociation rules (dissoc) which split them, and general dynamics dyn to model the rest of the interactions. These rules are defined using complex *patterns*, selecting which complexes are to be rewritten. Rewriting is, in part, automatic (for assoc and dissoc) and can be augmented by *imperative code* whenever one needs to describe custom multiset manipulations. The effect of this code is atomic: no other rule firing is interleaved. Having atomic "large" effects is useful, since otherwise such effects need to be carefully programmed across many rules, often using "infinite" stochastic rates and dealing with all the resulting concurrency issues [6]. For instance, if we want to model a vescicle releasing at once all its carried molecules (the number of which is not statically known), we can not use a basic rule, yet it is straightforward to program this behaviour.

We present the syntax of $\mathcal{L}$ in Sect. 2, and its semantics in Sect. 3. We then study how the language can be efficiently simulated. In order to do so, we apply *abstract interpretation* and construct two more abstract semantics in Sect. 4. We then apply this to construct efficient simulation algorithms for $\mathcal{L}$ in Sect. 5. We start from a simple, yet inefficient algorithm which is then adapted to exploit the abstract semantics. This results in improvements to space and time performance. The algorithms constructed in this way are the well-known Gillespie's Direct Method [4] as well as a new improvement of it which better takes advantage of the features of $\mathcal{L}$.

2 Syntax

In Figure 1 we show the syntax of the $\mathcal{L}$ language. We give a short comment on the constructs in that figure. $BasicType$ stands for a primitive type used in our language, while $BasicLiteral$ ranges over their values. The modeler can declare a *box* type, by specifying a name for it and of a set of *fields* having basic types. More formally, $FieldDecl$ is a sequence of field declarations of form $Field : BasicType$, where $Field$ is the name of the field, and is unique inside the box. The field list can be empty (ϵ). Then, the declaration of a box type $BoxDecl$ has the form $BoxType\{FieldDecl\}$, where $BoxType$ is a name for the declared type. Moreover, $FieldInit$ is a (possibly empty) sequence of initialized fields, while $BoxLiteral$ represents a box having all its declared fields instantiated. For example, $A\{\texttt{x} : \texttt{int};\ y\ :\ \texttt{real}\}$ is a declaration of a box type A containing fields x and y of the given types, and $A\{x = 3;\ y = 1.0\}$ is an instantiation. We use Box to denote the set of all possible box instantiations.

BasicType	::= `bool` | `int` | `real`
BasicLiteral	::= *BoolLiteral* | *IntLiteral* | *RealLiteral*
Field	::= *Ide*
FieldDecl	::= ϵ | *Field* : *BasicType*; *FieldDecl*
FieldInit	::= ϵ | *Field* = *BasicLiteral*; *FieldInit*
Exp	::= *BasicLiteral* | null | *Ide* | *Exp* $\wedge$ *Exp* | $\neg$*Exp*
	| *Exp* + *Exp* | *Exp* − *Exp* | *Exp* ∗ *Exp* | *Exp* = *Exp* | *Exp* < *Exp*
	| *Exp*.*Field* | *Exp*.count(*BoxType*) | *Exp*.first(*BoxType*)
BoxType	::= *Ide*
BoxDecl	::= *BoxType*{*FieldDecl*}
BoxLiteral	::= *BoxType*{*FieldInit*} NOTE: all declared fields **must** be instantiated
CplxLiteral	::= [*IntLiteral* : *BoxLiteral*; *CplxLiteralTail*]
CplxLiteralTail	::= ϵ | *IntLiteral* : *BoxLiteral*; *CplxLiteralTail*
Complexes	::= *IntLiteral* : *CplxLiteral*; *ComplexesTail*
ComplexesTail	::= ϵ | *IntLiteral* : *CplxLiteral*; *ComplexesTail*
Pattern	::= [*BoxType*{*FieldInit*} *PatternTail*]
PatternTail	::= ϵ | , *BasePattern*
Assoc	::= assoc *Pattern Pattern* rate *Exp* react *Block*
Dissoc	::= dissoc *Pattern* $Pattern^{no*}$ rate *Exp* react *Block*
Dyn	::= dyn $Pattern_1 \ldots Pattern_n$ rate *Exp* react *Block*
Block	::= *var Ide* := *Exp*; *Block* | *CmdBlock*
CmdBlock	::= end | *Cmd* ; *CmdBlock*
Cmd	::= skip | *Ide* := *Exp* | if *Exp* then *Block* else *Block*
	| while *Exp do Block* | *BoxCommand*
BoxCommand	::= *Ide* := *Exp*.spawn(*BoxLiteral*, ...) | *Exp*.spawn(*Exp*)
	| *Exp*.remove(*Exp*) | *Exp*.merge(*Exp*) | *Exp*.move(*Exp*, *Exp*)
	| foreach *Ide* : *BoxType* in *Exp* do *Block* | *Exp.Ide* := *Exp*
Decl	::= *Assoc* | *Dissoc* | *Dyn* | *BoxDecl*
Run	::= run *Complexes* end
Model	::= *Run* | *Decl*; *Model*

Fig. 1. Syntax of the $\mathcal{L}$ language

For any set S, we write mset S for the set of multisets over S, which we sometimes identify with the set of functions $S \to \mathbb{N}$. A *complex* is a multiset of boxes, which we represent in our syntax by a *CplxLiteral*. The latter is a non-empty sequence of the form *IntLiteral* : *BoxLiteral*, where *IntLiteral* denotes how many instances of *BoxLiteral* are present in the complex. When *IntLiteral* = 1, we omit to write it. For instance, $[2 : A\{x = 3;\ y = 1.0\}, B\{\}]$ is a complex. We use Cplx = mset Box to denote the set of all possible complexes. The whole system state is then defined via the *Run* clause, which specifies an initial sequence of *Complexes*, having form *IntLiteral* : *CplxLiteral* where *IntLiteral* represents the initial population of the complex in the system at hand.

The dynamics of the system is given by multiset rewriting rules, which continuously modify the system at hand (if no rule applies, the system does not evolve further). Our rules are based on complex patterns. A *Pattern* is a sequence of literals of form *BoxType*{*FieldInit*}, possibly followed by a wildcard ∗. Intuitively, a pattern without ∗ matches with complexes having exactly the specified boxes, while the wildcard allows matching with complexes including other boxes as well. More formally, we say that a box $B_1\{f_1 = v_1, \ldots, f_n = v_n\}$ matches with a box $B_2\{g_1 = h_1, \ldots, g_m = h_m\}$ if $B_1 = B_2$,

$n \leq m$ and for each $i \in [1..n]$ there exists $j \in [1..m]$ such that $f_i = g_j$ and $v_i = h_j$. Then, we say that a complex $c \in \mathsf{Cplx}$ matches with a pattern p, denoted with $c \models p$, if one of the following conditions holds:

- p does not end with $*$, and there is a *bijective* correspondence between boxes in p and those in c, where correspondent boxes match.
- p does end with $*$, and there is an *injective* correspondence between boxes in p and those in c, where correspondent boxes match.

The following example illustrates some pattern matchings.

