Online Companion to the Paper 'Model Checking Stochastic Automata for Dependability and Performance Measures'

Peter Buchholz, Jan Kriege, and Dimitri Scheftelowitsch

Department of Computer Science, TU Dortmund
{peter.buchholz,jan.kriege,dimitri.scheftelowitsch}@udo.edu

Abstract. This note includes some additional details as an extension of the paper "Model Checking Stochastic Automata for Dependability and Performance Measures" [4]. In particular, it introduces the Kronecker operations, which are the base of the approach, it presents some basic ideas to prove the steps of the model checking approach, and it introduces some algorithmic details. It should be noted that the note is a supplement of [4] and is not intended to be self-explanatory.

1 Stochastic Automata and the underlying Stochastic Process

The compositional generation of matrices describing automata transitions is based on Kronecker operations which are introduced first. For further details we refer to the literature [6, 7]. Let $\boldsymbol{A} \in \mathbb{R}^{r_a,c_a}$ and $\boldsymbol{B} \in \mathbb{R}^{r_b,c_b}$, then

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} \boldsymbol{A}(1,1)\boldsymbol{B} & \cdots & \boldsymbol{A}(1,c_A)\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}(r_A,1)\boldsymbol{B} & \cdots & \boldsymbol{A}(r_A,c_A)\boldsymbol{B} \end{pmatrix} \tag{1}$$

is the Kronecker product. The Kronecker sum of two square matrices $\boldsymbol{A} \in \mathbb{R}^{r_a,r_a}$ and $\boldsymbol{B} \in \mathbb{R}^{r_b,r_b}$is defined as

$$\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{A} \otimes \boldsymbol{I}_{r_b} + \boldsymbol{I}_{r_a} \otimes \boldsymbol{B} \tag{2}$$

where $\boldsymbol{I}_r$ is the identity matrix or order r. There are some fundamental rules that are useful if Kronecker operations are applied to describe the composition of stochastic processes, Markovian or not.

- If $\mathcal{S}^{(a)}$ and $\mathcal{S}^{(b)}$ are state spaces of order n and m, respectively, then the joint state space $\mathcal{S}^{(a)} \times \mathcal{S}^{(b)}$ includes nm states and states are described by tuples (i, j) with $i \in \mathcal{S}^{(a)}$ and $j \in \mathcal{S}^{(b)}$. If $\pi^{(a)} \in \mathbb{R}^n$ and $\pi^{(b)} \in \mathbb{R}^m$ are vectors including weights of the states, then
  - the composition of the weights such that the weight of state (i, j) in the joint state space is given by the sum of weights of the constituent states can be described by $\pi^{(a)} \oplus \pi^{(b)}$;
  - the composition of the weights such that the weight of state (i, j) in the joint state space is given by the product of weights of the constituent states can be described by $\pi^{(a)} \otimes \pi^{(b)}$.

– If $\boldsymbol{A}^{(a)} \in \mathbb{R}^{r_a,r_a}$ and $\boldsymbol{A}^{(b)} \in \mathbb{R}^{r_b,r_b}$ are transition matrices on $\mathcal{S}^{(a)}$ and $\mathcal{S}^{(b)}$, respectively, then a transition in the joint state space $\mathcal{S}^{(a)} \times \mathcal{S}^{(b)}$ is given by a transition from (i, j) into (k, l). The weights can be described in a matrix $\boldsymbol{B} \in \mathbb{R}^{r_ar_b,r_ar_b}$ which can be computed as follows:
  - If the transition weight is given by $\boldsymbol{B}((i, j), (k, l)) = \boldsymbol{A}^{(a)}(i, k)\boldsymbol{A}^{(b)}(j, l)$ (i.e., a joint transition in both processes), then $\boldsymbol{B} = \boldsymbol{A}^{(a)} \otimes \boldsymbol{A}^{(b)}$. This composition can be easily extended to non-square matrices describing transitions between different state spaces.
  - If transition occur independently and not at the same time, then a transition from (i, j) into (k, l) is only possible if $i = k$ or $j = l$). In this case $\boldsymbol{B} = \boldsymbol{A}^{(a)} \oplus \boldsymbol{A}^{(b)}$.

