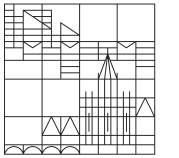
Universität Konstanz



Thesis Proposal Summary

Fabian Klopfer

Modelling of Complex Self-Organizing Systems Group, September 10, 2020

Introduction

In the last presentation . . .

- introduced CRNs & ODEs
- looked at Tribastone et al. [1]
- compared the algorithms informally
- tried to find representations between CRNs & WAs

What we discussed then

- Look at classical lumping rather than CRN species lumping
- Construct “middle ground” between WAs & CTMCs using fully probabilistic Segala
- Elaborate on connection between Kiefer [2] and partition refinement-based lumping e.g. Valmari et al. [3]

Tribastone et al. & Valmari et al. I

```
1  LargestEquivalence(χ,S,R,ℋ) :=
2    Init(R)
3    if(χ = B)
4      ℋ = BackwardPrepartitioning(S,R,ℋ,M)
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 S1. Computation of the largest equivalences.

```
1  Split(χ,S,R,M,ℋ,H_sp,spls) :=
2    T_S = ∅ //Set of species S_i with at least a non-zero fr/br[S_i,·,H_sp]
3    T_H = ∅ //Set of blocks containing the species in T_S
4    forall(S_j ∈ H_sp)
5      if(χ = F)
6        ComputeFR(S_j,M) //Compute fr(S_i,ρ,S_j) for all S_i,ρ. Populate T_S
7      else
8        ComputeBR(S_j,H_sp,M) //Compute br(S_i,ℳ,S_j) for all S_i,ℳ. Populate T_S
9    //M[S_i][ρ] stores fr(S_i,ρ,H_sp), or br(S_i,ρ_ℳ,H_sp), with ρ_ℳ a chosen ρ' ∈ ℳ
10   forall(S_i ∈ T_S)
11     H = get block of S_i
12     Discard label of H, if any
13     if(M[S_i] is not a zero row) //Discard spurious species from T_S
14       if(H contains no marked states) //Add only once H to T_H
15         Add H to T_H
16       Mark S_i in H
17   while(T_H ≠ ∅)
18     H = pop(T_H)
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 = {S_i ∈ H_1 | M[S_i] 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[S_i], 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_i]?H_1,...,H_b to spls except a sub-block of maximal size
37   while(T_S ≠ ∅)
38     S_i = pop(T_S)
39     touched[S_i]=false
40     forall(ρ ∈ ℒ(R))
41       M[S_i][ρ]=0
```

