Compositional CSL model checking for Boucherie product processes

(abstract of ongoing work)

P. Ballarini[1,2] and J. Hillston[2]

[1] Dipartimento di Informatica, Università di Torino, Italy,
ballarin@di.unito.it
[2] Laboratory for Foundations of Computer Science, University of Edinburgh, UK
{a1pballa,jeh}@inf.ed.ac.uk

Abstract. In the framework of system modeling, compositionality is a well known means to tackle the so-called state space explosion problem. The basic idea is to develop techniques which allow the analysis of a system's model in terms of a number of submodels obtained through decomposition of the whole system. In [2] Boucherie provides a method to compose two ergodic Continuous Time Markov Chains (CTMCs), the result of which he proves to have a product form solution (i.e. the steady state distribution of the composed CTMC is given by the "normalised" product of the steady state distributions of the components). In this work we investigate the possibility for a decompositional approach to model check formulae of the Continuous Stochastic Logic (CSL) on a Boucherie CTMC. We aim to show that for any given CSL formula ϕ which we want to check on the product process, it is possible to derive a set of *transformed* formulae, which are "component equivalent" to the original one (i.e. ϕ is satisfied in the product process if and only if the *transformed* formulae are satisfied in the components they refer to). So far, we have managed to achieve results for a restricted version of the CSL syntax: we have proved that compositional model checking over Boucherie product process is possible for any CSL state formulae except the ones obtained by application of the probabilistic operator ($\mathcal{P}_{\triangleleft p}(\phi)$).

1 Introduction and background

Generally speaking formal methods for systems analysis aim to provide means which allow one to model and study systems. Existing methodologies can be classified according to the system's characteristics (deterministic, non-deterministic, stochastic...) and the type of analysis (correcteness, validation, evaluation) one is interested to consider/perform.

Model checking of temporal logics like Linear Time-temporal Logic (LTL) [6] and Computation Tree Logic (CTL) [4], allows for testing of (temporal) qualitative properties over non-probabilistic systems (i.e. systems whose behaviour characterisation does not depend on a probability distribution). However, often, the implicit non-determinism of a system can be characterised only in terms of a probability distribution (such systems are referred as *stochastic systems* or *stochastic processes*).

Markov processes, are a relevant type of *stochastic processes*, which allow for both steady-state and transient-state analysis. The Probabilistic real-time Computation Tree Logic (PCTL) [5] provides means for both validation and evaluation of systems which can be modelled as a Discrete Time Markov Chain (DTMC).

On the other hand, CSL [1, 3] (a temporal logic based indeed on both CTL and PCTL), enriches the CTMC standard analysis of long run characteristics (steady-state and transient-state analysis), by adding the possibility of specification and verification of path measures.

1.1 The CSL syntax

In [3] Baier, Katoen and Hermanns have improved the (expressiveness of the) original CSL ([1]) by adding a novel steady-state operator ($\mathcal{S}$) plus a time-bounded version of the until operator ($\mathcal{U}^{\leq t}$). The resulting grammar of CSL formulae is:

$$\Phi ::= tt \;\mid\; a \;\mid\; \phi' \wedge \phi'' \;\mid\; \neg\phi' \;\mid\; \mathcal{S}_{\lhd p}(\phi') \;\mid\; \mathcal{P}_{\lhd p}(\varphi) \tag{1}$$

$$\varphi ::= \mathcal{X}(\phi) \;\mid\; \phi' \;\mathcal{U}\; \phi'' \;\mid\; \phi' \;\mathcal{U}^{\leq t}\; \phi'' \tag{2}$$

where Φ denotes the state-formulae, φ denotes the path-formulae and $a \in AP$ is a generic atomic proposition, $p \in [0, 1]$ and $\lhd \in \{\leq, <, \geq, >\}$.

