Efficient Syntax-driven Lumping of Differential Equations

Luca Cardelli[1], Mirco Tribastone[2], Max Tschaikowski[2], and Andrea Vandin[2]

[1] Microsoft Research & University of Oxford, UK
[2] IMT Institute for Advanced Studies Lucca, Italy

Abstract. We present an algorithm to compute exact aggregations of a class of systems of ordinary differential equations (ODEs). Our approach consists in an extension of Paige and Tarjan's seminal solution to the coarsest refinement problem by encoding an ODE system into a suitable discrete-state representation. In particular, we consider a simple extension of the syntax of elementary chemical reaction networks because i) it can express ODEs with derivatives given by polynomials of degree at most two, which are relevant in many applications in natural sciences and engineering; and ii) we can build on two recently introduced bisimulations, which yield two complementary notions of ODE lumping. Our algorithm computes the largest bisimulations in $O(r \cdot s \cdot \log s)$ time, where r is the number of monomials and s is the number of variables in the ODEs. Numerical experiments on real-world models from biochemistry, electrical engineering, and structural mechanics show that our prototype is able to handle ODEs with millions of variables and monomials, providing significant model reductions.

1 Introduction

Ordinary differential equations (ODEs) are widespread in many disciplines including chemistry, epidemiology, systems biology, electrical engineering, and control theory. Often, due to the complexity of the system under consideration, the state space size (intended as the number of ODE variables) is so large that it makes the numerical solution intractable (e.g., in protein-based interaction networks [11,4]). Formal kinds of analyses such as reachability computation suffer from the curse of dimensionality, particularly for nonlinear systems (e.g., [10,29]). It is therefore an important goal to be able to obtain reduced size models that appropriately preserve the original dynamics.

For discrete-state quantitative models based on labeled transition systems, the notion of bisimilarity has played a key role for model reduction, with efficient algorithms [1,13,34] based on Paige and Tarjan's celebrated solution to the coarsest refinement problem [27]. The main contribution of this paper is to lift this approach to ODE systems. In particular we focus on a class of *polynomial systems*, where the time derivatives are multivariate polynomials of degree at most two in the ODE variables. This class is quite general because it incorporates models frequently used in (bio-)chemistry (cf. [26]) as well as the ubiquitous linear ODEs.

We reconcile the established approaches based on discrete-state models with the continuous-state semantics of ODEs by reasoning at the level of a discrete-state *syntactic* representation of the ODE system. In particular, our class of interest can be encoded into

a variant of elementary chemical reaction networks (CRNs). This consists of species (the ODE variables) interacting through unary or binary reactions that are appropriately mapped onto monomials that govern the derivative of the species involved. To be able to encode an arbitrary polynomial ODE system (with degree at most two), we slightly extend the CRN syntax by allowing negative rates. This has important repercussions on the applicability of established results of CRN theory (e.g., [15]). Hence, to disambiguate, we refer to this extension as Reaction Networks (RN). Instead, all the results for exact quantitative bisimulations for CRNs, recently proposed by these authors in [7] (cf. Section 2), do carry over to RN. The *forward* bisimulation (FB) gives a partition of the ODE variables such that the sum of the ODEs can be written as an explicit function of the sum of the variables. With *backward* bisimulation (BB) species in the same block have the same ODE solution, provided that they start with the same initial condition.

Our key idea is to exploit the fact that the *syntactic* conditions for an equivalence relation over species to be either bisimulation can be expressed in the Larsen-Skou style of probabilistic bisimulation [23]. (Actually, while this is immediate for FB, in this paper we provide a novel characterization of BB tailored to that format, cf. Section 3.1.) Thus, we can approach the problem of computing the largest bisimulations by developing a variant of Paige and Tarjan's algorithm, along the lines of the efficient partition refinement algorithms of [13] and [34] for Markov chain lumping [5], and of [1] for probabilistic transition systems. In particular, for technical reasons that will be clarified later, we build on the Markov chain lumping algorithm of [34].

Our algorithm, presented in Sections 3 and 4, runs in $O(r \cdot s \cdot \log s)$ time, where s is the number of variables and r is the number of monomials in the ODEs. Interestingly, this can be related to continuous-time Markov chain (CTMC) lumping. The time complexity of our algorithm is a tight increase, in the following sense: Since RNs can encode arbitrary affine ODEs, *a fortiori* they can encode a CTMC through its Kolmogorov equations (cf. Section 5). For this encoding we show that FB and BB correspond to the well-known notions of ordinary and exact lumpability for CTMCs, respectively. In the affine ODE case, the time complexity of our algorithm collapses to $\mathcal{O}(r \cdot \log s)$, which is equivalent to that of the most efficient CTMC lumpability algorithms [13,34].

We show the practical usefulness of our algorithm by means of numerical experiments (in Section 6), with a prototype available at http://sysma.imtlucca.it/crnreducer/. Using the benchmark biochemical models of [7], we measure runtime speed-ups of up to four orders of magnitude over our own more straightforward $\mathcal{O}(r^2 \cdot s^5)$ algorithm used in [7]. We are now able to reduce the largest benchmark biochemical model within a few seconds on commodity hardware, as opposed to almost one day as reported in [7]. To evaluate the effectiveness on affine systems, we propose an application of the bisimulations beyond ODEs: we consider linear systems of equations $Ax = b$. Stationary iterative methods such as Jacobi's (e.g., [28]) can be interpreted as an affine dynamical system in *discrete time*, to which case the bisimulations carry over. For these, we report considerable aggregations for real-world applications in atmospheric modeling, structural mechanics, and electrical engineering, taken from the Sparse Matrix Collection [12].

Further related work. FB is a special case of the theory of ODE lumping [24,32], which is more general because it considers an arbitrary linear transformation of the state space, as opposed to a sum of variables for FB. While the theory is established,

no algorithm is available to compute such aggregations. BB is a generalization of a behavioral equivalence originally defined for Markovian process algebra [33]. FB and BB have been recently put in a unifying algorithmic context in [8], using the notions of *forward* and *backward differential equivalences* for a low-level syntax describing a more general class of nonlinear ODE systems. A symbolic partition-refinement algorithm to compute the largest differential equivalences is provided through a satisfiability modulo theories encoding. Clearly, unlike this approach, the algorithm of [8] is independent of the restriction of the RN language, and is not a variant of Paige and Tarjan's approach. As a result, it is more general but less efficient. Indeed, the runtimes reported in [8] are at best only comparable to those of our earlier algorithm [7].

This is the first application of Paige and Tarjan's seminal idea for a general class of ODE systems, whereas automatic exact ODE reduction algorithms are available for domain specific languages such as rule-based models of biochemical networks [11] and Markovian process algebra using FB-like (though not BB-like) conditions [18].

2 Background

Reaction Networks. An RN (S, R) is a pair of a finite set of *species* S and a finite set of *reactions* R. A reaction is a triple written in the form $\rho \xrightarrow{\alpha} \pi$, where ρ and π are multisets of species, called *reactants* and *products*, respectively, and $\alpha \neq 0$ is the *reaction rate*. We restrict to *elementary* reactions where $|\rho| \leq 2$ (while no restriction is posed on the products). We denote by $\rho(X)$ the multiplicity of species X in the multiset ρ, and by $\mathcal{MS}(S)$ the set of finite multisets of species in S. The operator $+$ denotes multiset union, e.g., $X + Y + Y$ (or just $X + 2Y$) is the multiset $\{\!|X, Y, Y|\!\}$. We also use X to denote either the species X or the singleton $\{\!|X|\!\}$.