With these rules, Kronecker operations can be used to generate matrices of Markov processes. The advantage of a Kronecker representation is that it can be exploited in computational algorithms, i.e. several matrix operations can be realized using the small matrices in the Kronecker operations rather than the large matrix which is built. This allows a very memory efficient implementation which sometimes results in a more efficient computation, for details we refer to the literature [3, 5–7].

The rules are applied to build the transition matrix of SAs (see Eqs. 11 and 12 in [4]). We now briefly derive the matrices.

In general, a SA is specified by the automaton and the clock processes. A clock process is characterized by a set of matrices $\left(\pi_0^{(c)}, \mathbf{G}_0^{(c)}, \mathbf{G}_1^{(c)}, \ldots, \mathbf{G}_{K_c}^{(c)}\right)$ that specify an MMAP or MRAP or a distribution if $\mathbf{G}_k^{(c)} = \mathbf{g}_k\pi_0^{(c)}$ (i.e. is given by the product of two vectors). Informally, the behavior of a SA can be described by the following two rules:

1. In a location I clocks from the set $ena(I)$ run independently and in parallel.
2. If in a location a clock generates an event, then this triggers a transition in the automaton and according to this transition clocks are enabled, disabled or reset.

Now we can apply the basic rules given above.

– In location I the state of clocks from set $act(I)$ is relevant. Thus the joint state can be described by a vector with $|act(I)|$ entries. If $\pi^{c)}$ is the weight vector for clock c and the weight of state $(i_1, \ldots, i_C)$ is given by $\prod_{c=1}^{C} \pi^{(c)}(i_c)$, then indices can be linearized and the joint weight vector is given by $\otimes_{c=1}^{C}\pi^{(c)}$.
– Since inside location I clocks from the set $act(I)$ run independently, the joint transitions are characterized by the Kronecker sum of the matrix $\mathbf{G}_0^{(c)}$ for $c \in ena(I)$. Additionally, clocks may be suspended. This means that they keep their state which is characterized by a transition matrix $\mathbf{0}$ of order $n^{(c)}$. Together this defines the diagonal blocks $\boldsymbol{H}_{I,I}^{D}$.
– Transitions in the automaton are triggered by clocks that generate events. If in location I clock $c' \in ena(I)$ generates an event of class k' and the new location is J (i.e., $trans(I, (c', k'), J) > 0$), then the state of clock c' changes due to the transition. The change of the automata state implies that immediately the state of other clocks may be modified. This means that the states changes synchronously and the Kronecker product has to be used to combine the matrices. The matrices in the Kronecker product describe the state changes of the clocks. We have to distinguish between

clock c' that triggers the transition and the remaining clocks $c \in C \setminus \{c'\}$. For clock c' the following three cases have to be distinguished:

1. If clock c' remains active (i.e., enabled or suspended) in the new location and is not reset, then a normal class k' transition occurs as described by matrix $\mathbf{G}_{k'}^{(c')}$.
2. If clock c' becomes inactive in the new location, then only the transition rates or weights are needed, such that a vector $\mathbf{G}_{k'}^{(c')}\mathbb{1}$ described the transition.
3. If clock c' is reset in the new location, the new state vector will be $\pi_0^{(c')}$ and the transition is characterized by the matrix $\mathbf{G}_{k'}^{(c')}\mathbb{1}\pi_0^{(c')}$.

For clocks $c \in C \setminus \{c'\}$ four cases have to be distinguished.

1. If clock $c \in act(I)$ remains active, then its state remains as it is and the transition can be characterized by an identity matrix $\boldsymbol{I}$ of order $n^{(c)}$.
2. If clock c is not active in the new location J, its state is no longer needed which is described by the column vector $\mathbb{1}$.
3. If clock c is reset, then the corresponding matrix equals $\mathbb{1}\pi_0^{(c)}$.
4. If $c \notin act(I)$ but $c \in act(J)$, then it has to be newly initialized and the state vector has to be considered in the state vector for location J which is described by vector $\pi_0^{(c)}$.
5. If clock c is not active in both locations I and J, then the state change can be described by the matrix (1), the neutral element of the Kronecker product, that does not modify the expression.