1.2 The Boucherie product process

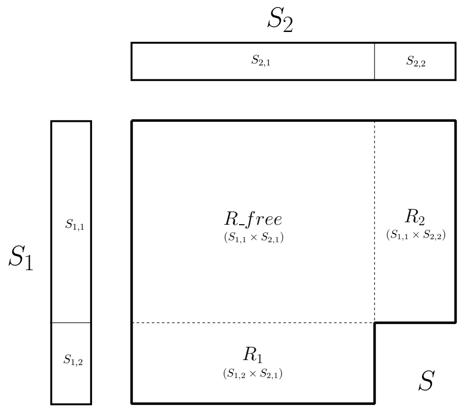

Fig. 1. Boucherie product process's state-space

In [2], Boucherie establishes a form of CTMC which is susceptible to product form solution. The multi-dimensional CTMC models a number of processes which compete over shared resources. When one process holds a resource, other processes cannot access the resource; furthermore they are subject to strong blocking conditions which means that they cannot evolve until the resource is released. In the simplest case, which is what we consider here, such a CTMC $\mathcal{M}$ consists of two sub-processes, $\mathcal{M}_1$ and $\mathcal{M}_2$, and a notional resource R. The resource is not modelled explicitly but has the effect of partitioning the state space of each component process. If $\mathcal{M}_i$ $(i = \{1, 2\})$ has state space S_i, then $S_i = S_{i,1} \cup S_{i,2}$, where $S_{i,1}$ denotes the set of states in which the resource is not needed, while $S_{i,2}$ denotes the set of states in which the resource is used. When we compose the processes $\mathcal{M}_1$ and $\mathcal{M}_2$ to form $\mathcal{M}$ $(S = S_1 \times S_2)$ we naturally exclude the area $S_{1,2} \times S_{2,2}$ as this would denote simultaneous possession of the resource (see Fig. 1.).

More formally the framework can be described as follows: Let S_k and q_k be the state space and transition rates, respectively, of the kth Markov process in a collection, $1 \leq k \leq K$. Let $S = S_1 \times \cdots \times S_K$ denote the state space of the *product process*. It is assumed that *in each transition of the product process only the state in one dimension changes.* The transition rates of the product process in dimension k are then given by the transition rate of the individual process, q_k. In other words, the product process consists of K individual processes which do not directly interact in any way.

Competition between the processes over resources introduces an indirect form of interaction between them. Let us assume that there is a set I of *notional resources*. This set may also contain a non-resource, possession of which will indicate that a process is working without a resource (as above). Then the state space of each Markov process can be partitioned into states corresponding to resource use. Only one notional resource may be held at a time. Competition over resources is then defined as follows:

Definition 1 (Competition). *Let I be an index set. For each k, let $S_{k,i}, i \in I$, be a set of mutually exclusive sets such that $\emptyset \neq S_{k,i} \subset S_k$ and $\bigcup_{i \in I} S_{k,i} = S_k$, $k = 1, \ldots, K$. CTMC k uses resource i if the CTMC is in state $n_k \in S_{k,i}$. CTMCs k_1 and k_2 compete over resource i if $\{n_{k_1}, n_{k_2} : n_{k_1} \in S_{k_1,i}, n_{k_2} \in S_{k_2,i}\} = \emptyset$. Let $C_{ki} \subset \{1, \ldots, K\}$ be the CTMCs that compete over resource i with CTMC k.*

For any process k, $1 \leq k \leq K$, if the current state is in the subset $S_{k,i}$ it signifies that the process is presently using the resource i and no other process j, such that $j \in C_{ki}$, can gain access to i and enter its subset of states $S_{j,i}$. Thus the competition and the sets C_{ki} define areas of the state space of the product process which are inaccessible. The transition rates of the product process are defined in a way which ensures this exclusion.

Definition 2 (Product process). *The CTMC on state space $S = \prod_{k=1}^{K} S_k$, with transition rates*

$$q(n, n') = \sum_{k=1}^{K} q_k(n_k, n'_k) \prod_{\ell=1,\ell\neq k}^{K} \mathbf{1}(n_\ell = n'_\ell)\mathbf{1}(\textit{if } n_\ell \in A_{\ell i} \textit{ then } k \notin C_{\ell i})^{1}$$

where $n = (n_1, \ldots, n_K)$, $n' = (n'_1, \ldots, n'_K)$, is called the product process *of the collection of CTMCs $1, \ldots, K$, competing over resources I.*