The *semantics* of an RN (S, R) is given by the (autonomous) ODE system $\dot{V} = F(V)$, with $F : \mathbb{R}^S \to \mathbb{R}^S$, where each component F_X, with $X \in S$ is defined as:

$$F_X(V) := \sum_{\rho \xrightarrow{\alpha} \pi \in R} (\pi(X) - \rho(X)) \cdot \alpha \cdot \prod_{Y \in S} V_Y^{\rho(Y)} .$$

This ODE satisfies a unique solution $V(t) = (V_X(t))_{X \in S}$ for any initial condition $V(0)$.

The restriction to elementary reactions ensures that the monomials are of degree at most 2; unary reactions give degree-one monomials; a *nullary* reaction, $\emptyset \xrightarrow{c} X$, adds a constant c to $F_X(V)$. (The encoding of an arbitrary polynomial ODE system is shown in Section 5.) Finally, we remark that a standard CRN with mass-action semantics (where reactions speeds are proportional to the product of the concentrations of the reactants) is recovered by restricting to positive reaction rates and nonnegative initial conditions.

Example 1. We now provide a simple RN, (S_e, R_e), with $S_e = \{A, B, C, D, E\}$ and $R_e = \{A + C \xrightarrow{\alpha} C + E, B + C \xrightarrow{\alpha} C + E, C \xrightarrow{\beta} A, D \xrightarrow{\beta} B\}$, which will be used as a running example in this section. Its ODE system is

$$\dot{V}_A = -\alpha V_A V_C + \beta V_C \qquad \dot{V}_C = -\beta V_C \qquad \dot{V}_E = \alpha V_A V_C + \alpha V_B V_C$$
$$\dot{V}_B = -\alpha V_B V_C + \beta V_D \qquad \dot{V}_D = -\beta V_D$$

We now overview the main definitions of [7], restating them in terms of an RN.

Forward Bisimulation. FB induces a partition associating an ODE with each block, representing the sum of the species in that block. It is defined in terms of *reaction* and *production* rates.

Definition 1 (Reaction and Production rates). *Let (S, R) be an RN, $X, Y \in S$, and $\rho \in S \cup \{\emptyset\}$. The* ρ-reaction rate *of X, and the* ρ-production rate *of Y-elements by X are defined respectively as*

$$\mathbf{crr}[X, \rho] := (\rho(X) + 1) \sum_{X+\rho \xrightarrow{\alpha} \pi \in R} \alpha, \quad \mathbf{pr}(X, Y, \rho) := (\rho(X) + 1) \sum_{X+\rho \xrightarrow{\alpha} \pi \in R} \alpha \cdot \pi(Y)$$

Finally, for $H \subseteq S$ we define $\mathbf{pr}[X, H, \rho] := \sum_{Y \in H} \mathbf{pr}(X, Y, \rho)$.

Definition 2. *Let (S, R) be an RN, $\mathcal{R}$ an equivalence relation over S and $\mathcal{H} = S/\mathcal{R}$. Then, $\mathcal{R}$ is a forward RN bisimulation (FB) if for all $(X, Y) \in \mathcal{R}$, all $\rho \in S \cup \{\emptyset\}$, and all $H \in \mathcal{H}$ it holds that*

$$\mathbf{crr}[X, \rho] = \mathbf{crr}[Y, \rho] \quad \textit{and} \quad \mathbf{pr}[X, H, \rho] = \mathbf{pr}[Y, H, \rho] \tag{1}$$

For instance, it can be shown that $\mathcal{H}_F = \{\{A, B\}, \{C\}, \{D\}, \{E\}\}$ for Example 1 is an FB. Indeed, the ODEs can be reduced by writing them in terms of $V_{AB} := V_A + V_B$:

$$\dot{V}_{AB} = -\alpha V_{AB} V_C + \beta V_C + \beta V_D \quad \dot{V}_C = -\beta V_C \quad \dot{V}_D = -\beta V_D \quad \dot{V}_E = \alpha V_{AB} V_C$$

Backward Bisimulation. BB leads to partitions where species in the same block have the same solution when starting with the same initial condition. It is defined according to the notion of *flux rates*.

Definition 3 (Cumulative flux rate). *Let (S, R) be an RN, $X \in S$, $\rho \in \mathcal{MS}(S)$, and $\mathcal{M} \subseteq \mathcal{MS}(S)$. Then, we define*

$$\mathbf{fr}(X, \rho) := \sum_{\rho \xrightarrow{\alpha} \pi \in R} (\pi(X) - \rho(X)) \cdot \alpha, \qquad \mathbf{fr}[X, \mathcal{M}] := \sum_{\rho \in \mathcal{M}} \mathbf{fr}(X, \rho).$$

We call $\mathbf{fr}(X, \rho)$ and $\mathbf{fr}[X, \mathcal{M}]$ ρ-flux rate *and* cumulative $\mathcal{M}$-flux rate *of X, respectively.*

Definition 4. *Let (S, R) be an RN, $\mathcal{R}$ an equivalence relation over S, and $\mathcal{H} = S/\mathcal{R}$. Then, $\mathcal{R}$ is a backward RN bisimulation (BB) if for any $(X, Y) \in \mathcal{R}$ it holds that*

$$\mathbf{fr}[X, \mathcal{M}] = \mathbf{fr}[Y, \mathcal{M}] \ \textit{for all} \ \ \mathcal{M} \in \{\rho \mid \rho \xrightarrow{\alpha} \pi \in R\}/\approx_{\mathcal{H}},$$

where any two $\rho, \sigma \in \mathcal{MS}(S)$ satisfy $\rho \approx_{\mathcal{H}} \sigma$ when $\sum_{Y \in H} \rho(Y) = \sum_{Y \in H} \sigma(Y)$ for all $H \in \mathcal{H}$.

It can be shown that $\mathcal{H}_B = \{\{A, B\}, \{C, D\}, \{E\}\}$ is a BB for the running example. Indeed, it is easy to see that $V_A(t) = V_B(t)$ and $V_C(t) = V_D(t)$ at all time points $t \geq 0$ whenever $V_A(0) = V_B(0)$ and $V_C(0) = V_D(0)$. So, one can remove the ODEs of $\dot{V}_B, \dot{V}_D$ and replace each V_B with V_A and each V_D by V_C, yielding the reduced ODE:

$$\dot{V}_A = -\alpha V_A V_C + \beta V_C \qquad \dot{V}_C = -\beta V_C \qquad \dot{V}_E = 2\alpha V_A V_C$$

In [7] it is discussed how to additionally obtain a reduced network up to a bisimulation $\mathcal{H}$, having one species per block of $\mathcal{H}$. For example, it can be shown that $\mathcal{H}_F$ induces the FB-reduced RN $S_e^F = \{A, C, D, E\}$ and $R_e^F = \{A + C \xrightarrow{\alpha} C + E, C \xrightarrow{\beta} A, D \xrightarrow{\beta} A\}$.

3 Computing the Coarsest RN Bisimulations

As introduced in Section 1, we exploit the fact that the conditions for FB and BB are in the Larsen-Skou style of probabilistic bisimulation, whereby, roughly speaking, two states are equivalent if their behavior toward any equivalence class is the same.