The different cases are considered in Eq. 10 in [4]. Kronecker products of this form build the blocks of matrix $\boldsymbol{H}^N$.

The matrix $\mathbf{H}$ and the initial vector π_0 specify a CTMC for SAMs and a rational process (RP) [2, 5] for general SAs. As already mentioned in [4], transitions in $\mathbf{H}^N$ correspond to transitions in the automaton where $\mathbf{H}^D$ contains rates or weights of local transitions and the diagonal entries. In this way $(\pi_0, \mathbf{H}^D, \mathbf{H}^N)$ can be interpreted as an MMAP/MRAP.

2 Computation of State Vectors

We have to distinguish between a state vector π_t at time t and the expected state vector $\boldsymbol{p}_t = E[\pi_t]$ at time t. The state vector is a concrete state (i.e., all non-zero entries belong to one location) that is observed at time t. For a given sequence

$$\mathcal{F}_{(I_0,\pi)} = ((I_0, (c_1, k_1), t_1), (I_1, (c_2, k_2), t_2), \ldots, (I_{F-1}, (c_F, k_F), t_F), I_F)$$

that starts in (I_0, π_0), the state can be computed as shown below. Since $\mathcal{F}_{(I_0,\pi)}$ has a density with which it can be observed, a distribution over the possible weight vectors can be defined for each $t \geq 0$. Consequently, the expectation $\boldsymbol{p}_t$ can be computed. However, $\boldsymbol{p}_t$ defines a distribution or weight vector over all/several locations.

To compute the expectation at time t $\boldsymbol{p}_t$, the differential equation

$$\boldsymbol{p}_t = \pi_0 e^{t\mathbf{H}} = \pi_0 \sum_{h=1}^{\infty} \frac{(t\mathbf{H})^h}{h!}$$

has to be solved. In general the system can be solved by a standard solver for differential equations, like a Runge Kutta method. For SAMs and the resulting CTMCs uniformization [7] can be used. Let $\mathbf{H}$ be the generator matrix of a CTMC and $\alpha \geq \max_{i \in \mathcal{M}}(|\mathbf{H}(i,i)|)$. Then $\mathbf{U} = \mathbf{H}/\alpha + \mathbf{I}$ is a stochastic matrix and

$$\boldsymbol{p}_t = \pi_0 \sum_{h=1}^{\infty} e^{-\alpha t} \frac{(\alpha t)^h}{h!} (\mathbf{U})^h \ .$$

This computation is more stable and allows one to truncate the sum after a finite number of steps with a predefined error bound. The stationary vector can be computed for irreducible systems as the solution of a system of linear equations using standard means. It should be noted that stationary and transient analysis can exploit the Kronecker representation of $\mathbf{H}$. For details see [1, 2].

For the computation of the state π that is reached after observing $\mathcal{F}_{(I_0,\pi)}$ starting in state (I_0, π) we first compute (see Eq. 13 in [4])

$$Dens(\mathcal{F}_{(I_0,\pi_0)}) = \pi_0 \left(\prod_{f=1}^{F} e^{t_f \bigoplus_{c \in act(I_{f-1})} \boldsymbol{X}^{(c)}_{I_{f-1}}} \bigotimes_{c \in act(I) \cup act(J)} \boldsymbol{Y}^{(c)}_{I_{f-1},I_f,(c_f,k_f)} \right) \mathbb{I} \tag{3}$$

where the matrices $\boldsymbol{X}^{(c)}_I$ and $\boldsymbol{Y}^{(c)}_{I,J,(c',k')}$ are defined in [4]. For $Dens(\mathcal{F}_{(I_0,\pi_0)}) = 0$, the sequence $\mathcal{F}_{(I_0,\pi)}$ cannot be observed, otherwise

$$\pi = \left(Dens(\mathcal{F}_{(I_0,\pi_0)})\right)^{-1} \pi_0 \left(\prod_{f=1}^{F} e^{t_f \bigoplus_{c \in act(I_{f-1})} \boldsymbol{X}^{(c)}_{I_{f-1}}} \bigotimes_{c \in act(I) \cup act(J)} \boldsymbol{Y}^{(c)}_{I_{f-1},I_f,(c_f,k_f)} \right) \tag{4}$$

and (I_F, π) is the state of the system after observing the sequence. Again Eq. 4 can be computed using a standard differential equation solver or for SAMs uniformization.