Observe that, as required, these transition rates mean that in each transition only one process can change its state, and that process k cannot access resource i when it is being used by process ℓ if $k \in C_{\ell i}$, and vice versa. In the paper Boucherie shows that this mechanism of pair-wise relations imposed on the individual processes of a product process can be used to model various types of competition. Moreover he establishes the following result:

Theorem 1 (Product-form distribution)
The product process of the collection of CTMCs $1, \ldots K$ competing over resources I has equilibrium distribution π at S, defined as

$$S = \prod_{k=1}^{K} S_k \setminus \left(\prod_{k=1}^{K} \prod_{i\in I} \prod_{j\in C_{ki}} S_{k,i} \times S_{j,i} \right)$$

given by

$$\pi(n) = G \prod_{k=1}^{K} \pi_k(n_k) \qquad n \in S$$

where G is a normalising constant, determined by the exact form of S, and $\pi_k(\cdot)$ is the equilibrium distribution of process S_k.

The result holds because each process can either operate independently of the other processes or it is blocked. For all $n \in S$, if process ℓ is in state n_ℓ and $\ell \neq k$ then process k either carries out a transition which is not in competition with ℓ ($\mathbf{1}(\text{if } i : n_\ell \in S_{\ell,i} \text{ then } k \notin C_{\ell i}) = 1$) or process k wants to access the resource which ℓ occupies ($\mathbf{1}(\text{if } i : n_\ell \in S_{\ell,i} \text{ then } k \notin C_{\ell i}) = 0$). In either case process k will satisfy its own global balance equations: these equations are trivially satisfied when the process is stopped and also true when the process is operating independently. It appears that the exclusion principle maintained by the transition rates of the product process imposes a protocol on the behaviour of the product process that ensures that the CTMCs in the collection behave as if they are independent.

Here, as stated above, we currently consider only the simplest form of Boucherie process, in which there are two processes and two notional resources. There is

[1] $\mathbf{1}$ is an indicator function: the value of $\mathbf{1}(\textit{statement})$ is 1 if the statement is true and 0 otherwise.

no competition over the first resource, but the two processes compete over the second resource. As shown in Fig. 1. we will find it convenient to refer to areas of the state space in which the second resource is in use or not. In the area denoted $R_{free} = S_{1,1} \times S_{1,2}$ neither process is using the resource; in the area $R_1 = S_{1,2} \times S_{2,1}$, $\mathcal{M}_1$ is using the resource; whilst in area $R_2 = S_{1,1} \times S_{2,2}$, $\mathcal{M}_2$ is using the resource.

In the following we introduce a practical example of a two component Boucherie product process. This will be our running example, which we will exploit to show examples of decompositional model checking.

Example 1 (Boucherie product process of of a two components GIS) *Let us consider a navigational device consisting of a pair of sensors which maintain complementary data about geographical location. In order to keep the sensors' internal data structures in complementary states, they share data via a register. Each sensor gains access to the register and locks it while it reads the current data value; it then uses this information to adjust the equipment it controls while also recalculating a value for the shared register based on its own internal data structures. It then updates the value in the register and releases it. In addition, sensor 1 maintains an external monitor and will periodically gather data from this monitor and use it to recalculate its internal data structures.*

Each sensor consists of two components: one responsible for resetting the equipment during each cycle, and one responsible for carrying out the data recalculation. In sensor 1, the recalculation component is assumed to be also responsible for interaction with the monitor. The CTMCs representing the two sensors are shown in Figure 2 while Figure 3 shows the product process.

From the starting state s_{10} the first sensor either gathers data from the monitor at rate r_5 and enters state s_{11}, or it reads data (rate r_1) and enters state s_{12}. When it reads data it acquires the resource (the register). Once it has read the data one component recalculates (rate r_2) while the other resets (rate r_4). When both are ready the data is updated (rate r_3) and the register is released. When data has been gathered from the monitor a recalculation is necessary (rate r_2) before returning to the initial state. The state space of each CTMC can be partitioned into two subsets: $S_{11} = \{s_{10}, s_{11}\}$ (no resource held) and $S_{12} = \{s_{12}, s_{13}, s_{14}, s_{15}\}$, being the partition for $\mathcal{M}_1$ and $S_{21} = \{s_{20}\}$ and $S_{22} = \{s_{21}, s_{22}, s_{23}, s_{24}\}$, being the partition for $\mathcal{M}_2$.