For FB, the notion of $\mathbf{pr}[X, H, \rho]$ in Definition 2 is already in such desired format: X is the species for which the equivalence is being checked, H is an equivalence class of "target" states, while ρ plays the role of a "label", identifying partner species reacting with X (akin to an action type in a probabilistic transition system). This is the intuitive correspondence that suggests us to employ a partition refinement approach based on Paige and Tarjan's algorithm, iteratively refining an input partition based on a *splitter* block that tells apart the behavior of two species toward that block, for some label ρ. One fundamental aspect of such an approach is that, at each iteration, the blocks of the current partition are used as potential splitters. This ensures that the list of splitters can be updated at essentially no additional cost while splitting the blocks.

For BB, instead, the situation is more delicate because the equivalence condition is based on the flux rate $\mathbf{fr}[X, \mathcal{M}]$. Unlike FB, here $\mathcal{M}$ does not represent an equivalence class of the species, but it is an equivalence class of *multi-sets* of species (all the possible reagents in the RN), which are equal up to $\approx_{\mathcal{H}}$, i.e., the equivalence induced by the current partition $\mathcal{H}$. Within this setting Paige and Tarjan's approach cannot be used directly because the splitters are not the partition blocks of the equivalence relation of interest. Thus, we first provide an alternative characterization of BB which allows to use (species) partition blocks as splitters. Then, we discuss a parameterized algorithm that can compute the coarsest refinement of a given partition of species up to FB or BB.

3.1 Splitter-based Characterization of Backward Bisimulation

The alternative characterization of BB is based on the following.

Definition 5 (Cumulative splitter flux rate). *Let (S, R) be an RN, $X, Y \in S$, $\mathcal{H}$ a partition of S, $H \in \mathcal{H}$ and $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$. We define*

$$\mathbf{sr}(X, Y, H') := \sum_{\rho' \in H'} \sum_{\substack{\rho \xrightarrow{\alpha} \pi \in R \\ \rho = Y + \rho'}} (\pi(X) - \rho(X)) \cdot \alpha', \quad \mathbf{sr}[X, H, H'] := \sum_{Y \in H} \mathbf{sr}(X, Y, H').$$

with $\alpha' = \frac{\alpha}{2}$ if $Y \neq \rho'$ and $Y \in H'$, or $\alpha' = \alpha$ otherwise. We call the quantity $\mathbf{sr}[X, H, H']$ the cumulative (H, H')-splitter flux rate *of X.*

Note that we account for summands that are counted twice due to the summation over H and H' in $\mathbf{sr}[X, H, H']$ by choosing $\alpha' \in \{\alpha, \frac{\alpha}{2}\}$ in the above definition.

Theorem 1. *Let (S, R) be an RN, $\mathcal{R}$ an equivalence relation over S and $\mathcal{H} = S/\mathcal{R}$. Then $\mathcal{R}$ is a BB if and only if for all $(X, Y) \in \mathcal{R}$, all $H \in \mathcal{H}$ and all $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$ it holds that $\mathbf{sr}[X, H, H'] = \mathbf{sr}[Y, H, H']$.* [3]

[3] All proofs are given in a technical report available at http://sysma.imtlucca.it/crnreducer/

With this characterization both $\mathbf{pr}$ and $\mathbf{sr}$ have three arguments, with analogous meaning, as discussed. In particular, the third argument of $\mathbf{sr}$ can now be also interpreted as a *label*. However, while in FB this ranges over the set of species (together with the distinguished species $\emptyset$ to indicate unary reactions), in BB it ranges over blocks of the candidate BB partition to be checked (again, together with the distinguished set $\{\emptyset\}$ for unary reactions). When used within the partition refinement algorithm, splitting a partition block leads to a refinement of the BB labels. In other words, unlike for FB the set of labels must be updated at every iteration. However, differently from the original definition of $\mathbf{fr}$, this only requires splitting a block rather than computing an equivalence relation over the species multi-sets appearing as reaction products. As we will see, this can be done at no additional cost.

Remark 1. The analogy with the probabilistic-bisimulation condition (where a label corresponds to an action type and the rates correspond to probabilities) may suggest to use a variant of the algorithm for probabilistic bisimilarity developed in [1]. Indeed, by suitably encoding an RN into a hyper-graph, the largest FB can be computed with [1]. However, a similar algorithm cannot be straightforwardly adapted to BB because the set of labels changes at every iteration. In particular, the bounds of Lemma 4.5 in [1] would not carry over if the labels were not kept fixed. For this reason, in this paper we consider an extension of the more recent [34], which also has the advantage of a simpler implementation because it does not require the intertwining between two classes of splitters like [1], or splay trees like [13].

3.2 Data Structures

We introduce the data structures used in our algorithm for computing the coarsest RN bisimulations. To achieve tight time and space bounds, we make use of pointer-based data structures only. Furthermore, we assume that species, partition blocks and reactions are stored once and then referred by other data structures via pointers.

Notation. Fix an RN (S, R), set $s := |S|$, $r := |R|$ and let $\mathcal{L}(R) := \{X \mid \exists X + Y \xrightarrow{\alpha} \pi \in R\} \cup \{\emptyset\}$ be the set of all *labels*. Set $l := |\mathcal{L}(R)|$ which can be bounded by $\mathcal{O}(\min(s, r))$. Finally, use $p := \max\{\sum_{X \in S} \mathbb{1}_{\{\pi(X)>0\}} \mid \rho \xrightarrow{\alpha} \pi \in R\}$ to denote the maximum number of different species which appear as products of a reaction. We will also use the fact that s is bounded by $(2 + p) \cdot r$. This is because each reaction can have at most 2 and p different species as reagents and products, respectively.[4]

We remark that, in general, p is bounded by s. However, we prefer to explicitly use this parameter because in the main application of this paper, i.e., the encoding of an arbitrary polynomial ODE system, p becomes a constant (i.e., 3). Instead, when an RN is used *directly* as the input specification to describe a model, as is the case in CRNs, p is typically small. For instance, in most reactions of biological processes the number of distinct products is typically one (e.g., for binding and internal state modification) or two (for unbinding or catalytic reactions). Indeed, across all the benchmark CRNs considered in Section 6, p never exceeds 3. This is due, for instance, to unbinding reactions favored by a catalyst, in the form $AB + C \to A + B + C$.

[4] We implicitly disregard pathological cases with species not appearing in any reaction.

RN representation. Species are stored in a list. We assume that the set $\mathcal{L}(R)$ is given and stored as a list of pointers to species (plus one entry for $\emptyset$), requiring $\mathcal{O}(l)$ space. However, its computation requires $\mathcal{O}(r)$ time because the reactions have to be scanned only once, assuming that a vector with a boolean entry per species is used to check (in constant time) if it has been already added to the list. Indices from 0 to $s-1$ and from 0 and $l-1$ are implicitly assigned to each species and label, respectively. A reaction is a structure with two fields, one for each possible reagent, and a list of pairs in the form `(species, multiplicity)` for the products. Storing R requires $\mathcal{O}(p \cdot r)$ space.

We make use of two vectors, `inc` and `out`, indexed by species. Each `inc`$[X]$ entry points to a list of pairs `(reaction, multiplicity)` containing all reactions with X in their products, accompanied by the corresponding product multiplicity of X for each reaction. Note that each reaction may appear in `inc`$[X]$ for at most p species, thus requiring $\mathcal{O}(p \cdot r)$ to store `inc`. The vector `out` is similar, but each `out`$[X]$ entry points to a list of reactions having X in their reagents. The space required by `out` is thus $\mathcal{O}(r)$.