```
1  U_B := ℐ ;  B_T := ∅ ;  w[s] := unused for every s ∈ S ;  𝒞 := {S ∪ {s_⊥}}
2  while U_B ≠ ∅ do
3      let B' be any block in U_B ;  U_B := U_B \ {B'} ;  𝒞 := 𝒞 \ {C_B'} ∪ {B', C_B' \ B'}
4      S_T := ∅
5      for s' ∈ B' do for s ∈ •s' do
6          if w[s] = unused then S_T := S_T ∪ {s} ;  w[s] := W(s,s')
7          else w[s] := w[s] + W(s,s')
8      for s ∈ S_T do if w[s] ≠ 0 then
9          B := the block that contains s
10         if B contains no marked states then B_T := B_T ∪ {B}
11         mark s in B
12     while B_T ≠ ∅ do
13         let B be any block in B_T ;  B_T := B_T \ {B}
14         B_1 := marked states in B ;  B := remaining states in B
15         if B = ∅ then give the identity of B to B_1 else make B_1 a new block
16         pmc := possible majority candidate of the w[s] for s ∈ B_1
17         B_2 := {s ∈ B_1 | w[s] ≠ pmc} ;  B_1 := B_1 \ B_2
18         if B_2 = ∅ then ℓ := 1 else
19             sort and partition B_2 according to w[s], yielding B_2, ..., B_ℓ
20             make each of B_2, ..., B_ℓ a new block
21         if B ∈ U_B then add B_1, ..., B_ℓ except B to U_B
22         else add [B_i]? B_1, ..., B_ℓ except a largest to U_B
23     for s ∈ S_T do w[s] := unused
```

Fig. 2. The coarsest lumping algorithm

Tribastone et al. & Valmari et al. II

- Uses MCs & partition-refinement with same complexity bounds [3]
- What Tribastone et al. base their algorithm on [1]:

> *This structural interpretation allows the development of an algorithm for computing maximal equivalences, building on analogous partition refinement techniques developed for Markov chain lumping (16, 17).*

Tribastone et al. & Valmari et al. III

Theorem 3 (Algorithm complexity)**.** *Algorithm S1 calculates the coarsest forward/backward equivalences that refine an input partition $\mathcal{H}$. Its time complexity is $\mathcal{O}(r \cdot p^2 \cdot l \cdot \log s) \leq \mathcal{O}\big(r^2 \cdot p^3 \cdot \log(s)\big)$, while its space complexity is $\mathcal{O}(a_r + r \cdot p \cdot s)$.*

Proof. The proof lifts the ideas of (17) to RNs. As in (17), we extend Algorithm S1 by an adjacent species $X_\perp$, a set of compound blocks $\mathcal{C}$, and two additional operations. In particular, we add the command $\mathcal{C} = \{S, \{X_\perp\}\}$ after the command on Line 5 and we add $\mathcal{C} = (C \setminus \{C_{H_{sp}}\}) \cup \{H_{sp}, C_{H_{sp}} \setminus H_{sp}\}$ after the command on Line 7. At the beginning of each iteration of the while loop on Line 6, any compound block $C \in \mathcal{C}$ on Line 7 can be represented as a unique union of blocks from the current partition $\mathcal{H}$. Here, $C_{H_{sp}}$ refers to the unique compound block such that $H_{sp} \subseteq C_{H_{sp}}$. By replacing statements $W(s_1, B') = W(s_2, B')$ from (17) with $\mathbf{fr}(X_1, \cdot, H_{sp}) = \mathbf{fr}(X_1, \cdot, H_{sp})$ (or $\mathbf{br}(X_1, \cdot, H_{sp}) = \mathbf{br}(X_1, \cdot, H_{sp})$), Lemma 1 from (17) carries over in a verbatim manner, thus showing that our algorithm is correct if and only if its extended version is correct. The same applies to Lemma 2 from (17) which ensures that the algorithm "does not split too much", meaning that the coarsest forward or backward equivalence that refines the partition is returned by the algorithm. By repeating the argumentation from (17), we show the first invariant: At each new iteration on Line 7, each compound block $C \in \mathcal{C}$ can be written as a union of blocks from the current partition $\mathcal{H}$ up to one single block from *spls*. That is, for each $C \in \mathcal{C}$, there exist unique blocks $H_1, \ldots, H_k \in \mathcal{H}$ such that $C = H_1 \cup \ldots \cup H_k$ with $H_1, \ldots, H_{k-1} \in spls$ and $H_k \notin spls$. Using this invariant, we copy-paste the proof for the termination from (17). Afterwards, we observe that the proof of the second invariant carries over, where the second invariant reads as: For any $H \in \mathcal{H}$, $X_1, X_2 \in H$ and $C \in \mathcal{C}$, it holds that $\mathbf{fr}(X_1, \cdot, C) = \mathbf{fr}(X_1, \cdot, C)$ (or $\mathbf{br}(X_1, \cdot, C) = \mathbf{br}(X_1, \cdot, C)$). Using the first and the second invariant, the proof of correctness follows as in (17).