Considering the sensors in isolation we can deduce that their equilibrium probability distributions are as follows:

$$
\begin{aligned}
\pi_1(s_{10}) &= r_2 r_3 r_4 (r_2 + r_4)/G_1 \\
\pi_1(s_{12}) &= r_1 r_2 r_3 r_4/G_1 \\
\pi_1(s_{13}) &= r_1 r_2^2 r_3/G_1 \\
\pi_1(s_{14}) &= r_1 r_3 r_4^2/G_1 \\
\pi_1(s_{15}) &= r_1 r_2 r_4 (r_2 + r_4)/G_1 \\
\pi_1(s_{11}) &= r_3 r_4 r_5 (r_2 + r_4)/G_1
\end{aligned}
$$

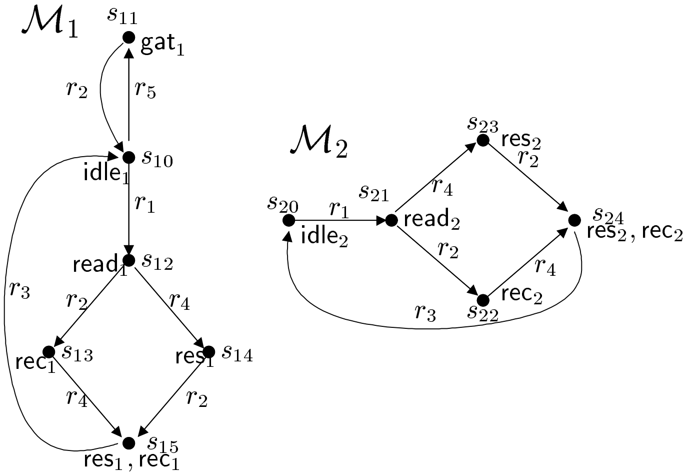

Fig. 2. State space of the two components $\mathcal{M}_1$ and $\mathcal{M}_2$,showing the states labeling

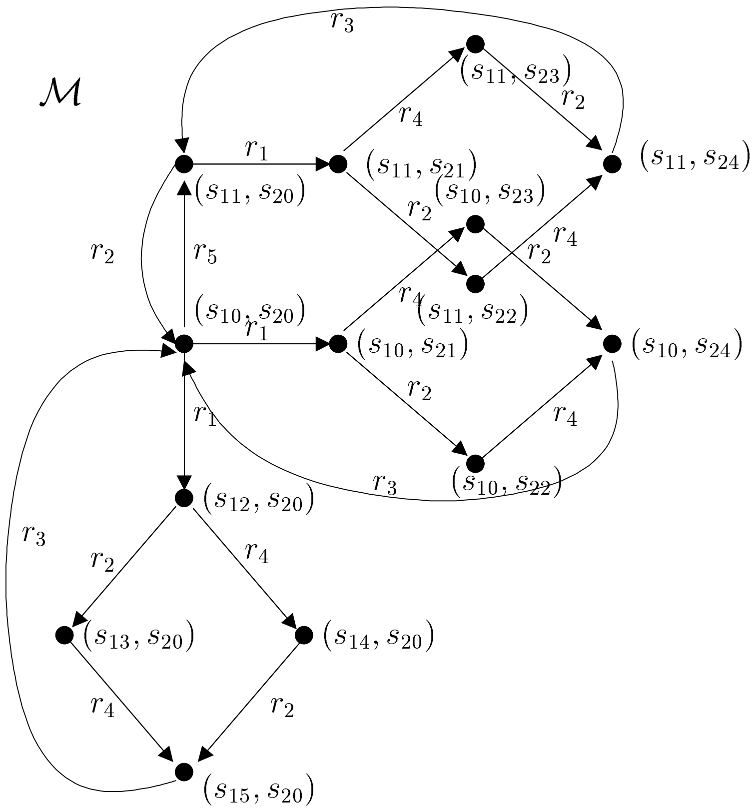

Fig. 3. State space of the product process $\mathcal{M}$

where $G_1 = (r_2 r_4 + r_4^2)(r_1 r_2 + r_1 r_3 + r_2 r_3 + r_3 r_5) + r_1 r_2^2 r_3$.