In the algorithm we build sets of elements. However, insertions in sets can be implemented in constant time because an element is never added to a set more than once.

Refinable partition. A partition is stored as a doubly linked list of pointers to its blocks. Each block record contains an integer to store its size and pointers to two doubly linked lists that divide the species into `marked` and `unmarked` (as a result of operations that are used to split blocks, discussed later). Each species has a pointer to its block in the current partition. Thus, finding the block for a species, marking, and unmarking take constant time. Also, it is possible to scan the species of a block in time linear with respect to its size, and to split it in time proportional to the number of marked states.

The operation of splitting a block H creates a new block H_1 containing the marked species of H, while H maintains those that are not marked. This requires to assign the list pointed by H.`marked` to H_1.`unmarked` and to assign an empty list to H.`marked`. These operations are done in constant time, while a time proportional to originally marked species of H is necessary to update their reference to the new block H_1. If instead H originally contained just marked or unmarked species, then no split is actually performed, and marked species get unmarked at no further cost.

Splitters. The list of pointers `spls` refers to the blocks of the current partition that will be used as splitters. An $s \times l$ matrix `M` of real numbers is maintained to efficiently compute conditional, production and flux rates. A possible majority candidate (pmc) of an array `A` of size s is either the value which appears more than $\lfloor s/2 \rfloor$ times in `A`, or any other value if it does not exist. We calculate the pmc row of `M` by extending the algorithm from [34] to vectors in a straightforward manner.

We denote the row of species X in `M` by `M`$[X]$, that is `M`$[X] \in \mathbb{R}^l$. In the course of splitting, we sort species according to the lexicographical order on their rows in `M`. Clearly, sorting a set $H \in \mathcal{H}$ takes $\mathcal{O}(l \cdot |H| \cdot \log |H|)$ time, as $\mathcal{O}(|H| \cdot \log |H|)$ comparisons are needed, each requiring $\mathcal{O}(l)$ time.

This leads to an overall $\mathcal{O}(p \cdot r + l \cdot s) \leq \mathcal{O}(s \cdot r + r \cdot s) = \mathcal{O}(r \cdot s)$ space complexity. Other auxiliary lists and sets of pointers presented in the remainder of the section will respect the space bound given above.

```
1  CoarsestRNBisimulation(χ, S, R, H) :=
2    M = build an s × l matrix of reals
3    if (χ = FB)
4      H = RefineCRR(S, R, M, H)
5    spls = shallow copy of H
6    while (spls ≠ ∅)
7      H_sp = pop(spls)
8      Split(χ, S, R, M, H, H_sp, spls)
```

Algorithm 1. Computation of the coarsest bisimulations.

3.3 Overview

Algorithm 1 provides the parametric procedure `CoarsestRNBisimulation` for computing the coarsest RN bisimulations that refine a given initial partition $\mathcal{H}$ of species of an RN (S, R). The first argument (χ) specifies either FB or BB.

We first observe that the $\mathbf{crr}$-condition of FB can be implemented as an initialization step that pre-partitions the species according to the values of $\mathbf{crr}$. This is because $\mathbf{crr}$ is a "global" property of the RN, i.e., it does not depend on the current partition. Instead, the conditions on $\mathbf{pr}$ and $\mathbf{sr}$ for FB and BB, respectively, require the iterative partition-refinement treatment. Consequently, our algorithm starts (Lines 3-4) by invoking, if necessary, the `RefineCRR` procedure.

RefineCRR *(Algorithm 2).* This procedure provides the coarsest refinement of $\mathcal{H}$ which satisfies the $\mathbf{crr}$-condition of FB. It refines $\mathcal{H}$ according to the ρ-reaction rates for each species X and label ρ. In particular, in this procedure each entry $\mathtt{M}[X][\rho]$ is used to store $\mathbf{crr}(X, \rho)$, and is assumed to be initialized with 0. We can thus compute the values of $\mathbf{crr}$ for all labels and species in one iteration of R only (Lines 3-7), requiring $\mathcal{O}(r)$ time. Then, we refine $\mathcal{H}$ (Lines 10-12). This can be done, for each initial block $H \in \mathcal{H}$, by sorting the species $X \in H$ according to a lexicographical ordering on their $\mathtt{M}[X]$ row. After sorting, all species belonging to the same sub-block will be alongside each other, and it is easy to transform them into new blocks in $\mathcal{O}(|H|)$ time. As discussed, the sorting of each block requires $\mathcal{O}(l \cdot |H| \cdot \log|H|)$ time, and the total time spent in sorting is thus $\mathcal{O}(l \cdot \sum_{H \in \mathcal{H}} |H| \cdot \log|H|) \leq \mathcal{O}(l \cdot \sum_{H \in \mathcal{H}} |H| \cdot \log s) = \mathcal{O}(l \cdot s \cdot \log s)$. Finally, Line 13 resets to 0 all entries of `M`, requiring $\mathcal{O}(l \cdot s)$ time.

Overall, this yields $\mathcal{O}(r + l \cdot s \cdot \log s)$ time complexity. Given that $s \leq (2 + p) \cdot r$, this can be bounded by $\mathcal{O}(r \cdot p \cdot l \cdot \log s)$.

Iterative Refinement *(Algorithm 1, Lines 5-8).* The procedure performs the iterative partition refinement required by our bisimulations as an extension of the algorithm for Markov chains of [34], as discussed. If $\chi = FB$, blocks of $\mathcal{H}$ are *split* into sub-blocks of species with same ρ-production rates towards the block H_{sp} for all $\rho \in \mathcal{L}(R)$. Instead, if $\chi = BB$, blocks are split with respect to their (H_{sp}, H')-splitter flux rates with respect to all labels $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$.

Line 5 creates the linked list `spls` of initial candidate splitters containing pointers to each $H \in \mathcal{H}$: all blocks of $\mathcal{H}$ are considered as (initial) candidate splitters. Then, Lines 6-8 iterate while there are candidate splitters to be considered: after selecting a

```
1  RefineCRR(S, R, M, H) :=
2    //Iterate once R to compute crr[X, ρ] for all ρ and X
3    forall (X --α--> π ∈ R)
4      M[X][∅] = M[X][∅] + α
5    forall (X + Y --α--> π ∈ R)
6      M[X][Y] = M[X][Y] + α
7      M[Y][X] = M[Y][X] + α
8    //Refine H according to the M rows, and store it in H'
9    H' = ∅
10   forall (H ∈ H)
11     Sort and split H wrt crr[X], for all X ∈ H, yielding H_1, ..., H_b
12     Add H_1, ..., H_b to H'
13   CleanRowsOfMatrix(M,S)
14   return H'
15
16 CleanRowsOfMatrix(M, H) :=
17   forall (X ∈ H and ρ ∈ L(R))
18     M[X][ρ]=0
```

Algorithm 2. Pre-partitioning according to the condition of FB on $\mathbf{crr}$.

splitter (H_{sp}) and removing it from `spls`, the procedure `Split` is invoked to refine each block of $\mathcal{H}$ with respect to H_{sp}.