3 Compositional Model Checking

If vector $\pi_0 = \otimes_{c \in act(I_0)} \pi^{(c)}_0$, then Eq. 4 can be decomposed. The decomposition is based on the following relation.

$$e^{\oplus_{c \in C} \boldsymbol{X}^{(c)}_{I_c}} = \otimes_{c \in C} e^{\boldsymbol{X}^{(c)}_{I_c}} \tag{5}$$

This implies that instead of a matrix exponential of order $\prod_{c \in C} n^{(n)}$, $|C|$ matrix of exponentials of order $n^{(c)}$ can be computed. The proof of (5) is based on the relation

$$\begin{aligned} e^{\mathbf{G}^{(1)} \oplus \mathbf{G}^{(2)} t} &= \sum_{i=0}^{\infty} \frac{((\mathbf{G}^{(1)} \oplus \mathbf{G}^{(2)})t)^i}{i!} = \sum_{i=0}^{\infty} \frac{((\mathbf{G}^{(1)} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{G}^{(2)})t)^i}{i!} = \\ \sum_{i=0}^{\infty} \frac{1}{i!} \sum_{j=0}^{i} \binom{i}{j} (\mathbf{G}^{(1)} t)^j \otimes (\mathbf{G}^{(2)} t)^{i-j} &= \sum_{i=0}^{\infty} \frac{(\mathbf{G}^{(1)} t)^i}{i!} \otimes \sum_{j=0}^{\infty} \frac{(\mathbf{G}^{(2)} t)^j}{j!} = e^{\mathbf{G}^{(1)} t} \otimes e^{\mathbf{G}^{(2)} t} \end{aligned}$$

(5) can be exploited to evaluate (4). If vector π_0 is only available as a full vector, then the matrix exponentials $e^{t_f \oplus_{c \in act(I_{f-1})} \boldsymbol{X}^{(c)}_{I_{f-1}}}$ can be represented as $\otimes_{c \in act(I_{f-1})} e^{t_f \boldsymbol{X}^{(c)}_{I_{f-1}}}$. Even if

the matrices in the Kronecker product may become full matrices, it is usually recommended to compute these matrices which are of order $n^{(c)}$. If $\pi_0 = \otimes_{c \in C} \pi_0^{(c)}$, then one can compute

$$\pi^{(c)} = (Dens(\mathcal{F}_{(I_0,\pi_0)}))^{-1} \pi_0 \left(\prod_{f=1}^{F} e^{t_f \boldsymbol{X}^{(c)}_{I_{f-1}}} \boldsymbol{Y}^{(c)}_{I_{f-1},I_f,(c_f,k_f)} \right) \tag{6}$$

for every $c \in C$ where $\pi^{(c)} = (1)$, $\boldsymbol{X}_I^{(c)} = (0)$ for $c \notin act(I)$ and $\boldsymbol{Y}^{(c)}_{I,J,(c',k')} = (1)$ for $c \in C \setminus \{act(I) \cup act(J)\}$. Then

$$\pi = \otimes_{c \in C} \pi^{(c)}.$$

If clock c is an MMAP, then vector $\pi^{(c)}$ can be computed using uniformization otherwise a general differential equation solver has to be applied.

The decompositional approach can be directly applied for the evaluation of Eqs. 19 and 21 in [4].

4 Some Detailed Examples

We consider the two running examples in some more detail.