$$\begin{aligned}
\pi_2(s_{20}) &= r_3 r_6 r_7 (r_6 + r_7)/G_2 \\
\pi_2(s_{21}) &= r_1 r_3 r_6 r_7/G_2 \\
\pi_2(s_{22}) &= r_1 r_3 r_6^2/G_2 \\
\pi_2(s_{23}) &= r_1 r_3 r_7^2/G_2 \\
\pi_2(s_{24}) &= r_1 r_6 r_7 (r_6 + r_7)/G_2
\end{aligned}$$

where $G_2 = (r_6 r_7 + r_7^2)(r_1 r_3 + r_1 r_6 + r_3 r_6) + r_1 r_3 r_6^2$.

2 Compositional CSL model checking of a Boucherie product process

In this section we face the problem of defining a compositional semantics for the CSL formulae which refer to a Boucherie product process $\mathcal{M}$ with components $\mathcal{M}_1$ and $\mathcal{M}_2$.

Partitioning the Atomic Proposition set. As a first step, we partition the set of the atomic propositions AP according to the system (de)composition, which is:

$$AP = AP_1 \cup AP_2$$

where AP_i is the set of atomic proposition referring to component $\mathcal{M}_i$. As a result, the set Φ of CSL state-formulae defined over AP through (1), is also partitioned in three disjont subsets, namely:

$$\Phi = \Phi_1 \cup \Phi_2 \cup \Phi_{12}$$

We called *single component* formulae the formulae ϕ_i belonging to Φ_i (i.e. the formulae defined over AP_i and which refer to component $\mathcal{M}_i$ only) and *mixed* formulae the formuale $\phi_{12} \in \Phi_{12}$ (i.e. the formulae which involve atomic propositions from both AP_1 and AP_2).
Furthermore, for any given ϕ_i formula, we denote the set of states of component $\mathcal{M}_i$ in which ϕ_i is satisfied as $Sat_i(\phi_i) \subseteq S_i$. As a consequence of the partition of the $\mathcal{M}_i$ state-space also the set $Sat_i(\phi_i)$ is partitioned. Hence we will use $Sat_{i_j}(\phi_i)$ to denote the states of model $\mathcal{M}_i$ satisfying ϕ_i and belonging to $S_{i,j}$ (with $i, j \in \{1, 2\}$).

***Single component* formulae:** the syntax of *single component* formulae is obtained from (1), by replacing the general atomic proposition a with an atomic proposition $a_i \in AP_i$, hence

$$\begin{aligned}
\phi_i &::= \ tt \ \mid \ a_i \ \mid \ \phi_i' \wedge \phi_i'' \ \mid \ \neg\phi_i' \ \mid \ \mathcal{S}_{\lhd p}(\phi_i') \ \mid \ \mathcal{P}_{\lhd p}(\varphi_i) \\
\varphi_i &::= \mathcal{X}(\phi_i) \ \mid \ \phi_i' \, \mathcal{U} \, \phi_i' \ \mid \ \phi_i' \, \mathcal{U}^{\leq t} \, \phi_i'
\end{aligned}$$

***Mixed* formulae:** *mixed* formulae, on the other hand, are characterised by the syntax

$$\begin{aligned}\phi_{12} ::= \;& \phi_1 \wedge \phi_2 \;\Big|\; \phi_2 \wedge \phi_1 \;\Big|\; \phi_i \wedge \phi'_{12} \;\Big|\; \phi'_{12} \wedge \phi_i \;\Big|\; \phi'_{12} \wedge \phi''_{12} \;\Big|\; \neg\phi'_{12} \;\Big|\\ & \mathcal{S}_{\lhd p}(\phi'_{12}) \;\Big|\; \mathcal{P}_{\lhd p}(\varphi'_{12})\\ \varphi_{12} ::= \;& \phi_1 \,\mathcal{U}\, \phi_2 \;\Big|\; \phi_2 \,\mathcal{U}\, \phi_1 \;\Big|\; \phi_i \,\mathcal{U}\, \phi'_{12} \;\Big|\; \phi'_{12} \,\mathcal{U}\, \phi_i \;\Big|\; \phi'_{12} \,\mathcal{U}\, \phi''_{12} \;\Big|\; \mathcal{X}\, \phi'_{12}\end{aligned}$$