Example 1. Consider complexes $c_1 = [A\{x = 1\}]$, $c_2 = [B\{\}]$, $c_3 = [A\{x = 0\}, A\{x = 1\}]$, $c_4 = [A\{x = 1\}, B\{\}]$ and $c_5 = [A\{x = 1, y = 4\}]$ and patterns $p_1 = [A]$, $p_2 = [A, A]$, $p_3 = [A, *]$, $p_4 = [B]$, $p_5 = [B, *]$ and $p_6 = [A\{x = 1\}]$. Then, only the following relations hold: $c_1 \models p_1$, $c_1 \models p_3$, $c_1 \models p_6$, $c_2 \models p_4$, $c_2 \models p_5$, $c_3 \models p_2$, $c_3 \models p_3$, $c_4 \models p_3$, $c_4 \models p_5$, $c_5 \models p_1$, $c_5 \models p_3$, $c_5 \models p_6$. ■

Having defined patterns, we can now discuss the rewriting rules which lead the evolution of the system. Our language features three kinds of such rules, namely *Assoc*, *Dissoc* and *Dyn*. A rule assoc p_1 p_2 rate *Exp* react *Block* allows pairs of complexes matching with p_1 and p_2 to associate. When that happens, the two reactant complexes merge their boxes and form a new larger complex, mimicking the association of two macromolecules in biological systems. The rate expression *Exp* provides the rate for the stochastic transition, thus defining the "speed" of the association. The rate expression is allowed to inspect the boxes in the two reactants via two special variables $\texttt{reactant}_1$ and $\texttt{reactant}_2$. For instance, assoc $[A]$ $[B, *]$ rate $5.2 * \texttt{reactant}_1.\mathsf{first}(A).\texttt{mass}$ defines a rate proportional to the mass of the first reactant. When an assoc rule is fired, after the complexes are associated the code block specified in the react part is run. This can access the newly formed product (via a special `product` variable) and modify it further, e.g. by changing box fields, or adding/removing boxes. The react code block can also spawn entirely new complexes.

A rule dissoc p_1 p_2 rate *Exp* react *Block* specifices the dual operation, namely dissociation of a complex into two subcomplexes. Here, p_1 specifies the complex to break up, while p_2 matches with a subcomplex to separate (no wildcard $*$ is allowed in p_2). The rate expression *Exp* can access $\texttt{reactant}_1$ to provide a dissociation stochastic rate, which is intended to define how fast is the reactant to split. In the case p_2 has multiple matches inside the reactant, we let all of them define an equally probable dissociation, hence effectively dividing the rate among all the possible splits. After the rule triggers and the split is performed, the react code block is run, and can access the new complexes using the variables $\texttt{product}_1$ and $\texttt{product}_2$.

Rule dyn is used to define a generic molecular dynamics. Its semantics is as for assoc, except that no complex merge is performed, and the react code block still has access to the unmerged complexes $\texttt{reactant}_1$ and $\texttt{reactant}_2$. This rule effectively subsumes assoc and dissoc, in that association/dissociation can be programmed manually in the code block. However, associations and dissocations are so common to deserve a special construct. Instead, the typical use for dyn is the modeling of monomolecular reactions, in which only one reactant is present.

The code blocks in rules are written in an imperative language, the constructs of which are mostly standard. Therefore, we just briefly discuss the more peculiar ones. Since the state of $\mathcal{L}$ is stored in complexes, i.e. in multisets of boxes, we need constructs to inspect and modify those. We provide a way to loop over all the boxes of a given type in a complex (foreach b : *BoxType* in *complex*). To precisely define the semantics of such loop, we require that the visit order follows the lexicographic order of box values. The expression *complex*.first(*BoxType*) returns the first box in such ordering. Further commands allows one to add (*complex*.spawn) and remove (*complex*.remove) boxes in a complex. Similar operations can be done at the complex level: new complexes can be created, and existing ones removed. We provide also ways to move boxes between complexes (move) as well as to merge two complex as it happens for association (merge).

Example 2. As a simple example, we provide an $\mathcal{L}$ model for the enzymatic reaction shown below:

$$\mathrm{E} + \mathrm{S} \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} \mathrm{ES} \xrightarrow{k_2} \mathrm{EI} \xrightarrow{k_3} \mathrm{E} + \mathrm{P}$$

The first double arrow models an enzyme molecule (E) associating and dissociating to a substrate molecule (S). When associated, the complex ES can react (second arrow): the enzyme changes the substrate into some intermediate molecule (I). This reaction is not reversible. Then, the intermediate molecule can dissociate from the enzyme, which releases a product (P) in the system (third arrow). In $\mathcal{L}$, we can model this behaviour as follows. Below, the react blocks are used to change S into I, and then I into P.

```
E{}   S{}   I{}   P{}
assoc [E] [S] rate k1;
dissoc [E,S] [S] rate k-1;
dyn [E,S] rate k2 react
    reactant1.remove(reactant1.first(S));
    reactant1.spawn(I{});
    end;
dissoc [E,I] [I] rate k3 react
    product2.remove(product2.first(I));
    product2.spawn(P{});
    end;
run 100 : [E]; 100 : [S]; end
```

■

3 Semantics

In this section we provide a semantics for the rules of our $\mathcal{L}$ language. To keep our presentation short, we focus on the semantics of the assoc rule, only. The formal semantics of the dissoc and dyn rules can be indeed defined similarly.

Suppose we are given a set of complexes annotated with their *names*. Names uniquely identify the instances of complexes present in the system, so that complexes comprised by exactly the same boxes are still distinguishable. For the sake of simplicity, we assume that complexes' names are natural numbers. Then, suppose that we are

given an assoc rule which states that all the pairs of complexes satisfying patterns p_1 and p_2 react with a stochastic rate specified by an expression e. The semantics of the assoc rule then collects the stochastic transitions involving all such pairs of complexes, and their corresponding rates.

We start by defining two auxiliary binary set operators, which are similar to the cartesian product. Operator $\otimes$ defines the set of *unordered* pairs ($\{x_1, x_2\}$), while operator $\hat{\otimes}$ defines the set of *unordered* pairs of *distinct* elements (hence requiring $x_1 \neq x_2$).

Definition 1 ($\otimes$ and $\hat{\otimes}$). *Given two arbitrary sets* S_1 *and* S_2*, we define*

$$\mathsf{S}_1 \otimes \mathsf{S}_2 = \{\{x_1, x_2\} \mid x_1 \in \mathsf{S}_1 \wedge x_2 \in \mathsf{S}_2\} \quad \mathsf{S}_1 \hat{\otimes} \mathsf{S}_2 = \{\{x_1, x_2\} \mid x_1 \in \mathsf{S}_1 \wedge x_2 \in \mathsf{S}_2 \wedge x_1 \neq x_2\}.$$

The following definition introduces the notion of system, representing a set of complexes enriched with their names, and the notion of transition, representing two different complexes that can react with a certain rate.

Definition 2 (Systems and Transitions). *A system* $\sigma^\flat$ *is a set of annotated complexes, i.e., ordered pairs* $\langle n, c\rangle \in \mathbb{N} \times \mathsf{Cplx}$*, where c and n represent a complex and its unique name, respectively. We write* $\mathsf{Sys}_0 = \wp(\mathbb{N} \times \mathsf{Cplx})$ *to denote the set of all the possible systems. A transition is an ordered pair whose first element determines the reactants, while the second element is the rate of the reaction. The reactants are characterized by an unordered pair of annotated complexes. We write* $\mathsf{Tr}_0 = ((\mathbb{N} \times \mathsf{Cplx}) \hat{\otimes} (\mathbb{N} \times \mathsf{Cplx})) \times \mathbb{R}^+$ *to denote the set of all possible transitions. Moreover, we let* $\mathsf{Res}_0 = \wp(\mathsf{Tr}_0)$.