4.1 First Running Example

The first running example is shown in Figure 1. It describes a simple model containing two processes that iterate between a local computation phase and an access to a shared resource. Process 1 has preemptive priority over process 2 when accessing the shared resource. The preemption is of the type *preemptive resume* which means that the second process starts from that point where its has been interrupted. Furthermore, the sequence of local computation steps of the first process are correlated, all other times are independently and identically distributed.

The automaton has 5 location with the following interpretations:

1. Both processes are computing locally.
2. Process 1 has access to the shared resource and process 2 is computing locally.
3. Process 2 has access to the shared resource and process 1 is computing locally.
4. Process 1 has access to the shared resource and process 2 is waiting for access.
5. Process 1 has access to the shared resource and process 2 has been interrupted at the shared resource.

The events are driven by 4 clocks with the following interpretation:

$c1$ The clock triggers the local computation times of process 1. It is never disabled, which means that it keeps its state even if not enabled. The clock is described by a MAP or RAP.

$c2$ The clock triggers the local computations of process 2, it is either enabled or disabled, but never suspended. The clock is described by a PHD or MED.

$c3$ The clock triggers the access times to the shared resource of the first process. It is never suspended. The clock is described by a PHD or MED.

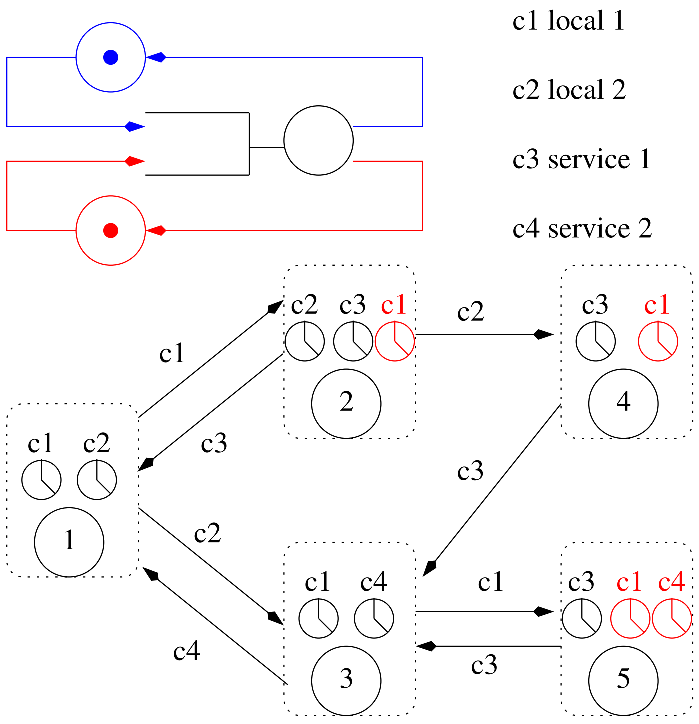

Fig. 1. Simple system with two processes and a shared resource.

*c*4 The clock triggers the access times of the second process to the shared resource. It is suspended, if the second process is preempted during its access to the shared resource. It is not enabled when the second process is computing locally. The clock is also described by a PHD or MED.

The following transitions describe the dynamics of the model. Transitions are triggered by clocks, i.e., if a clock elapses in a location, then the corresponding transition occurs. In this example, successor states are defined by the source location and the clock, i.e., there is always a unique successor location.

a) Clock $c1$ elapses in location 1 and causes a transition to location 2. This means that process 1 finishes local computation and accesses the shared resource. Since local computation times of process 1 are possibly correlated, the state of $c1$ is kept which means that the clock is suspended. Clock $c2$ remains enabled and keeps its state. Clock $c3$ triggering the access time of process 1 to the shared resource is newly enabled.

b) Clock $c2$ elapses in location 1 and causes a transition into location 3. This means that process 2 finishes its local computations and gets access to the shared resource.