We now provide an overview of the `Split` procedure (Algorithm 3). A detailed presentation is given in Section 4, together with the complexity results. `Split` first computes either $\mathbf{pr}[X, H_{sp}, \rho]$ for all $X \in S$ and $\rho \in \mathcal{L}(R)$ (FB case) or $\mathbf{sr}[X, H_{sp}, H']$, for all $X \in S$ and $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$ (BB case). The rates are computed for all labels at once and are stored in `M` similarly to `RefineCRR`. We remark that BB uses different labels than FB. Nevertheless, as will be discussed in Section 4, the number of labels used by BB is bounded by l as well, and hence we can safely use `M` also in the BB case.

Then, we iterate over the set of blocks containing a species for which at least one non-zero rate has been computed. Each partition block H is split in sub-blocks with either same $\mathbf{pr}[\cdot, H_{sp}, \rho]$ for all $\rho \in \mathcal{L}(R)$ (FB), or same $\mathbf{sr}[\cdot, H_{sp}, H']$ for all $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$ (BB), updating the list `spls`. Following the usual approach of Paige and Tarjan [27], a sub-block with maximal size is not added to `spls`. However, this is done only if the block that is split (i.e., H) has been already used as a splitter, as otherwise the algorithm would be incorrect (see the discussion in [34]).

4 The Split Procedure

We now provide a detailed description of the `Split` procedure shown in Algorithm 3. It begins (Line 2) by initializing the set of pointers S_T that will refer to all species X for which either there exists a ρ such that $\mathbf{pr}[X, H_{sp}, \rho] \neq 0$ if $\chi = FB$, or for which there exists a block H' (or $\{\emptyset\}$) such that $\mathbf{sr}[X, H_{sp}, H'] \neq 0$ if $\chi = BB$. Similarly, Line 3 initializes the set H_T which will point to the blocks of the species in S_T. We remark that only the blocks in H_T may be split due to the current splitter H_{sp}. If $\chi = FB$, Lines 4-8 compute $\mathbf{pr}[X, H_{sp}, \rho]$ and store it in $\mathtt{M}[X][\rho]$ for each X and ρ. This is done by `ComputePR` in Algorithm 4. The procedure scans all the reactions in the `inc` list of each $Y \in H_{sp}$. We can have either unary or binary reactions (Lines 2-3 or 4-6,

```
1  Split(χ,S,R,M,H,H_sp,spls) :=
2    S_T = ∅ //Set of species X with at least a non-zero pr/sr[X,H_sp,·]
3    H_T = ∅ //Set of blocks containing the species in S_T
4    forall(Y ∈ H_sp)
5      if(χ = FB)
6        ComputePR(Y,M) //Compute pr[X,Y,ρ] for all X and ρ. Populate S_T
7      else
8        ComputeSR(Y,H_sp,M) //Compute sr[X,Y,H'] for all X and H'. Populate S_T
9    //Now each M[X][ρ] stores pr[X,H_sp,ρ] (or sr[X,H_sp,H'], with ρ = H'.label)
10   forall(X ∈ S_T)
11     H = get block of X
12     Discard label of H, if any
13     if(M[X] is not a zero row) //Discard spurious species from S_T
14       if(H contains no marked states) //Add only once H to H_T
15         Add H to H_T
16       Mark X in H
17   while(H_T ≠ ∅)
18     H = pop(H_T)
19     H_1 = marked states of H
20     H = not marked states of H
21     if(H = ∅)
22       Give the identity of H to H_1
23     else
24       Make H_1 a new block
25     pmc = PMCRow(H_1,M)
26     H_2 = {X ∈ H_1 | M[X] not equal to the pmc-row}
27     H_1 = H_1 \ H_2
28     if(H_2 = ∅)
29       b = 1 //No need to split H_1.
30     else
31       Sort and split H_2 according to M[X], yielding H_2,...,H_b
32       Make each of H_2,...,H_b a new block
33     if(H ∈spls)
34       Add H_1,...,H_b except H to spls
35     else
36       Add [H,]?H_1,...,H_b to spls except a sub-block of maximal size
37   while(S_T ≠ ∅)
38     X = pop(S_T)
39     touched[X]=false
40     CleanRowsOfMatrix(M,X)
```

Algorithm 3. The Split procedure.

respectively). In the latter case, if the two reagents are equal (i.e., $X = X'$) we add $\alpha \cdot \pi(Y)$ twice to $\mathtt{M}[X][X]$. This corresponds to the $\rho(X)+1$ factor of Definition 1. The actual updates on the entries of M are performed by the simple sub-routine `Update` in Lines 9-13 of Algorithm 4 which also updates S_T if necessary.

If $\chi = BB$, Lines 4-8 of Algorithm 3 compute $\mathbf{sr}[X, H_{sp}, H']$ and store it in $\mathtt{M}[X, \rho_{H'}]$ for each $X \in S$ and $H' \in \mathcal{H} \cup \{\{\emptyset\}\}$, with $\mathcal{H}$ the current partition. The symbol $\rho_{H'}$ denotes a label in $\mathcal{L}(R)$ which identifies H' and is discussed below. The flux rates are computed by `ComputeSR` of Algorithm 5. It is similar to `ComputePR`, but it scans the reactions in the `out` lists of each species $Y \in H_{sp}$. By Definition 5, unary reactions contribute to splitter flux rates with $\{\emptyset\}$ as third parameter. Here we associate the label $\emptyset \in \mathcal{L}(R)$ to unary reactions. For each unary reaction $Y \xrightarrow{\alpha} \pi \in \mathtt{out}[Y]$ (Lines 2-5), $\mathtt{M}[Y][\emptyset]$ is decreased by α and we increase $\mathtt{M}[X][\emptyset]$ of each species X in π by $\alpha \cdot \pi(X)$. Instead, each binary reaction $Y + Y' \xrightarrow{\alpha} \pi \in \mathtt{out}[Y]$ contributes to those with the block of Y' as third parameter. As depicted in Lines 6-15, we provide each

```
1  ComputePR(Y,M):=
2    forall((X --α--> π,π(Y)) ∈ inc[Y])
3      Update(M,X,∅,π(Y),α)
4    forall((X+X' --α--> π,π(Y)) ∈ inc[Y])
5      Update(M,X,X',π(Y),α)
6      Update(M,X',X,π(Y),α)
7
8  //Sub-routine to update M and S_T
9  Update(M,X,ρ,mult,α):=
10   if(!touched[X])
11     touched[X] = true
12     add X to S_T
13   M[X][ρ] = M[X][ρ] + α·mult
```

Algorithm 4. Compute $\mathbf{pr}$ wrt the splitters.

```
1  ComputeSR(Y,H_sp,M):=
2    forall(Y --α--> π ∈ out[Y])
3      Update(M,Y,∅,1,−α)
4      forall(X ∈ π)
5        Update(M,X,∅,π(X),α)
6    forall(Y+Y' --α--> π ∈ out[Y])
7      H' = get block of Y'
8      if(H' does not have a label)
9        H'.label = Y'
10     if(Y ≠ Y' and H' = H_sp)
11       α = α/2
12     Update(M,Y,H'.label,1,−α)
13     Update(M,Y',H'.label,1,−α)
14     forall(X ∈ π)
15       Update(M,X,H'.label,π(X),α)
```

Algorithm 5. Compute $\mathbf{sr}$ wrt the splitters.

block H' with a field `label` used to point to the label in $\mathcal{L}(R)$ assigned to H'. This will be a species in $H' \cap \mathcal{L}(R)$. In particular, in Line 7 we get the block of Y' (H'). Then, if no label is currently assigned to H', we set Y' as label of H'. Finally, the entries of `M` are updated by `Update` similarly to the FB case, but using H'.`label` as label. Note that all reactions involving species Y' of a block H' will contribute to the same H'.`label` entries of `M`, thus computing the summation over the elements of H' of Definition 5. We remark that we may have blocks $H'' \in \mathcal{H}$ with $H'' \cap \mathcal{L}(R) = \emptyset$. Those do not contribute to `ComputeSR` as both reagents of an arbitrary binary reaction are elements of $\mathcal{L}(R)$. Finally, we note that in Lines 10-11 we halve the rate of reactions with two different reagents $Y + Y'$ belonging to the splitter block H_{sp}, as done in Definition 5.