It is worth noting that Res_0 denotes all possible sets of transitions, and therefore represents the set of all possible semantics of an assoc rule.

Example 3. Let $c_1 = [A\{x = 0\}]$, $c_2 = [A\{x = 1\}]$ and $c_3 = [B]$ be three complexes, and suppose that a system $\sigma^\flat$ is composed of 2, 1 and 2 instances of c_1, c_2 and c_3 respectively. Then, we represent $\sigma^\flat$ as follows: $\sigma^\flat = \{\langle 0, c_1\rangle, \langle 1, c_1\rangle, \langle 0, c_2\rangle, \langle 0, c_3\rangle, \langle 1, c_3\rangle\}$. If we suppose that $\langle 0, c_1\rangle$ and $\langle 0, c_2\rangle$ can react with a rate 1.0, the corresponding transition is given as: $\langle\{\langle 0, c_1\rangle, \langle 0, c_2\rangle\}, 1.0\rangle$. ■

We now define a function which selects from a system only those annotated complexes which match with a given pattern.

Definition 3. *Given a pattern p, we define a map* $[\![p]\!]^0 : \mathsf{Sys}_0 \to \mathsf{Sys}_0$ *called the evaluation of p in* Sys_0 *as:* $[\![p]\!]^0\sigma^\flat = \{\langle n, c\rangle \in \sigma^\flat \mid c \models p\}$*, for any* $\sigma^\flat \in \mathsf{Sys}_0$.

Example 4. Consider the system $\sigma^\flat$ defined in Example 3 and a pattern $[A]$ denoting the complexes composed of only one box whose its type is A. Since complexes c_1 and c_2 match with $[A]$, while c_3 does not, we have $[\![[A]]\!]^0\sigma^\flat = \{\langle 0, c_1\rangle, \langle 1, c_1\rangle, \langle 0, c_2\rangle\}$. ■

We now define to the actual semantics of the assoc rule. Its general form is the following: assoc p_1 p_2 rate e react *Block*. Starting from a system $\sigma^\flat$, we evaluate patterns p_1 and p_2 (Definition 3), obtaining two systems $\sigma_1^\flat$ and $\sigma_2^\flat$ whose complexes match with patterns p_1 and p_2 respectively. Our goal is to determine all possible transitions whose first reactant belongs to $\sigma_1^\flat$, and the second one belongs to $\sigma_2^\flat$. The rate of each

transition is determined by evaluating the expression e. This evaluation depends on an *environment*, i.e., a function which maps special variables $\texttt{reactant}_1$ and $\texttt{reactant}_2$ to their values. These values are the complexes which take part in the transition. The following definition explains how it is possible to evaluate the rate characterized by an expression e of a reaction between two complexes c_1 and c_2.

It is worth noting that the rate expression e should be "symmetric" in $\texttt{reactant}_1$ and $\texttt{reactant}_2$: swapping their values should not affect the resulting association rate. This reflect the fact that in biology association happens between unordered complex pairs. To stress and enforce this symmetry, below we compute the rate as the average of the rates resulting from both orders.

Definition 4. *Let c_1 and c_2 be two complexes and let e be an expression characterizing the rate of the reaction between these complexes. We determine the rate of this reaction as follows:*

$$\mathsf{rate}(c_1, c_2, e) = \frac{[\![e]\!]^{\mathsf{exp}}\rho_1 + [\![e]\!]^{\mathsf{exp}}\rho_2}{2},$$

where $\rho_1 = [\texttt{reactant}_1 \mapsto c_1, \texttt{reactant}_2 \mapsto c_2]$ *and* $\rho_2 = [\texttt{reactant}_1 \mapsto c_2, \texttt{reactant}_2 \mapsto c_1]$.

Given an assoc rule and a system, we define the set of all possible transitions that can occur between complexes of the system.

Definition 5 (Semantics of assoc). *Consider a rule* assoc p_1 p_2 rate e react B *and a state* $\sigma^\flat \in \mathsf{Sys}_0$. *We define the map* $[\![p_1, p_2, e]\!]_0^{\mathsf{assoc-nc}} : \mathsf{Sys}_0 \to \mathsf{Res}_0$ *as*

$$[\![p_1, p_2, e]\!]_0^{\mathsf{assoc-nc}}\sigma^\flat = \left\{\langle\{\langle n_1, c_1\rangle, \langle n_2, c_2\rangle\}, \mathsf{rate}(c_1, c_2, e)\rangle \in \mathsf{Tr}_0 \mid \langle n_1, c_1\rangle \in [\![p_1]\!]^0\sigma^\flat \wedge \langle n_2, c_2\rangle \in [\![p_2]\!]^0\sigma^\flat\right\}.$$

The collecting semantics $[\![p_1, p_2, e]\!]_0^{\mathsf{assoc}} : \wp(\mathsf{Sys}_0) \to \wp(\mathsf{Res}_0)$ *is defined as follows:* $[\![p_1, p_2, e]\!]_0^{\mathsf{assoc}}\Sigma^\flat = \left\{[\![p_1, p_2, e]\!]_0^{\mathsf{assoc-nc}}\sigma^\flat \mid \sigma^\flat \in \Sigma^\flat\right\}$.

Note that the semantics of the assoc rule contains only transitions of type Tr_0, i.e., the ones in which reactants must be different. This does not prevent two identical complexes to react, since they have different names. However, a complex is prevented to react with itself, which would be unwanted.

Example 5. Let us consider, one more time, the system $\sigma^\flat$ defined in Example 3 and let us determine the semantics of the rules

$$\mathsf{rule}_1 : \text{ assoc } [A]\ [B] \text{ rate } 1.0 \qquad \mathsf{rule}_2 : \text{ assoc } [A]\ [A] \text{ rate } 1.0.$$

In Example 4 we showed that $[\![[A]]\!]^0\sigma^\flat = \{\langle 0, c_1\rangle, \langle 1, c_1\rangle, \langle 0, c_2\rangle\}$. We can, similarly show that $[\![[B]]\!]^0\sigma^\flat = \{\langle 0, c_3\rangle, \langle 1, c_3\rangle\}$. The following table represents the semantics of the rules: rule_1 gives rise to transitions 1.-6. from the following table, while rule_2 gives rise to transitions 7.-9.

rule_1	1.	$\langle\{\langle 0, c_1\rangle, \langle 0, c_3\rangle\}, 1\rangle$	2.	$\langle\{\langle 0, c_1\rangle, \langle 1, c_3\rangle\}, 1\rangle$	3.	$\langle\{\langle 1, c_1\rangle, \langle 0, c_3\rangle\}, 1\rangle$
	4.	$\langle\{\langle 1, c_1\rangle, \langle 1, c_3\rangle\}, 1\rangle$	5.	$\langle\{\langle 0, c_2\rangle, \langle 0, c_3\rangle\}, 1\rangle$	6.	$\langle\{\langle 0, c_2\rangle, \langle 1, c_3\rangle\}, 1\rangle$
rule_2	7.	$\langle\{\langle 0, c_1\rangle, \langle 0, c_2\rangle\}, 1\rangle$	8.	$\langle\{\langle 1, c_1\rangle, \langle 0, c_2\rangle\}, 1\rangle$	9.	$\langle\{\langle 0, c_1\rangle, \langle 1, c_1\rangle\}, 1\rangle$