Clock $c1$ remains enabled and keeps its state since process 1 continues with local computations. Furthermore, clock $c4$, triggering the access to the shared resource of process 2, is newly enabled. Clock $c2$ is disabled since local computation times of process 2 are independently and identically distributed such that the state of the clock is not kept during phases where process 2 is not computing locally.

c) Clock $c2$ elapses in location 2. The second process ends its local computations and tries to access the shared resource that is held by process 1. Since process 1 has priority at the shared resource, process 2 has to wait. Clock $c2$ is disabled since the local computation has ended and clock $c4$ is disabled since the access time to the shared resource by the second process has not yet started.

d) Clock $c3$ elapses in location 2. In this case, process 1 ends its access to the shared resource and continues its local computation with the state where it finished before. The system is again in location 1.

e) Clock $c1$ elapses in location 3. This implies that the first process ends local computation and gets access to the shared resource. Since process 1 has preemptive priority at the shared resource, access is gained. Since the priority is preemptive resume, clock $c4$, triggering the access of the second process to the shared resource, keeps its state.

f) Clock $c4$ elapses in location 3. Process 2 ends its access to the shared resource and starts local computation. Clock $c2$, triggering the local computation of process 2, becomes enabled.

g) Clock $c3$ elapses in location 4. The first process finishes its access to the shared resource and starts local computation with the suspended state of clock $c1$. Since process 2 waits for access to the shared resource, it starts the access by initializing clock $c4$ which is enabled in the new location 3.

h) Clock $c3$ elapses in location 5. This case is similar to the previous one. However, in contrast to the previous case, process 2 had been interrupted during its access to the shared resource and starts from the state where is has been interrupted which is stored for the suspended clock $c4$.

4.2 Second Running Example

The second running example from [4] describes a system with one processor and two hard drives which are united into one RAID unit shown in Figure 2. The system is down if either the processor or both disk drives fail. Additionally, the system can be under maintenance. The times are modeled by seven clock processes: Failure times for the hard drives are distributed according to the clock processes c_1 and c_2, respectively. Failure and maintenance times for the processor are modeled by the process p that has two classes: Class a indicates a failure, class b indicates a maintenance. Note, that the system can be under maintenance only if both disk drives are working, otherwise it requires the more time consuming repair. The repair times for disks are modeled by the clock processes r_1 and r_2. If the system is down the repair time is modeled by r_d. The maintenance time is modeled by r_m. As described in [4] the process p is modeled by an PHD with two classes, c_1 and c_2 are modeled by ME distributions and the processes $r_.$ are described by Erlang distributions. In the following we will give MRAP descriptions for all clock processes that are equivalent to the mentioned distributions and then show

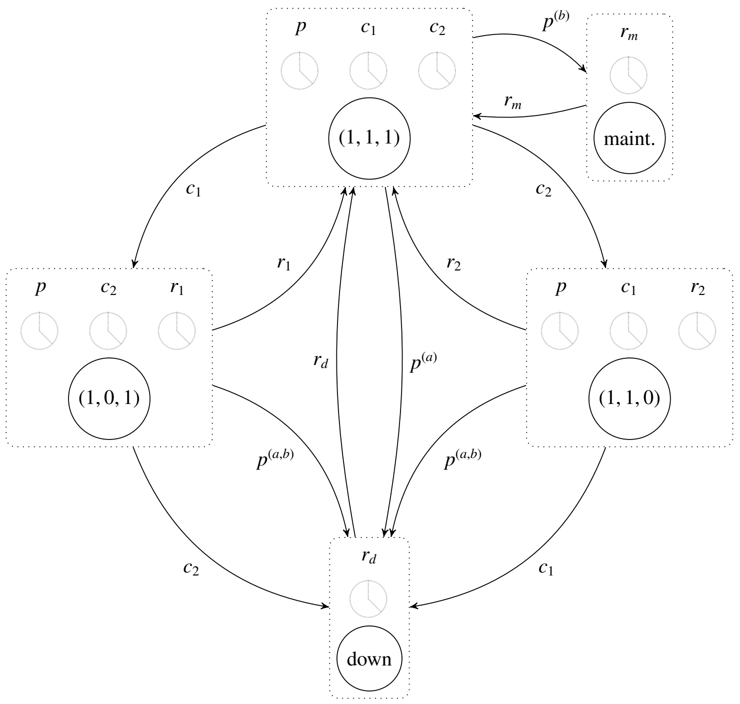

Fig. 2. A system with two hard drives and a processor.

how matrices $\mathbf{H}^D$ and $\mathbf{H}^N$ can be constructed that describe the complete automaton as an MRAP.