2.1 Restricted CSL syntax

For the time being we consider a restricted version of the CSL syntax obtained from (1) by omitting the probabilistic operator $\mathcal{P}$. Hence the formulae we are considering here, are the subclass of the (standard) CSL state-formulae, characterised by the following syntax:

$$\phi ::= tt \;\Big|\; a \;\Big|\; \phi \wedge \phi \;\Big|\; \neg\phi \;\Big|\; \mathcal{S}_{\lhd p}(\phi) \tag{3}$$

This restriction results indeed on both *single component* and *mixed* formulae, whose restricted syntax is then given by:

$$\phi_1 ::= tt \;\Big|\; a_1 \;\Big|\; \phi'_1 \wedge \phi''_1 \;\Big|\; \neg\phi'_1 \;\Big|\; \mathcal{S}_{\lhd p}(\phi'_1) \tag{4}$$

$$\phi_{12} ::= \phi_1 \wedge \phi_2 \;\Big|\; \phi_2 \wedge \phi_1 \;\Big|\; \phi_i \wedge \phi'_{12} \;\Big|\; \phi'_{12} \wedge \phi_i \;\Big|\; \phi'_{12} \wedge \phi''_{12} \;\Big|\; \neg\phi'_{12} \;\Big|\; \mathcal{S}_{\lhd p}(\phi'_{12}) \tag{5}$$

2.2 Compositional model checking for single component formulae

We now focus on the definition of a compositional semantics for the "restricted" *single component* formulae[2] (4).
For this purpose, we introduce a "transformation" function, namely $f_t\colon \Phi_1 \to \Phi_1$, whose argument and returned value is a ϕ_1 formula. The function f_t is recursively defined as follows:

$$f_t(\phi_1) = \begin{cases} a_1 & \textbf{if } \;\phi_1 = a_1 \\ \neg f_t(\phi'_1) & \textbf{if } \;\phi_1 = \neg\phi'_1 \\ f_t(\phi'_1) \wedge f_t(\phi''_1) & \textbf{if } \;\phi_1 = \phi'_1 \wedge \phi''_1 \\ \mathcal{S}_{g(p,\phi'_1,\mathcal{M}_1)}(f_t(\phi'_1)) & \textbf{if } \;\phi_1 = \mathcal{S}_{\lhd p}\phi'_1 \end{cases} \tag{6}$$

[2] For the sake of simplicity, the following arguments are referred to ϕ_1 *single component* formulae (i.e. formulae referring to component $\mathcal{M}_1$). It is straightforward to show that equivalent considerations do hold also for ϕ_2 formulae.

where the function $g\colon ([0,1], \Phi, CTMC) \rightarrow [0,1]$ returns a probability value p', which depends on the triple $(p, \phi, \mathcal{M})$ given as argument. In the case of a Boucherie product-process, g is defined in the following way:

$$g(p, \phi_1, \mathcal{M}_1) = \begin{cases} \frac{p}{G} & \textbf{if } \; Sat_1(f_t(\phi_1)) \subseteq S_{1,1} \\ \\ \frac{p}{G \cdot K} & \textbf{if } \; Sat_1(f_t(\phi_1)) \subseteq S_{1,2} \\ \\ \frac{p - G \cdot J}{G \cdot K} & \textbf{if } \; (Sat_{1_1}(f_t(\phi_1)) \neq \emptyset) \wedge \\ & \qquad (Sat_{1_2}(f_t(\phi_1)) \neq \emptyset) \end{cases} \tag{7}$$

where G is the product-form *normalization* constant, and J and K are respectively defined as:

$$J = \sum_{s' \in Sat_{1_1}(f_t(\phi_1))} \pi_1(s') \sum_{s'' \in S_{2,2}} \pi_2(s'')$$

$$K = \sum_{t'' \in S_{2,1}} \pi_2(t'')$$

Example 2 *Referring to the Boucherie framework described in the running example 1, let us consider the following ϕ_1 formulae and see how they are transformed by means of f_t.*

1. $\phi_1 \equiv (res_1 \wedge rec_1) \qquad \Rightarrow \qquad f_t(\phi_1) = (res_1 \wedge rec_1)$

   which shows that the transformation of a state-formula which does not involve the steady-state operator, is the identity.