Transitions 1.-6. occur between a complex matching with $[A]$ and a complex matching with $[B]$ and their rate is 1, while in transitions 7.-9. both complexes match with $[A]$. ■

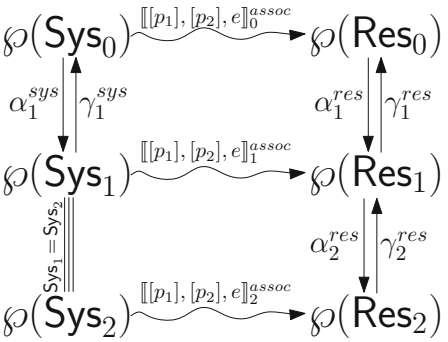

Fig. 2. Relationships between different levels of abstraction

Below, we state a simple property of the semantics of our expressions and commands. As it happens in nominal calculi and in most languages with references, we have that the actual names (n) used to annotate the "objects" (the complexes $\langle n, c\rangle$) are immaterial.

Lemma 1. *The semantics of expressions, commands, and rewriting rules are stable w.r.t. the renaming of annotated complexes. In particular, renaming does not affect the computed transition rates for rules, or their modifications of the state.*

4 Abstraction

There might be a huge number of annotated complexes satisfying the two patterns specified by an assoc rule that react. Hence, the number of reactions they give rise can becomes enormous. Although precise, the provided semantics for rules can lead to an inefficient simulation, as we will see. In this section we define two levels of abstraction which we shall exploit to perform some optimization in the simulation of $\mathcal{L}$ models. We call the actual semantics of the assoc rules defined in Section 3 the semantics of *level 0*, while we refer to our new abstraction levels as *level 1* and *level 2*. For both of them, we define some opportune abstractions of Sys_0 (Sys_1 and Sys_2) and Res_0 (Res_1 and Res_2), show how they can be related (via pairs of functions α^{sys}–γ^{sys} and α^{res}–γ^{res}) and define two abstract semantics ($[\![p_1, p_2, e]\!]_1^{\mathsf{assoc}}$ and $[\![p_1, p_2, e]\!]_2^{\mathsf{assoc}}$) representing a sound approximation of the actual semantics $[\![p_1, p_2, e]\!]_0^{\mathsf{assoc}}$ of the assoc rules. In Figure 2 we give a brief outline of this idea which we then formalize in Sections 4.1 and 4.2.

4.1 Abstraction at Level 1

In Section 3 we have stated that the information about complexes' names is not relevant for the actual semantics of the assoc rules. Therefore, it is possible to abstract away this piece of information and define another, more abstract semantics of these rules. In the following we will define counterparts of Definitions 2, 3 and 5.

Although the names of complexes are not relevant for the computation of reaction rates, it is important to know the exact amount of each complex present in a system,

i.e. its population. This fact gives rise to another definition of the notion of system, on a different level of abstraction. More precisely, for each complex, we memorize the number of instances of that complex present in a system. Then we remove information about complexes' names from the definition of transitions as well. The following definition formalizes these intuitions.

Definition 6. *A system σ is a multiset of complexes, i.e., $\sigma \in \mathsf{mset\ Cplx} = \mathsf{Sys}_1$. A transition is an ordered pair whose first element determines the set of reactants, while the second element is the rate of the reaction. The set of reactants can be composed of* 1 *or* 2 *elements. The former case occurs when two identical complexes react, while the latter case occurs when the reactants are not identical. We write $\mathsf{Tr}_1 = (\mathsf{Cplx} \otimes \mathsf{Cplx}) \times \mathbb{R}^+$ to denote the set of all possible transitions. Moreover, we let $\mathsf{Res}_1 = \wp(\mathsf{Tr}_1)$.*

Example 6. Under the hypotheses of Example 3 regarding the structure of complexes c_1, c_2 and c_3 and their populations which are, respectively 2, 1 and 2, we define the state $\sigma = [c_1 \mapsto 2, c_2 \mapsto 1, c_3 \mapsto 2]$. ■

Definition 7. *We define a function function* $\mathsf{names} : \mathsf{Cplx} \times \mathsf{Sys}_0 \to \wp(\mathbb{N})$ *which for every complex $c \in \mathsf{Cplx}$ and every system $\sigma^\flat \in \mathsf{Sys}_0$ returns all possible names that c might have in $\sigma^\flat$ as $\mathsf{names}(c, \sigma^\flat) = \{n \in \mathbb{N} \mid \langle n, c \rangle \in \sigma^\flat\}$.*

It is worth noting that $\langle \wp(\mathsf{Sys}_0), \subseteq, \cup, \cap, \varnothing, \mathsf{Sys}_0 \rangle$, and $\langle \wp(\mathsf{Sys}_1), \subseteq, \cup, \cap, \varnothing, \mathsf{Sys}_1 \rangle$ (or shortly, $\langle \wp(\mathsf{Sys}_0), \subseteq \rangle$ and $\langle \wp(\mathsf{Sys}_1), \subseteq \rangle$) are complete lattices. In the following we show that they are also related by a Galois connection [1].

Definition 8. *Let $\beta_1^{sys} : \mathsf{Sys}_0 \to \mathsf{Sys}_1$ be defined as $\beta_1^{sys}(\sigma^\flat) = \lambda c.\ |\mathsf{names}(c, \sigma^\flat)|$. We define the abstraction map $\alpha_1^{sys} : \wp(\mathsf{Sys}_0) \to \wp(\mathsf{Sys}_1)$ and the concretization map $\gamma_1^{sys} : \wp(\mathsf{Sys}_1) \to \wp(\mathsf{Sys}_0)$ as:*

$$\alpha_1^{sys}(\Sigma^\flat) = \{\beta_1^{sys}(\sigma^\flat) \mid \sigma^\flat \in \Sigma^\flat\} \qquad \gamma_1^{sys}(\Sigma) = \bigcup\{\Sigma^\flat \mid \alpha_1^{sys}(\Sigma^\flat) \subseteq \Sigma\}.$$

Lemma 2. $\left\langle \langle \wp(\mathsf{Sys}_0), \subseteq \rangle, \alpha_1^{sys}, \gamma_1^{sys}, \langle \wp(\mathsf{Sys}_1), \subseteq \rangle \right\rangle$ *is a Galois connection.*

The abstraction map α_1^{sys} modifies a system $\sigma^\flat$ by substituting the names of the complexes belonging to that system with their population. The definition of γ_1^{sys} depends on α_1^{sys} and derives from a well-known result from the theory of abstract interpretation. Its meaning is clarified by the following lemma.

Lemma 3. *Let $\delta_1^{sys} : \mathsf{Sys}_1 \to \wp(\mathsf{Sys}_0)$ be a function defined as:*

$$\delta_1^{sys}(\sigma) = \{\sigma^\flat \in \mathsf{Sys}_0 \mid \forall c \in \mathsf{Cplx}.\ |\mathsf{names}(c, \sigma^\flat)| = \sigma(c)\}.$$

Then, $\gamma_1^{sys}(\Sigma) = \bigcup_{\sigma \in \Sigma} \delta_1^{sys}(\sigma)$.