In particular, the clock processes are modeled by the following MRAPs:

– Clock processes c_1 and c_2:

$$\pi_0^{(c_1)} = \pi_0^{(c_2)} = (0.2, 0.3, 0.5)$$

$$\mathbf{G}_0^{(c_1)} = \mathbf{G}_0^{(c_2)} = \begin{pmatrix} -0.1 & 0.0 & 0.0 \\ 0.0 & -0.3 & 0.2 \\ 0.0 & -0.2 & -0.3 \end{pmatrix} \qquad \mathbf{G}_1^{(c_1)} = \mathbf{G}_1^{(c_2)} = \begin{pmatrix} 0.02 & 0.03 & 0.05 \\ 0.02 & 0.03 & 0.05 \\ 0.10 & 0.15 & 0.25 \end{pmatrix}$$

– Clock process p:

$$\pi_0^{(p)} = (0.6,\ 0.4,\ 0.0)$$

$$\mathbf{G}_0^{(p)} = \begin{pmatrix} -2.0 & 0.0 & 0.0 \\ 0.0 & -3.0 & 1.0 \\ 0.0 & 1.0 & -4.0 \end{pmatrix} \qquad \mathbf{G}_1^{(p)} = \begin{pmatrix} 0.2 & 0.4 & 0.0 \\ 0.6 & 0.1 & 0.0 \\ 0.8 & 0.7 & 0.0 \end{pmatrix} \qquad \mathbf{G}_2^{(p)} = \begin{pmatrix} 1.0 & 0.4 & 0.0 \\ 0.6 & 0.7 & 0.0 \\ 1.0 & 0.5 & 0.0 \end{pmatrix}$$

– Clock processes r_1 and r_2:

$$\pi_0^{(r_1)} = \pi_0^{(r_2)} = (1,0,0,0,0)$$

$$\mathbf{G}_0^{(r_1)} = \mathbf{G}_0^{(r_2)} = \begin{pmatrix} -100 & 100 & 0 & 0 & 0 \\ 0 & -100 & 100 & 0 & 0 \\ 0 & 0 & -100 & 100 & 0 \\ 0 & 0 & 0 & -100 & 100 \\ 0 & 0 & 0 & 0 & -100 \end{pmatrix} \qquad \mathbf{G}_1^{(r_1)} = \mathbf{G}_1^{(r_2)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 100 & 0 & 0 & 0 & 0 \end{pmatrix}$$

– Clock process r_d:

$$\pi_0^{(r_d)} = (1,0)$$

$$\mathbf{G}_0^{(r_d)} = \begin{pmatrix} -6 & 6 \\ 0 & -6 \end{pmatrix} \qquad \mathbf{G}_1^{(r_d)} = \begin{pmatrix} 0 & 0 \\ 6 & 0 \end{pmatrix}$$

– Clock process r_m:

$$\pi_0^{(r_m)} = (1,0)$$

$$\mathbf{G}_0^{(r_m)} = \begin{pmatrix} -20 & 20 \\ 0 & -20 \end{pmatrix} \qquad \mathbf{G}_1^{(r_m)} = \begin{pmatrix} 0 & 0 \\ 20 & 0 \end{pmatrix}$$