Now that S_T and `M` have been populated, Lines 10-16 of Algorithm 3 build H_T and mark all species in S_T as discussed in Section 3.2. The marking operation could have not been done in Lines 4-8 because it changes the order of species in a block, and hence might interfere with the iteration of the `forall` statement of Line 4. Note that Line 13 discards species in S_T whose `M`-rows have only zeros. This can happen because positive and negative values can sum up to zero (see, e.g., lines 3 and 5 of Algorithm 5). In addition, Line 12 reinitializes all `label` fields of the blocks in H_T, a super-set of those to which `ComputeSR` might have assigned a label.

It is now possible to refine $\mathcal{H}$ and update the list of candidate splitters by splitting each block $H \in H_T$ according to the $\mathbf{pr}$ or $\mathbf{sr}$ values (Lines 17-36). Lines 19-20 perform the split operation discussed in Section 3.2. They split (in constant time) the species in H which appear in H_T (the marked ones) from those which do not appear in H_T (the unmarked ones). Those $X \in H$ that yield $\mathtt{M}[X][\cdot] \neq 0$ form the block H_1, while the other remain in H. If the new H is empty, H_1 contains the elements originally present in H and thus receives its identity. Otherwise H_1 is made to a new block in $O(|H_1|)$ time.

Lines 25-27 further split H_1 by moving some of its elements in a new block H_2 in $O(|H_1|)$ time. In particular, we calculate the `pmc`-row in order to split H_1 into (a new) H_1 and H_2. In case more than half of the species of the original H_1 have their `M`-row equal, the new block H_1 will contain those species with the `pmc`-row; otherwise, it will contain any sub-set of H_1 with same row in `M`. In both cases the obtained H_1 does not

need to be further split. Instead H_2 might need to be split further. We note that H_2 might be empty, meaning that there was no need in splitting H_1. In such case H_1 remains unchanged; in the opposite case, instead, H_2 is split in Lines 31-32 and the obtained sub-blocks are added to $\mathcal{H}$. We remark that we are guaranteed that each sub-block of H_2 has at most half the elements originally in H. Moreover, it is worth noting that splitting blocks in $\mathcal{H}$ affects `spls` because `spls` stores pointers to the elements of $\mathcal{H}$.

Finally, we add the so obtained sub-blocks to `spls` by storing the corresponding pointers in `spls`. As discussed, we do not add a sub-block with maximal size if the original H has already been used as splitter (Line 36). Note that $[H,]^? H_1$ means that we add only one of the two blocks to `spls` if Line 22 gave the identity of H to H_1. Instead, in Line 34 there is no need to add the new H to `spls` because it is already there (i.e., the original H was there, and hence the refined H inherited its presence).

The procedure terminates by resetting the vector `touched`, used to build S_T, and the rows of `M` regarding the species in S_T.

Theorem 2. *Algorithm 1 calculates the coarsest RN bisimulations that refine a partition $\mathcal{H}$. Its time complexity is $\mathcal{O}(r \cdot p \cdot l \cdot \log s)$, while its space complexity is $\mathcal{O}(r \cdot s)$.*

The proof lifts the ideas of [34] to RNs. As discussed previously, the complexity stated above relates to an arbitrary RN. We shall see next that in the encoding of a polynomial ODE system the factor $p \cdot l$ simplifies to s, while it becomes a constant for CTMCs.

5 Applications

We discuss how to encode into an RN an arbitrary polynomial ODE system of degree at most two. Based on this, we consider the special case of an affine ODE system, which gives reduced time and space complexities; here, we will show an application of RN bisimulations for the numerical solution of systems of linear equations using stationary iterative methods. Finally, we relate RN bisimulations to CTMC lumpability [5].

It is easy to see that the encoding of polynomials ODEs to RNs is not unique (cf. [16] for CRNs). Here, we propose one for which the algorithmic complexity can be directly related to the number of monomials appearing in the ODE system, leaving the question of investigating minimality issues to future work.

Polynomial systems. We consider the ODE system $\dot{y} = G(y)$ with components

$$\dot{y}_k = G_k(y) := \sum_{1 \le i,j \le n} \alpha_{i,j}^{(k)} \cdot y_i \cdot y_j + \sum_{1 \le i \le n} \alpha_i^{(k)} \cdot y_i + \beta^{(k)}, \quad 1 \le k \le n, \qquad (2)$$

and with $\alpha_{i,j}^{(k)}, \alpha_i^{(k)}, \beta^{(k)} \in \mathbb{R}$.

Lemma 1. *The RN (S_G, R_G), with $S_G := \{1, \ldots, n\}$ and*

$$R_G := \left\{ i + j \xrightarrow{\alpha_{i,j}^{(k)}} i + j + k \mid \alpha_{i,j}^{(k)} \neq 0 \right\} \cup \left\{ i \xrightarrow{\alpha_i^{(k)}} i + k \mid \alpha_i^{(k)} \neq 0 \right\} \cup \left\{ \emptyset \xrightarrow{\beta^{(k)}} k \mid \beta^{(k)} \neq 0 \right\},$$

has ODEs $\dot{V}_k = G_k(V)$, for $1 \le k \le n$.

Note that with this encoding r relates to the number of monomials used in the ODEs (while s is the number of ODE variables). As anticipated in Section 1, Theorem 2 and Lemma 1 imply that Algorithm 1 gives the coarsest FB and BB partitions of an arbitrary polynomial ODE system in $\mathcal{O}(r \cdot s \cdot \log s)$ time and $\mathcal{O}(r \cdot s)$ space.

Affine systems. Equation (2) also subsumes the interesting case of affine ODE systems where $G(y) = Cy + d$ for some $C \in \mathbb{R}^{n\times n}$ and $d \in \mathbb{R}^n$. In this case, Theorem 2 and Lemma 1 imply that the complexity reduces to $\mathcal{O}(r \cdot \log s)$ time and $\mathcal{O}(r + s)$ space. Here we consider the problem of computing a solution of a linear system of equations $Ax = b$, with $A \in \mathbb{R}^{n\times n}$ and $x, b \in \mathbb{R}^n$. Stationary iterative methods approximate a solution with updates in the form $x(k+1) = F(x(k))$ where k is the iteration index and F is affine. For instance, Jacobi's method is written as $x(k + 1) = -Rx(k) + D^{-1}b$, with $x(0) = 0$, where D, R are such that D is a diagonal matrix and $A = D + R$. Under the assumption of strict diagonal dominance for A, it converges to the solution of $Ax = b$ (e.g., [28]). We interpret this sequence as a dynamical system, but in *discrete time*, and observe that the bisimulations carry over to the discrete time case. We denote the encoding of the Jacobi iterations by the RN $(S_{A,b}, R_{A,b})$. Then, the following holds.