In the following we perform a similar abstraction to the set of transitions Res_0 by removing all pieces of information regarding the names of the complexes that can react. At this point, it might be the case that more than one transition has the same reactants. We group all these transitions together and we assign them a rate obtained as sum of

the rates of each single transition. For instance, consider the transitions 5. and 6. from Example 5: $\langle\{\langle 0, c_2\rangle, \langle 0, c_3\rangle\}, 1\rangle, \langle\{\langle 0, c_2\rangle, \langle 1, c_3\rangle\}, 1\rangle \in \mathsf{Tr}_0$. By removing complexes' names, we have two transitions of the same form: $\langle\{c_2, c_3\}, 1\rangle$. The idea is then to substitute them with only one transition with rate $1+1=2$: $\langle\{c_2, c_3\}, 2\rangle \in \mathsf{Tr}_1$. This rate is called *propensity*. We formalize our idea: the map comp we define below substitutes the reactants of each transition by the set composed of their complexes, while the map prop calculates the propensities of new transitions in the way we hinted at above.

Definition 9. *We define maps* $\mathsf{comp} : \mathsf{Res}_0 \to \wp(\mathsf{Cplx} \otimes \mathsf{Cplx})$ *and* $\mathsf{prop} : (\mathsf{Res}_0 \times (\mathsf{Cplx} \otimes \mathsf{Cplx})) \to \mathbb{R}$ *as:*

$$\begin{aligned}\mathsf{comp}(R^\flat) &= \{\{c_1, c_2\} \mid \exists n_1, n_2 \in \mathbb{N}.\exists r \in \mathbb{R}^+.\, \langle\{\langle n_1, c_1\rangle, \langle n_2, c_2\rangle\}, r\rangle \in R^\flat\}\\ \mathsf{prop}(R^\flat, A) &= \textstyle\sum_{\langle B,r\rangle \in R^\flat \ s.t.\ \mathsf{comp}(\{\langle B,r\rangle\}) = A} r\end{aligned}$$

Example 7. Let $R^\flat$ be the set composed of the 9 transitions obtained in Example 5. Then, $\mathsf{comp}(R^\flat) = \{\{c_1\}, \{c_1, c_2\}, \{c_1, c_3\}, \{c_2, c_3\}\}$. On the other hand, the propensities are $\mathsf{prop}(R^\flat, \{c_1\}) = 1$, $\mathsf{prop}(R^\flat, \{c_1, c_2\}) = \mathsf{prop}(R^\flat, \{c_2, c_3\}) = 2$, $\mathsf{prop}(R^\flat, \{c_1, c_3\}) = 4$ and $\mathsf{prop}(R^\flat, A) = 0$ for all other $A \in \wp(\mathsf{Cplx} \otimes \mathsf{Cplx})$. ■

We can now define another pair of abstraction and concretization maps in order to relate Res_0 and Res_1.

Definition 10. *We define the abstraction map* $\alpha_1^{res} : \wp(\mathsf{Res}_0) \to \wp(\mathsf{Res}_1)$ *and the concretization map* $\gamma_1^{res} : \wp(\mathsf{Res}_1) \to \wp(\mathsf{Res}_0)$ *as:*

$$\begin{aligned}\alpha_1^{res}(\mathcal{R}^\flat) &= \{\{\left\langle A, \mathsf{prop}(R^\flat, A)\right\rangle \mid A \in \mathsf{comp}(R^\flat)\} \mid R^\flat \in \mathcal{R}^\flat\}\\ \gamma_1^{res}(\mathcal{R}) &= \bigcup\{\mathcal{R}^\flat \mid \alpha_1^{res}(\mathcal{R}^\flat) \subseteq \mathcal{R}\}.\end{aligned}$$

Lemma 4. $\left\langle\langle \wp(\mathsf{Res}_0), \subseteq\rangle, \alpha_1^{res}, \gamma_1^{res}, \langle \wp(\mathsf{Res}_1), \subseteq\rangle\right\rangle$ *is a Galois connection.*

We define a map which removes from a system $\sigma \in \mathsf{Sys}_1$ corresponding to a multiset of complexes (Definition 6) all those complexes not matching with a given pattern.

Definition 11. *Given a pattern* p *and a system* $\sigma \in \mathsf{Sys}_1$*, the evaluation of* p *in* σ *is a map* $[\![p]\!]^1 : \mathsf{Sys}_1 \to \mathsf{Sys}_1$ *defined as:*

$$[\![p]\!]^1\sigma = \lambda c.\ \begin{cases}\sigma(c) & \text{if } c \models p\\ 0 & \text{otherwise.}\end{cases}$$

Example 8. Consider the system σ defined in Example 6 and a pattern $[A]$. Since c_1 and c_2 match with p, while c_3 does not, we have $[\![[A]]\!]^1\sigma = [c_1 \mapsto 2, c_2 \mapsto 1, c_3 \mapsto 0]$. ■

Given a system $\sigma \in \mathsf{Sys}_1$, two complexes in it that may react, and the rate expression specified by an assoc rule, the following definition specifies how the propensity of all possible transitions induced by σ which have these complexes as reactants is computed.

Definition 12. *Let c_1 and c_2 be two complexes appearing in a system $\sigma \in \mathsf{Sys}_1$ and let e be an expression characterizing the rate of the reaction between these complexes. We determine the propensity of this reaction as follows:*

$$\mathsf{p}_\sigma(c_1, c_2, e) = \frac{\sigma(c) \cdot (\sigma(c)-1)}{2} \cdot [\![e]\!]\rho_1 \qquad \mathsf{p}_\sigma(c_1, c_2, e) = \sigma(c_1) \cdot \sigma(c_2) \cdot \frac{[\![e]\!]\rho_2 + [\![e]\!]\rho_3}{2},$$

where the first equation applies when $c_1 = c_2 = c$, otherwise the second applies; also, above we let $\rho_1 = [\mathtt{reactant}_1 \mapsto c, \mathtt{reactant}_2 \mapsto c]$, $\rho_2 = [\mathtt{reactant}_1 \mapsto c_1, \mathtt{reactant}_2 \mapsto c_2]$ and $\rho_3 = [\mathtt{reactant}_1 \mapsto c_2, \mathtt{reactant}_2 \mapsto c_1]$.

In the following we define $[\![p_1, p_2, e]\!]_1^{\mathsf{assoc}}$, the abstract semantics at level 1 of the actual semantics of the assoc rules (Definition 5).

Definition 13 (Level 1 semantics of assoc). *Consider a state $\sigma \in \mathsf{Sys}_1$ and a rule* assoc p_1 p_2 rate e react B. *We define the map $[\![p_1, p_2, e]\!]_1^{\mathsf{assoc-nc}} : \mathsf{Sys}_1 \to \mathsf{Res}_1$ as*

$$[\![p_1, p_2, e]\!]_1^{\mathsf{assoc-nc}}\sigma = \{\langle\{c_1, c_2\}, \mathsf{p}_\sigma(c_1, c_2, e)\rangle \in \mathsf{Tr}_1 \mid [\![p_1]\!]^1\sigma(c_1) \neq 0 \wedge [\![p_2]\!]^1\sigma(c_2) \neq 0\}.$$

The collecting semantics $[\![p_1, p_2, e]\!]_1^{\mathsf{assoc}} : \wp(\mathsf{Sys}_1) \to \wp(\mathsf{Res}_1)$ is defined as follows: $[\![p_1, p_2, e]\!]_1^{\mathsf{assoc}}\Sigma = \{[\![p_1, p_2, e]\!]_1^{\mathsf{assoc-nc}}\sigma \mid \sigma \in \Sigma\}$.