We now turn to the proof of complexity. We first consider the complexity of the algorithm without considering the `Init` procedure. Space complexity has been already considered. Arguing as in (17), one can show that any $S_i \in S$ appears at most $\lfloor \log(s) + 1 \rfloor$ times in a splitter. Thus, if $\chi = F$, Algorithm S1 needs $\mathcal{O}(\sum_{S_i \in S} \log(s) \cdot |\mathbf{nzs}[S_i]| \cdot p_r) \leq \mathcal{O}(\log(s) \cdot r \cdot p^2)$ number of steps if we ignore lines 13, 25, 31 and 40 in Algorithm S4. The p_r factor comes from the fact that `ComputeFR` has to iterate over the pairs $(m, S_i) \in \rho$.

Schützenberger's construction & Bisimulation I

Schützenberger [4] vs. Buchholz [5]:

$$\overrightarrow{\mu}(\sigma) = \overrightarrow{F}\mu(\sigma)\overrightarrow{F}_R^{-1} \qquad \overrightarrow{F}\mu(\sigma) = \overrightarrow{\mu}(\sigma)\overrightarrow{F} \tag{1}$$

$$\hat{P} = WPV \qquad WP = \hat{P}V_R^{-1} \tag{2}$$

With $W \cdot V = I$ and P the transition matrix.

$$W \cdot V = I$$
$$W \cdot V \cdot V_R^{-1} = I \cdot V_R^{-1}$$
$$W = V_R^{-1}$$

Schützenberger's construction & Bisimulation II

Thus we get

$$\overrightarrow{\mu}(\sigma) = \overrightarrow{F}\,\mu(\sigma)\,\overrightarrow{F}_R^{-1} \qquad \overrightarrow{F}\,\mu(\sigma) = \overrightarrow{\mu}(\sigma)\,\overrightarrow{F} \tag{3}$$

$$\hat{P} = WPW_R^{-1} \qquad WP = \hat{P}W \tag{4}$$

WA $\leftrightarrow$ f.p.S. $\leftrightarrow$ CTMC

$\Rightarrow$ A **WA** may be normalized by left. mult. vector of sum of row sums, yielding a **f.p. Segala**, i.e. the row sum of all matrices together equals 1.
Rescaling each row of a **f.p. Segala** to have a row sum of 1 per matrix yields a **labelled DTMC**, which can be viewed as **labelled uniformized CTMC**.

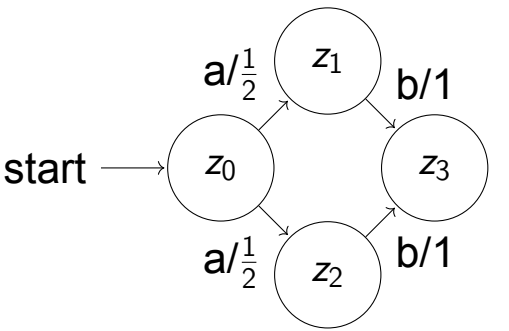
$\Leftarrow$ A **labelled CTMC** can be uniformized to a **labelled DTMC**. Scaling each probability by the number of letters yields a **f.p. Segala**, which is already a **WA**

No need for a common ground: WA $\leftrightarrow$ CTMC is more straight forward: A labelled CTMC is already a WA and scale the WA by the row sum per row and letter to get a labeled DTMC which can be interpreted as uniformized labelled CTMC $\square$

Connection between the algorithms I

Weighted automaton $\mathcal{A} = (n, \Sigma, \mu, \alpha, \eta)$ with

- $n = 3$,
- $\Sigma = \{a, b\}$,
- $\alpha = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$,
- $\eta = (\eta_0, \eta_1, \eta_2, \eta_3)$,
- $\mu(a) = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$,
- $\mu(b) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.



Connection between the algorithms II

$$\mu(a)^2 = (0); \quad \mu(b)^2 = (0); \quad \mu(b)\mu(a) = 0$$

$\Rightarrow$ Words to consider: a, b, ab

$$r^{(i)} = \begin{pmatrix} r_{a,0}^{(i)} & r_{a,1}^{(i)} & r_{a,2}^{(i)} & r_{a,3}^{(i)} \\ r_{b,0}^{(i)} & r_{b,1}^{(i)} & r_{b,2}^{(i)} & r_{b,3}^{(i)} \end{pmatrix}$$

$$v_i = \alpha\mu(a)r_a^{(i)} + \alpha\mu(b)r_b^{(i)} + \alpha\mu(a)\mu(b)r_{ab}^{(i)}$$

$$= \left(0, \frac{1}{2}\alpha_0, \frac{1}{2}\alpha_0, 0\right) r_a^{(i)} + (0, 0, 0, \alpha_1 + \alpha_2)\, r_b^{(i)} + (0, 0, 0, \alpha_0)\, r_{ab}^{(i)}$$

$$= \left(0, \frac{1}{2}r_{a,0}^{(i)}\alpha_0, \frac{1}{2}r_{a,0}^{(i)}\alpha_0, r_{b,0}^{(i)}[\alpha_1 + \alpha_2] + r_{a,0}^{(i)}r_{b,1}^{(i)}[\alpha_0]\right)$$

Connection between the algorithms III

$$\overrightarrow{F} = \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ 0 & \frac{1}{2} r_{a,0}^{(1)} \alpha_0 & \frac{1}{2} r_{a,0}^{(1)} \alpha_0 & r_{b,0}^{(1)} [\alpha_1 + \alpha_2] + r_{a,0}^{(1)} r_{b,1}^{(1)} [\alpha_0] \\ 0 & \frac{1}{2} r_{a,0}^{(2)} \alpha_0 & \frac{1}{2} r_{a,0}^{(2)} \alpha_0 & r_{b,0}^{(2)} [\alpha_1 + \alpha_2] + r_{a,0}^{(2)} r_{b,1}^{(2)} [\alpha_0] \end{pmatrix}$$

Using partition refinement we get $\{s_1, s_2\}$, thus