$\mathbf{H}^D$ and $\mathbf{H}^N$ are composed of submatrices $\mathbf{H}_{I,J}$ that are constructed according to Eqs. 11 and 12 from [4]. For our example we have

$$\mathbf{H}^D = \begin{pmatrix} \mathbf{H}^D_{(1,1,1),(1,1,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}^D_{(1,0,1),(1,0,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{H}^D_{(1,1,0),(1,1,0)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}^D_{maint,maint} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}^D_{down,down} \end{pmatrix}$$

and

$$\mathbf{H}^N = \begin{pmatrix} \mathbf{0} & \mathbf{H}^N_{(1,1,1),(1,0,1)} & \mathbf{H}^N_{(1,1,1),(1,1,0)} & \mathbf{H}^N_{(1,1,1),maint} & \mathbf{H}^N_{(1,1,1),down} \\ \mathbf{H}^N_{(1,0,1),(1,1,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}^N_{(1,0,1),down} \\ \mathbf{H}^N_{(1,1,0),(1,1,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}^N_{(1,1,0),down} \\ \mathbf{H}^N_{maint,(1,1,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{H}^N_{down,(1,1,1)} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

According to [4, Eq. 11] the matrices $\mathbf{H}^D_{I,I}$ are given as the Kronecker sum of the matrices $\mathbf{G}_0^{(c)}$ of all enabled clock processes in location I. In location $(1,1,1)$ the processes c_1, c_2 and p are enabled and consequently we get the 27×27 matrix

$$\mathbf{H}^D_{(1,1,1),(1,1,1)} = \mathbf{G}_0^{(c_1)} \oplus \mathbf{G}_0^{(c_2)} \oplus \mathbf{G}_0^{p}.$$

In a similar way we obtain

$$\begin{aligned} \mathbf{H}^D_{(1,0,1),(1,0,1)} &= \mathbf{G}_0^{(r_1)} \oplus \mathbf{G}_0^{(c_2)} \oplus \mathbf{G}_0^{p} \\ \mathbf{H}^D_{(1,1,0),(1,1,0)} &= \mathbf{G}_0^{(r_2)} \oplus \mathbf{G}_0^{(c_1)} \oplus \mathbf{G}_0^{p} \\ \mathbf{H}^D_{maint,maint} &= \mathbf{G}_0^{(r_m)} \\ \mathbf{H}^D_{down,down} &= \mathbf{G}_0^{(r_d)}. \end{aligned}$$

The matrices $\mathbf{H}^N_{I,J}$ are computed as the Kronecker product of clock processes that are active in locations I and J according to [4, Eq. 11]. Depending on whether the clock process remains active, becomes active or becomes inactive one of the cases from [4, Eq. 10] applies. As an example we will consider the submatrix $\mathbf{H}^N_{(1,1,1),(1,0,1)}$. We have to treat the clock processes r_1, c_1, c_2 and p that are active in at least one of the two locations.

- Clock process r_1 becomes active in location $(1,0,1)$ (but was not active in $(1,1,1)$). This is case (f) from [4, Eq. 10] and thus, we use $\pi_0^{(r_1)}$ in the Kronecker product.
- Clock process c_1 triggered the transition and becomes inactive (case (b) from [4, Eq. 10]. Hence, $\mathbf{G}_1^{(c_1)}\mathbf{1}$ is used.
- Clock processes c_2 and p remain active but did not trigger the transition and are not reset (case (d) from [4, Eq. 10]). Both are represented by an identity matrix of the corresponding order.

In summary, we get

$$\mathbf{H}^N_{(1,1,1),(1,0,1)} = \pi_0^{(r_1)} \otimes (\mathbf{G}_1^{(c_1)}\mathbf{1}) \otimes \boldsymbol{I} \otimes \boldsymbol{I}$$

resulting in a 27×45 matrix. The remaining matrices $\mathbf{H}^N_{I,J}$ are constructed in a similar way.

References

1. P. Buchholz. Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics*, 31(4):375–404, 1999.
2. P. Buchholz. Numerical analysis of rational processes beyond Markov chains. *Perform. Eval.*, 70(9):646–662, 2013.
3. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations and sparse matrices with applications to the solution of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.
4. P. Buchholz, J. Kriege, and D. Scheftelowitsch. Model checking stochastic automata for dependability and performance measures. In *DSN*, 2014 (to appear).
5. P. Buchholz and M. Telek. Rational automata networks: A non-Markovian modeling approach. *INFORMS J. on Computing*, 25(1):87–101, 2013.
6. C. F. Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(12):85 – 100, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.
7. W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.