Example 9. Let us consider, one more time, rule_1 and rule_2 introduced in Example 5 and the system σ defined in Example 6. Let us determine the semantics at level 1 of these rules in σ. In Example 8 we showed that $[\![[A]]\!]^1\sigma = [c_1 \mapsto 2, c_2 \mapsto 1, c_3 \mapsto 0]$. We can, similarly show that $[\![[B]]\!]^1\sigma = [c_1 \mapsto 0, c_2 \mapsto 0, c_3 \mapsto 2]$. By Definition 12, we have:

$$\begin{array}{lll} \mathsf{p}_\sigma(c_1, c_3, 1) = 4 & \mathsf{p}_\sigma(c_2, c_3) = 2 & \\ \mathsf{p}_\sigma(c_1, c_2, 1) = 2 & \mathsf{p}_\sigma(c_1, c_1) = 1 & \mathsf{p}_\sigma(c_2, c_2) = 0. \end{array}$$

The following table represents the semantics at level 1 of our rules: rule_1 gives rise to transitions 1.-2., while rule_2 gives rise to transitions 3.-4.

rule_1	1.	$\langle\{c_1, c_3\}, 4\rangle$	2.	$\langle\{c_2, c_3\}, 2\rangle$
rule_2	3.	$\langle\{c_1\}, 1\rangle$	4.	$\langle\{c_1, c_2\}, 2\rangle$

It is worth noting that since $\mathsf{p}_\sigma(c_2, c_2) = 0$, transition $\langle\{c_2\}, \mathsf{p}_\sigma(c_2, c_2)\rangle$ does not belong to Tr_1, and therefore cannot be in $[\![[A], [A], e]\!]_1^{\mathsf{assoc-nc}}$. ■

The following lemma shows a relationship between the semantics at levels 0 and 1.

Lemma 5. *Given an expression e and two patterns p_1 and p_2, the following condition holds:*

$$[\![p_1, p_2, e]\!]_1^{\mathsf{assoc}} = \alpha_1^{res} \circ [\![p_1, p_2, e]\!]_0^{\mathsf{assoc}} \circ \gamma_1^{sys}.$$

4.2 Abstraction at Level 2

In this section we propose an additional abstraction of Res_1 which calculates the cumulative propensity of all the reactions the assoc rule gives rise to. Abstraction of systems is the same one we introduced in the previous subsection.

Definition 14. *A system $\sigma^\sharp$ is as a multiset of complexes, i.e., $\sigma^\sharp \in \mathsf{Sys}_2 = \mathsf{Sys}_1 = \mathsf{mset}\ \mathsf{Cplx}$. Moreover, we let $\mathsf{Res}_2 = \mathbb{R}$.*

We shall sometimes write σ instead of $\sigma^\#$ to stress the fact that $\mathsf{Sys}_2 = \mathsf{Sys}_1$. It is worth noting that both $\langle \wp(\mathsf{Res}_1), \subseteq \rangle$ and $\langle \wp(\mathsf{Res}_2), \subseteq \rangle$ are complete lattices. In the following we show that they are also related by a Galois connection [1].

Definition 15. *We define the abstraction map $\alpha_2^{res} : \wp(\mathsf{Res}_1) \to \wp(\mathsf{Res}_2)$ and the concretization map $\gamma_2^{res} : \wp(\mathsf{Res}_2) \to \wp(\mathsf{Res}_1)$ as:*

$$\alpha_2^{res}(\mathcal{R}) = \{\textstyle\sum_{\langle A,r\rangle \in R} r \mid R \in \mathcal{R}\} \qquad \gamma_2^{res}(\mathcal{R}^\sharp) = \bigcup\{\mathcal{R} \mid \alpha_2^{res}(\mathcal{R}) \subseteq \mathcal{R}^\sharp\}.$$

Lemma 6. $\left\langle \langle \wp(\mathsf{Res}_1), \subseteq \rangle, \alpha_2^{res}, \gamma_2^{res}, \langle \wp(\mathsf{Res}_2), \subseteq \rangle \right\rangle$ *is a Galois connection.*

We define $[\![p_1, p_2, e]\!]_2^{\mathsf{assoc}}$, the semantics of the assoc rules at level 2.

Definition 16 (Level 2 semantics of assoc). *Consider a state $\sigma \in \mathsf{Sys}_2$ and a rule* assoc p_1 p_2 rate e react *B. We define the map $[\![p_1, p_2, e]\!]_2^{\mathsf{assoc-nc}} : \mathsf{Sys}_2 \to \mathsf{Res}_2$ as*

$$[\![p_1, p_2, e]\!]_2^{\mathsf{assoc-nc}}\sigma = \sum_{\langle A,r\rangle \in [\![p_1,p_2,e]\!]_1^{\mathsf{assoc-nc}}\sigma} r.$$

The collecting semantics $[\![p_1, p_2, e]\!]_2^{\mathsf{assoc}} : \wp(\mathsf{Sys}_2) \to \wp(\mathsf{Res}_2)$ is defined as follows: $[\![p_1, p_2, e]\!]_2^{\mathsf{assoc}}\Sigma = \{[\![p_1, p_2, e]\!]_2^{\mathsf{assoc-nc}}\sigma \mid \sigma \in \Sigma\}$.

The semantics of an assoc rule at level 1 determines all possible transitions Res_1 the rule gives rise to, and for each of them its propensity is calculated as well. At level 2 the rule's semantics determines only the cumulative propensity of the transitions obtained at level 1.

Example 10. Consider the rules introduced in Example 5 and the system σ defined in Example 6. Let us determine the semantics at level 2 of these rules in σ. In Example 9 we showed that the transitions obtained from rule_1 are $\langle\{c_1, c_3\}, 4\rangle$ and $\langle\{c_2, c_3\}, 2\rangle$, so their cumulative propensity is $4 + 2 = 6$. On the other hand, rule_2 gave transitions $\langle\{c_1\}, 1\rangle$ and $\langle\{c_1, c_2\}, 2\rangle$ and their cumulative propensity is $1+2 = 3$. Thus, the semantics at level 2 of rule_1 and rule_2 are 6 and 3 respectively.

Although this is a quite simple example, we can notice that there is an actual reduction of numbers of transitions appearing in different semantics of these two rules: rule_1 has 6 transitions Res_0 at level 0, 2 transitions Res_1 at level 1 and 1 transition Res_2 at level 2, while rule_2 has 3 transitions Res_0 at level 0, 2 transitions Res_1 at level 1 and 1 transition Res_2 at level 2. ■

In order to compute the semantics of an assoc rule at level 2, we should first determine the semantics of that rule at level 1 (Definition 16). Therefore, the complexity of the computation of the semantics at level 2 appears to be higher then the one of semantics at level 1. In some cases, however, we can compute the semantics at level 2 without computing the one at level 1, as shown in the following lemma. The proofs relies on well-known combinatorial properties.