Theorem 3. *An RN bisimulation $\mathcal{H} = \{H_1, \ldots, H_m\}$ on $(S_{A,b}, R_{A,b})$ induces a reduced discrete-time model $\hat{x}(k + 1) = \hat{A}\hat{x}(k) + \hat{b}$, with $\hat{A} \in \mathbb{R}^{m\times m}$ and $\hat{x}(k), \hat{b} \in \mathbb{R}^m$. If $\mathcal{H}$ is an FB then, $\hat{x}_i(k) = \sum_{l\in H_i} x_l(k)$ for all $1 \leq i \leq m$ and $k \geq 0$. If $\mathcal{H}$ is a BB then, $\hat{x}_i(k) = x_l(k)$ for all $1 \leq i \leq m$, $l \in H_i$ and $k \geq 0$.*

Here, $\hat{A}$ and $\hat{b}$ can be obtained by constructing the reduced RN up to FB/BB [7].

Continuous-time Markov chains. Let us fix a CTMC with transition rate matrix $Q = (q_{i,j})_{1\leq i,j\leq n}$. Then the probability distribution $\pi = (\pi_i)_{1\leq i\leq n}$ solves the Kolmogorov linear ODEs $\dot{\pi} = \pi Q$.

Lemma 2. *Let Q be the transition matrix of a CTMC and (S_Q, R_Q) be the RN encoding according to (2) of its Kolmogorov ODEs. Then, $\mathcal{H}$ is an ordinarily (resp., exactly) lumpable partition for Q if and only if $\mathcal{H}$ is an FB (resp., a BB) for (S_Q, R_Q).*

By Theorem 2 and Lemma 1, Algorithm 1 calculates the coarsest ordinarily and exactly lumpable partitions of Q in $\mathcal{O}(r \cdot \log s)$ time and $\mathcal{O}(r + s)$ space. Thus, we recover the bounds of Markov-chain specific algorithms [13,34]. We also remark that, in the case of BB, Lemma 2 recovers a result from [8] using an alternative proof. Finally, it can be shown that Lemma 2 is still valid if the reactions are encoded via $R_Q = \{i \xrightarrow{q_{i,j}} j \mid q_{i,j} \neq 0\}$, using one-to-one reactions only. Though not affecting asymptotic complexity, this reduces memory and time consumption, and thus we will use it in our prototype.

6 Evaluation

We evaluate our algorithm using i) the biochemical networks evaluated in [7] as case studies for degree-two polynomial systems; ii) $Ax = b$ systems from [12]; and iii) selected CTMCs from the MRMC distribution [20]. Comparing against the reductions of [7] and MRMC also allowed us to validate the implementation of our algorithm.

Original model				*FB reduction*				*BB reduction*			
Id	*Ref.*	$\|R\|$	$\|S\|$	*Red.(s)* [7]	*Red.(s)*	$\|R\|$	$\|S\|$	*Red.(s)* [7]	*Red.(s)*	$\|R\|$	$\|S\|$
Biochemical reaction networks											
M1	[30]	3 538 944	262 146	4.61E+4	7.49E+0	990	222	7.65E+4	1.21E+1	2 614	222
M2	[30]	786 432	65 538	1.92E+3	1.58E+0	720	167	3.68E+3	2.51E+0	1 873	167
M3	[30]	172 032	16 386	8.15E+1	2.89E–1	504	122	1.77E+2	6.03E–1	1 305	122
M4	[30]	48	18	1.00E–3	1.00E–3	24	12	2.00E–3	2.00E–3	44	12
M5	[31]	194 054	14 531	3.72E+1	3.88E–1	142 165	10 855	1.32E+3	6.00E–1	91 001	6 634
M6	[14]	187 468	10 734	3.07E+1	6.09E–1	57 508	3 744	2.71E+2	1.40E+0	145 650	5 575
M7	[14]	32 776	2 506	1.26E+0	1.19E–1	16 481	1 281	1.66E+1	2.14E–1	*32 776*	*2 506*
M8	[2]	41 233	2 562	1.12E+0	2.69E–1	33 075	1 897	1.89E+1	3.97E–1	*41 233*	*2 562*
M9	[2]	5 033	471	1.91E–1	1.60E–2	4 068	345	4.35E–1	2.40E–2	*5 033*	*471*
M10	[3]	5 797	796	1.61E–1	1.90E–2	4 210	503	7.37E–1	3.30E–2	*5 797*	*796*
M11	[9]	5832	730	3.89E–1	1.50E–2	1296	217	6.00E–1	2.40E–2	237	217
M12	[21]	487	85	2.00E–3	2.00E–3	264	56	6.00E–3	3.00E–3	431	56
M13	[6]	24	18	1.20E–2	4.00E–3	*24*	*18*	7.00E–3	4.00E–3	7	3
Affine systems											
F1	[12]	10 319 760	1 489 753	9.74E+3	8.70E+2	1 295 514	188 101	—	2.23E+2	*10 319 760*	*1 489 753*
F2	[12]	8 814 880	1 270 433	8.86E+2	5.58E+2	1 108 224	160 951	—	1.55E+2	4 420 168	639 509
F3	[12]	2 101 250	525 826	3.71E+2	1.24E+1	526 338	131 842	—	4.79E+1	*2 101 250*	*525 826*
F4	[12]	4 706 074	143 572	6.72E+0	6.70E+0	565 288	47 858	—	1.47E+1	2 739 188	112 444
F5	[12]	706 577	116 836	3.23E+0	3.11E+0	609 459	73 423	—	2.86E+0	609 307	73 348
Continuous-time Markov chains											
C1	[25]	22 871 849	3 101 445	4.00E+4	2.01E+3	1 069 777	135 752	—	1.34E+3	1 166 931	148 092
C2	[17]	11 583 520	2 373 652	1.73E+2	9.78E+1	5 792 531	1 187 597	—	3.07E+2	5 814 622	1 187 597
C3	[22]	10 485 761	1 048 576	1.48E+1	1.76E+1	3301	792	—	1.23E+1	5083	792

Table 1. FB and BB reductions. Entries labeled with "—" indicate that the reduction algorithm did not terminate within 24 hours. Greyed out entries indicate no reduction.

The results are presented in Table 1. To ease layout, we label the models with short identifiers (first column), and refer to the publications in the second column for details. Headers $|R|$ and $|S|$ give the number of reactions and species of the original and reduced RNs. Headers "*Red.* [7]" and "*Red.*" provide the runtimes of the algorithm considered in [7] and of the proposed approach, respectively. Measurements were taken on a 2.6 GHz Intel Core i5 machine with 4 GB of RAM. The experiments are replicable using a prototype available at http://sysma.imtlucca.it/crnreducer/.

***Biochemical models* (M1–M13).** For consistency, we computed the coarsest bisimulations that refine the same initial partitions as specified in [7]. Specifically, for each RN (S, R), in the case of FB we considered the trivial partition $\{S\}$ (thus yielding the largest bisimulation); due to the side condition of BB, in that case the initial partition was chosen in agreement with the initial conditions — two species are in the same initial block if their initial conditions, read from the original model specification, are equal (thus ensuring that the reduction is a lossless simplification of the original one).

We refer to [7] for a description of the models and the biological interpretation of the bisimulations therein computed. Here, we confirm the same reductions, at a much improved performance over that of [7]. For the largest model (M1) we registered a speedup of four orders of magnitude —now all cases can be reduced within seconds.