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Connection between the algorithms IV

Chose $\alpha = (1, 0, 0, 0)$.
Now Kiefer restricts the $r^{(i)}_{\sigma,k}$ to $\mathbb{N}_+$.
Resulting in that it is not possible to find parameters such that $\overrightarrow{F} = W$.
Even when we loosen that restriction, chose $r^{(1)}_{a,0} = 1, r^{(1)}_{b,0} = r^{(1)}_{b,1} = 0, r^{(2)}_{a,0} = 0, r^{(2)}_{b,0} = 1$.
Then almost $\overrightarrow{F} = W$ but the entry with $i = j = 4$ is 0 instead of 1.

Summary I

- Kiefers et al. [2] basis construction not able to produce the same basis as partition refinement
- yields non-sparse transition matrices
- induces computational overhead, possibly more overhead than gaining by dimensionality reduction
- Alternative: Use Householder reflectors for finding the basis [6], yields triangular basis at least
- Partition-refinement does **not** explicitly construct a basis.
- rather constructs new Q-matrix than transforming old one

Summary II

LUMPCTMC(P, Q)

1 $L :=$ blocks of P
2 while $L \neq \emptyset$
3 $\quad S :=$ POP(L)
4 $\quad$ SPLIT(S, P, L)
5 $n' :=$ # of blocks in P
6 allocate $n' \times n'$ matrix Q'
7 initialize Q' to zero
8 for every block B_k of P
9 $\quad x_i :=$ arbitrary state in B_k
10 $\quad$ for every x_j such that $x_i \to x_j$
11 $\quad\quad$ Let B_l be $[x_j]_P$
12 $\quad\quad Q'(B_k, B_l) := Q'(B_k, B_l) + Q(x_i, x_j)$
13 return Q'

Algorithm 1. Pseudocode of the lumping algorithm.

$Q'(B_k, B_l)$, the rate from B_k to B_l in the quotient chain, is $q(x_i, B_l) = \sum_{x_j \in B_l} Q(x_i, x_j)$, where x_i is an

SPLIT(S, P, L)

1 $L', L'' := \emptyset$
2 for every $x_j \in S$
3 $\quad$ for every $x_i \to x_j$
4 $\quad\quad x_i$.sum $:= 0$
5 for every $x_j \in S$
6 $\quad$ for every $x_i \to x_j$
7 $\quad\quad x_i$.sum $:= x_i$.sum $+ Q(x_i, x_j)$
8 $\quad\quad L' := L' \cup \{x_i\}$
9 for each $x_i \in L'$
10 $\quad B :=$ block of x_i
11 $\quad$ delete x_i from B
12 $\quad$ INSERT(B_T, x_i)
13 $\quad$ if $B \notin L''$ add B to L''
14 for every $B \in L''$
15 $\quad B_l :=$ largest block of $\{B, V_{k_1}, \ldots, V_{k_{|B_T|}}\}$
16 $\quad L := L \cup \{B, V_{k_1}, \ldots, V_{k_{|B_T|}}\} - \{B_l\}$

Algorithm 2. Pseudocode of SPLIT procedure.

Where to go from here I

Options:

1. Your proposals
2. Work out an overview of frameworks for (stochastic) dynamical systems and methods for minimization (in brief)
3. Explore the space of possible bisimulations and how to construct them; [5] and [4] describe them in abstract/general form (?)
4. Carry over the bisimulation/Schützenberger's construction approach to ODEs/CRNs as Tribastone et al. did for partition refinement/MC lumping (?)
5. Or other topic

Where to go from here II

What's your big picture/frame of reference/context?

Where is this going?

How/why is it important/significant?
Am I creating something of value in the thesis (for you)?

I don't feel comfortable/feel frustrated as I do not know where Kiefer $\leftrightarrow$ Valmari/Tribastone is heading

Is it enough for a thesis?

What's your view on that?

Bibliography


L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin, “Maximal aggregation of polynomial dynamical systems,” Proceedings of the National Academy of Sciences of the United States of America, vol. 114 38, pp. 10 029–10 034, 2017.


S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, “On the complexity of equivalence and minimisation for q-weighted automata,” Logical Methods in Computer Science, vol. 9, 2013.


A. Valmari and G. Franceschinis, “Simple o (m logn) time markov chain lumping,” in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2010, pp. 38–52.


M. P. Schützenberger, “On the definition of a family of automata,” Information and control, vol. 4, no. 2-3, pp. 245–270, 1961.


P. Buchholz, “Bisimulation relations for weighted automata,” Theoretical Computer Science, vol. 393, no. 1-3, pp. 109–123, 2008.


S. Kiefer and B. Wachter, “Stability and complexity of minimising probabilistic automata,” in International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 268–279.