Lemma 7. *Let e be an expression with no occurrence of* $\texttt{reactant}_1$ *and* $\texttt{reactant}_2$*, $\sigma \in \mathsf{Sys}_2$ be a system. Then, the following equation holds:*

$$[\![p_1, p_2, e]\!]_2^{\mathsf{assoc}}\sigma = [\![e]\!]\rho \cdot \left(|[\![p_1]\!]^1\sigma| \cdot |[\![p_2]\!]^1\sigma| - \binom{|[\![p_1]\!]^1\sigma \cap [\![p_2]\!]^1\sigma|}{2}\right) - \left(\binom{|[\![p_1]\!]^1\sigma \cap [\![p_2]\!]^1\sigma|}{1}\right).$$

Example 11. Let us consider one more time the system σ from Example 6 and the rule_2 from Example 5. We compute rule_2's semantics at level 2 using Lemma 7. We showed in Example 8 that $[\![[A]]\!]^1\sigma = [c_1 \mapsto 2, c_2 \mapsto 1, c_3 \mapsto 0]$, and therefore $|[\![[A]]\!]^1\sigma| = 2 + 1 + 0 = 3$. We have $[\![[A], [A], 1]\!]_2^{\mathsf{assoc}}\sigma = 3 \cdot 3 - \binom{3}{2} - \binom{3}{1} = 3 \cdot 3 - \frac{3 \cdot 2}{2} - 3 = 3$, which is equal to the semantics of rule_2 computed by Definition 16 in Example 10. ■

5 Simulation

In this section we discuss how to simulate biological models expressed in our rule-based language. In doing that, we shall discuss how the abstractions provided in Sect. 4 can be exploited so to build optimized algorithms. In order to keep our presentation concise, we shall pretend that the model at hand is composed by assoc rules, only. Other kind of rules (dissoc, dyn) can indeed be handled through the same techniques.

5.1 Level 0 Simulation

We start by considering the problem of simulating a level 0 system, described via an initial state $\sigma^\flat \in Sys_0$ and a set of rules. A straightforward way to represent $\sigma^\flat$ is that of storing the information relative to each complex $\langle n, c\rangle \in \sigma^\flat$ in its own memory area. That is, we allocate an "object" for every single complex in $\sigma^\flat$ in which we store n and (a representation of) the multiset of the boxes in c, each one with its own state variables.

Simulating the system then can be done as follows. First, for each assoc rule, say indexed by $k \in K$, we compute $[\![[p_{k,1}], [p_{k,2}], e]\!]_0^{assoc-nc}$ by enumerating all the annotated complex pairs matching with the patterns. This provides us with sets of level 0 transitions $\{\langle A_{k,j}, a_{k,j}\rangle \mid j \in J\}$ (for some index set J), where each $A_{k,j}$ mentions exactly two annotated complexes. We assign to each transition the probability obtained by normalizing the rates (i.e., $a_{k,j}/\sum_{k,j} a_{k,j}$), and then randomly choose one of them, say $\langle A_{\nu,\mu}, a_{\nu,\mu}\rangle$. Simulation time is advanced by a random amount, generated according to the exponential distribution $Exp(\sum_{k,j} a_{k,j})$. The two annotated complexes in $A_{\nu,\mu}$ are removed from $\sigma^\flat$, then associated by merging their multisets of boxes (say c_1 and c_2). Finally we create a new annotated complex $\langle n', c_1 \cup c_2\rangle$ for some fresh n', and insert it in $\sigma^\flat$. At this point, the react code block of the rule is run (possibly modifying the newly created complex via its product variable, and spawning new complexes as well). The whole procedure is then repeated.

Below, we provide pseudo-code for the whole simulation procedure. We let index sets K, J to start counting from 1. Summing over the multi-index $\langle k, j\rangle \in K \times J$ follows the lexicographic ordering.

Level 0 simulation algorithm

1: {assoc $[p_{k,1}][p_{k,2}]$ rate e_k react $c_k \mid k \in K$} := the set of rules of the model
2: $\sigma^\flat$:= the initial state ; simulation time $t := 0$

3: **while** $t < t_{\max}$ **do**
4: **for all** rule indexes $k \in K$ **do**
5: compute the level 0 transitions $\{\langle A_{k,j}, a_{k,j}\rangle | j \in J\} := [\![[p_{k,1}], [p_{k,2}], e_k]\!]_0^{assoc-nc}\sigma^\flat$
6: **end for**
7: compute $a_0 := \sum_{k,j} a_{k,j}$; generate a random number $u := \mathcal{U}(0, a_0)$
8: find the minimum rule-transition index $\langle \nu, \mu\rangle$ such that $u \leq \sum_{\langle k,j\rangle=\langle 1,1\rangle}^{\langle \nu,\mu\rangle} a_{k,j}$
9: $t := t + Exp(a_0)$; apply reaction μ by associating complexes in $A_{\nu,\mu}$, running command c_ν, and updating state $\sigma^\flat$ accordingly
10: **end while**

The above simple algorithm is faithful to the continuous-time Markov chain which define the stochastic behaviour of biochemical systems. However, its space and time performance makes it unpractical for real-world applications. Indeed, one can note that it allocates a rather large amount of memory. This is because biochemical systems can involve a large number of complexes, hence allocating memory for each one of them should be avoided. Fortunately, in typical models it is often the case that many distinct complexes actually have identical state, so one can actually reduce the memory footprint by aggressively sharing the state data (and using copy-on-write to preserve the semantics). Even with this optimization, time performance suffers by the explicit enumeration of all possible reacting pairs. To overcome this problem, we abstract the system to level 1.

5.2 Level 1 Simulation

Lemma 1 states that, since an expression e can not access to the n component of a complex $\langle n, c\rangle$, evaluating e for distinct complexes sharing the same state yields the same value. Because of this, one can then evaluate it only once, and multiply the result by the number of complex pairs, hence obtaining the cumulative rate for all such transitions. In order to do that, we do not need to actually enumerate all the complex pairs, but just to compute their number, which can easily be done exploiting combinatorial formulae while keeping track of the amount of complexes in each state, i.e. counting the population for each species. This greatly improves the time performance of the simulation algorithm.

Since we now need only a population count, we can avoid to store the names n for each complex, so to furher improve the memory footprint. This essentially amount to move to the level 1 abstraction, i.e. turning $\sigma^\flat \in Sys_0$ into a $\sigma \in Sys_1$. Simulating at that level results in a less detailed simulation output, which describes which species interact without specifying the actual identities of the involved complexes. Since identities are unimportant from a biological point of view, and quantities are, the result of simulation still preserves all the relevant information of level 0.

Below, we adapt the level 0 simulation algorithm so to work at level 1. This actually results in the well-known Gillespie's Direct Method [4], which indeed works precisely at that abstraction level. A minor difference worth mentioning is that in our models an unbounded number of new chemical species (i.e., complexes) can be formed during simulation, while in the reaction-based models considered by Gillespie the set of such species is finite and statically known before simulation is started.