***Systems of linear equations* (F1–F5).** These are encodings of the Jacobi iterative method to solve large-scale real-world linear systems from the Sparse Matrix Collection [12]. F1–F2 (original names *Bourchtein/atmosmodl* and *Bourchtein/atmosmodd*, respectively) arise from atmospheric modeling; F3 (*Wissgott/parabolicFEM*) is to be computed during a finite-element-method solution to a convection-diffusion reaction; F4 (*TTK/engine*) comes from a problem in structural mechanics; F5 (*IBMEDA/dc1*) arises from the simulation of an electrical circuit. For F1–F4 we verified (in $\mathcal{O}(r)$ steps) that the sparse matrix is strictly diagonal dominant, a known sufficient condition for the convergence of Jacobi's method. All cases enjoy significant reductions with either bisimulation, up to one order of magnitude fewer species and reactions for F1.

In some cases (i.e., F2, F4, and F5) the FB runtimes are comparable to those of [7]. This can be explained by noting that, in the encoding of affine ODEs, the splitting based on the "labels" cannot yield a significant improvement because the RN has only unary reactions (hence only one label, $\emptyset$). This is not the case in the biochemical benchmarks M1–M13, which as a matter of fact showed significant runtime differences.

Regarding BB, the algorithm of [7] was not able to compute any BB reduction within 24 hours. The remarkable performance improvement is due to the novel splitter-based characterization of BB (Section 3.1), while with [7] it was required to compute, at each iteration, the equivalence classes for multi-sets of species according to Definition 4.

***CTMCs* (C1–C3).** These are the three largest CTMCs of the MRMC distribution [20], used in [19] to study the impact of ordinary CTMC lumpability in model checking. In particular, these are: a protocol for wireless group communication (C1, original model name *FDT3E3_PE16E4_S4OD40*); a cluster model (C2, *WORKSTATION_CLUSTER_N256*); and a peer-to-peer protocol (C3. *TORRENT_N04*). The initial partitions for both FB and BB are consistent with the atomic propositions on the CTMC states.

Being affine ODE systems, the above observations regarding the runtime comparisons with [7] carry over to these models. Instead, a thorough comparison against MRMC is difficult because of the different languages were used for the implementation (C with specialized data structures for sparse matrices for MRMC, vs. Java with plain data structures from its API in our prototype) and because MRMC is CTMC-specific. However, MRMC ran one order of magnitude faster and was less memory demanding, indicating the potential in improving performance in optimized versions of our prototype.

7 Conclusion

The main advantage in aggregating dynamical systems from a chemical reaction network syntax lies in adapting established and efficient bisimulation algorithms for discrete-state models. The numerical benchmarks have demonstrated scalability as well as the effectiveness of exact aggregations in non-synthetic models. Future work will concern the equivalences and related algorithms to handle higher-degree polynomial nonlinearities.

Acknowledgment

This work was partially supported by the EU project QUANTICOL, 600708. L. Cardelli is partially funded by a Royal Society Research Professorship.

References

1. C. Baier, B. Engelen, and M. E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60(1):187–231, 2000.
2. D. Barua, J. R. Faeder, and J. M. Haugh. A bipolar clamp mechanism for activation of jak-family protein tyrosine kinases. *PLoS Computational Biology*, 5(4), 2009.
3. D. Barua and W. S. Hlavacek. Modeling the effect of apc truncation on destruction complex function in colorectal cancer cells. *PLoS Comput Biol*, 9(9):e1003217, 09 2013.
4. M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.
5. P. Buchholz. Exact and ordinary lumpability in finite markov chains. *Journal of Applied Probability*, 31(1):59–75, 1994.
6. L. Cardelli. Morphisms of reaction networks that couple structure to function. *BMC Systems Biology*, 8(1):84, 2014.
7. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Forward and backward bisimulations for chemical reaction networks. In *CONCUR*, pages 226–239, 2015.
8. L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Symbolic computation of differential equivalences. In *POPL*, 2016. To appear.
9. J. Colvin, M. I. Monine, J. R. Faeder, W. S. Hlavacek, D. D. Von Hoff, and R. G. Posner. Simulation of large-scale rule-based models. *Bioinformatics*, 25(7):910–917, 2009.
10. T. Dang, C. Le Guernic, and O. Maler. Computing reachable states for nonlinear biological models. *TCS*, 412(21):2095–2107, 2011.
11. 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 *LICS*, pages 362–381, 2010.
12. T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1, 2011.
13. S. Derisavi, H. Hermanns, and W.H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.
14. J. R. Faeder, W. S. Hlavacek, I. Reischl, M. L. Blinov, H. Metzger, A. Redondo, C. Wofsy, and B. Goldstein. Investigation of early events in FcεRI-mediated signaling using a detailed mathematical model. *The Journal of Immunology*, 170(7):3769–3781, 2003.
15. M. Feinberg. Chemical reaction network structure and the stability of complex isothermal reactors — I. the deficiency zero and deficiency one theorems. *Chemical Engineering Science*, 42(10):2229 – 2268, 1987.
16. V. Hars and J. Toth. On the inverse problem of reaction kinetics. In *Colloquia Mathematica Societatis Janos Bolyai*, 1979.
17. B.R. Haverkort, H. Hermanns, and J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *SRDS*, pages 228–237, 2000.
18. G. Iacobelli, M. Tribastone, and A. Vandin. Differential Bisimulation for a Markovian Process Algebra. In *MFCS*, 2015.
19. J.-P. Katoen, T. Kemna, I.S. Zapreev, and D.N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *TACAS*, pages 87–101, 2007.
20. J.-P. Katoen, M. Khattri, and I.S. Zapreev. A Markov Reward Model Checker. In *QEST*, pages 243–244, 2005.
21. P. Kocieniewski, J. R. Faeder, and T. Lipniacki. The interplay of double phosphorylation and scaffolding in MAPK pathways. *Journal of Theoretical Biology*, 295:116–124, 2012.
22. M. Z. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *CAV*, pages 234–248, 2006.

23. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
24. G. Li and H. Rabitz. A general analysis of exact lumping in chemical kinetics. *Chemical Engineering Science*, 44(6):1413 – 1430, 1989.
25. M. Massink, J.-P. Katoen, and D. Latella. Model checking dependability attributes of wireless group communication. In *DSN*, pages 711–720, 2004.
26. J. D. Murray. *Mathematical Biology I: An Introduction*. Springer, 3rd edition, 2002.
27. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
28. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.
29. M. A. B. Sassi, R. Testylier, T. Dang, and A. Girard. Reachability analysis of polynomial systems using linear programming relaxations. In *ATVA*, pages 137–151, 2012.
30. M. W. Sneddon, J. R. Faeder, and T. Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. *Nature Methods*, 8(2):177–183, 2011.
31. R. Suderman and E. J. Deeds. Machines vs. ensembles: Effective MAPK signaling through heterogeneous sets of protein complexes. *PLoS Comput Biol*, 9(10):e1003278, 10 2013.
32. J. Toth, G. Li, H. Rabitz, and A. S. Tomlin. The effect of lumping and expanding on kinetic differential equations. *SIAM Journal on Applied Mathematics*, 57(6):1531–1556, 1997.
33. M. Tschaikowski and M. Tribastone. Exact fluid lumpability for Markovian process algebra. In *CONCUR*, LNCS, pages 380–394, 2012.
34. A. Valmari and G. Franceschinis. Simple $O(m \log n)$ time Markov Chain lumping. In *TACAS*, pages 38–52, 2010.