Level 1 simulation algorithm (Gillespie's Direct Method)

1: {assoc $[p_{k,1}][p_{k,2}]$ rate e_k react $c_k midk \in K$} := the set of rules of the model
2: σ := the initial state ; simulation time $t := 0$
3: **while** $t < t_{\max}$ **do**
4: **for all** rule indexes $k \in K$ **do**
5: compute the level 1 transitions $\{\langle A_{k,j}, a_{k,j}\rangle | j \in J\} := [\![[p_{k,1}], [p_{k,2}], e_k]\!]_1^{assoc-nc}\sigma$ exploiting Def. 12 to compute propensities $a_{k,j}$
6: **end for**
7: compute $a_0 := \sum_{k,j} a_{k,j}$; generate a random number $u := \mathcal{U}(0, a_0)$
8: find the minimum rule-transition index $\langle \nu, \mu\rangle$ such that $u \leq \sum_{\langle k,j\rangle=\langle 1,1\rangle}^{\langle \nu,\mu\rangle} a_{k,j}$
9: $t := t + Exp(a_0)$; apply reaction μ by associating complexes in $A_{\nu,\mu}$, running command c_ν, and updating state σ accordingly
10: **end while**

Comparing the above to the level 0 algorithm, we find the main, important difference in line 5, where we exploit the combinatorial formula in Def. 12 to compute propensities.

Several further standard optimizations can be applied, e.g., after a transition has been applied, we can avoid to recompute those propensities which are known to be unaffected by that transition. This is typically done via a dependency graph [3].

5.3 Level 2 Simulation

We saw how abstracting the model from level 0 to level 1 improves the space and time performance of the simulation. We now investigate the consequences of further abstracting the model to level 2. As we shall see, under some assumptions, this may lead to further improvements in time performance.

Recall that level 2 systems $\sigma^{\#} \in Sys_2$ are actually identical to level 1 systems $\sigma \in Sys_1$, so no information about the complexes is actually lost here. Instead, level 2 abstract the transitions which are being generated by the semantics. More in detail, we have that *all* the level 1 transitions ($\in Res_1$) which are being generated by any given rule are collapsed into a single level 2 transition ($\in Res_2$) having as its rate the sum of all the rates of level 1 transitions. In this way, the propensities of level 1 transitions are combined to form a single *cumulative propensity* for the rule at hand. Also, in the common case in which the rate expression of such rule is simply a constant, the cumulative propensity for the rule can be computed efficiently exploiting Lemma 7. This suggests a possible modification to the level 1 simulation algorithm (Direct Method) which we provide below.

Level 2 simulation algorithm

1: {assoc $[p_{k,1}][p_{k,2}]$ rate e_k react $c_k \mid k \in K$} := the set of rules of the model
2: $\sigma^{\#}$:= the initial state ; simulation time $t := 0$
3: **while** $t < t_{\max}$ **do**
4: **for all** rule indexes $k \in K$ **do**
5: compute the level 2 transition $r_k := [\![[p_{k,1}], [p_{k,2}], e_k]\!]_2^{assoc-nc}\sigma^{\#}$ exploiting the formula in Lemma 7
6: **end for**

7: compute $r_0 := \sum_{k \in K} r_k$ and generate a random number $u := \mathcal{U}(0, r_0)$
8: find the minimum rule-index ν such that $u \leq \sum_{k=1}^{\nu} r_k$; let $u := u - \sum_{k=1}^{\nu-1} r_k$
9: compute the level 1 transitions $\{\langle A_j, a_j \rangle \mid j \in J\} := [\![[p_{\nu,1}], [p_{\nu,1}], e_\nu]\!]_1^{assoc-nc} \sigma^{\#}$ exploiting Def. 12 to compute propensities a_j
10: find the minimum transition index μ such that $u \leq \sum_{j=1}^{\mu} a_j$
11: $t := t + Exp(r_0)$; apply reaction μ by associating complexes in A_μ, running command c_ν and updating state $\sigma^{\#}$ accordingly
12: **end while**

The main change with respect to level 1 can be summarized as follows. In level 1 simulation, we generate all the level 1 transitions and then randomly pick among them. In level 2 simulation, we only generate *one level 2 transition per rule* from which we randomly pick one. After such choice is done, we know the rule ν which is to be applied: we then generate level 1 transitions for that rule *only*, and then pick among those. The net result is that we perform two random choices among two small sets instead of one choice in a large set. Assume for the sake of illustration that a model features 20 association rules, and that each pattern in them matches with 5 complexes. Generating all the level 1 transitions requires enumerating all the $\simeq 20 \cdot 5 \cdot 5 = 500$ cases. Instead, performing two separate choices for level 2 and level 1 transition requires enumerating only $\simeq 20 + 5 \cdot 5 = 45$ cases.

In order to exploit the observation above, it is important to exploit Lemma 7 to compute level 2 transitions efficiently. For that, we need to quickly compute, for each association rule, the quantities $|[\![p_1]\!]^1\sigma|$, $|[\![p_2]\!]^1\sigma|$, and $|[\![p_1]\!]^1\sigma \cap [\![p_2]\!]^1\sigma|$. This can be done by keeping track for each rule of three sets of complexes: those matching with p_1, p_2, and both. These sets need to be updated infrequently: updates are needed only when a complex c is added to a system σ for which $\sigma(c) = 0$, i.e., when the first copy of c appears in the system. Having these three sets, computing the wanted cardinalities is done by summing all the populations of the complexes. An incremental approach which adjusts such quantities at every step – without recomputing them – is also feasible.

Steps number 9 and 10 can be further optimized. There, we find the set of complexes A_μ to be associated by generating all the level 1 transitions for rule ν, and then using their rates as weights for the random choice of A_μ. This might be improved by using an alternative way to extract A_μ from the same distribution, so to avoid the expensive enumeration of all the level 1 transitions. One such way is as follows. Randomly extract a complex c_1 from those matching with pattern $p_{\nu,1}$, using their population count as weights. Complex c_2 can be chosen similarly using $p_{\nu,2}$. Here, however, in the case c_1 also matches with $p_{\nu,2}$, we decrement the population of c_1 by one unit. This adjustment reflects the fact that association must involve two *distinct* annotated complexes. This alternative way of extracting $A_\mu = \{c_1, c_2\}$ is advantageous since it requires at most a linear scan of all the complexes matching p_1 and p_2. By contrast, enumerating the level 1 transitions can generate a quadratic number of them.

6 Conclusions

We introduced $\mathcal{L}$, a rule-based imperative language for the modeling and simulation of biochemical systems. We provided a concrete semantics for it, as well as two abstract

ones. We then exploited the abstractions so to devise efficient simulation algorithms for $\mathcal{L}$. Future work will investigate extensions of $\mathcal{L}$ to more specific kinds of models, e.g. those involving compartments or other formalisms to represent space.

References

1. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proceedings of the 4th POPL, pp. 238–252. ACM (1977)
2. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-Based Modelling of Cellular Signalling. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 17–41. Springer, Heidelberg (2007)
3. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. Journal of Physical Chemistry A 104(9), 1876–1889 (2000)
4. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
5. Priami, C., Quaglia, P., Romanel, A.: BlenX Static and Dynamic Semantics. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 37–52. Springer, Heidelberg (2009)
6. Priami, C., Quaglia, P., Zunino, R.: An imperative language of self-modifying graphs for biological systems. In: ACM Symposium on Applied Computing (SAC) (to appear, 2012)
7. Romanel, A., Priami, C.: On the decidability and complexity of the structural congruence for beta-binders. Theor. Comput. Sci. 404(1-2), 156–169 (2008)