2. $\phi_1 \equiv \mathcal{S}_{\lhd p}(res_1 \wedge rec_1) \qquad \Rightarrow \qquad f_t(\phi_1) = \mathcal{S}_{\lhd p'}(res_1 \wedge rec_1)$

   where
   $$p' = \frac{p}{G \cdot \pi_2(s_{20})} = \frac{p}{G \cdot (r_3 r_6 r_7 (r_6 + r_7)/G_2)}$$
   since $Sat(res_1 \wedge rec_1) = \{s_{15}\} \subseteq S_{1,2}$ and, furthermore, $S_{2,1} = \{s_{20}\}$.

3. *Let us consider the application of the steady-state operator to a formula which is satisfied in states of both parts $S_{1,1}$ and $S_{1,2}$, $idle_1 \vee read_1$ which in the CSL syntax we would re-write as $\phi_1' \equiv \neg(\neg idle_1 \wedge \neg read_1)$. Since, as we have seen, the transformation of non steady-state formulae is the identity (i.e. $f_t(\phi_1') = \phi_1'$), we have that,*

   $\phi_1 \equiv \mathcal{S}_{\lhd p}\neg(\neg idle_1 \wedge \neg read_1) \qquad \Rightarrow \qquad f_t(\phi_1) = \mathcal{S}_{\lhd p'}\neg(\neg idle_1 \wedge \neg read_1)$

where $p' = g(p, \neg(\neg idle_1 \wedge \neg read_1), \mathcal{M}_1) = \frac{p - G \cdot J}{G \cdot K}$ *which results in,*

$$p' = \frac{p - G \cdot \left[r_1 r_3 r_4 (r_2 + r_4)\big(r_1(r_3(r_6 r_7 + r_6^2 + r_7^2) + r_6 r_7 (r_6 + r_7))\big)\right]}{G \cdot G_1 \cdot r_3 r_6 r_7 (r_6 + r_7)}$$

4. *finally let's consider a case showing a nested steady-state operator. According to the definition of* f_t *we have that*

$$\phi_1 \equiv \mathcal{S}_{\triangleleft p_1}(read_1 \wedge \mathcal{S}_{\triangleleft p_2}(idle_1)) \qquad \Rightarrow \qquad f_t(\phi_1) = \mathcal{S}_{\triangleleft p_1'}(read_1 \wedge \mathcal{S}_{\triangleleft p_2'}(idle_1))$$

where $p_2' = \frac{p_2}{G}$ *as* $Sat_1(idle_1) = \{s_{10}\} \subseteq S_{1,1}$ *and* $p_1' = \frac{p_1}{G \cdot \pi_2(S_{20})}$ *since* $Sat(read_1 \wedge \mathcal{S}_{\triangleleft p_2'}(idle_1)) = \{s_{11}\} \subseteq S_{1,2}$*, given that* $Sat(\mathcal{S}_{\triangleleft p_2'}(idle_1)) = S_1$*.*

The following theorem states that a single component formula ϕ_1 is satisfied in a Boucherie product process if and only if its transformation $f_t(\phi_1)$ is satisfied in the component $\mathcal{M}_1$.

Theorem 2 (Compositional model checking of *single component* formulae)
For any state (s', s'') *of a Boucherie product-process* $\mathcal{M}$ *and any single component state formula* ϕ_1 *defined as in (4), the following holds:*

$$(s', s'') \models \phi_1 \iff s' \models_1 f_t(\phi_1)$$

where f_t *is defined as in (6).*

proof. the proof is done by structural induction over the form ϕ_1. For the sake of brevity, here we omit it.

Example 3 *Referring to the cases showed in the previous Example 2, we have that, for any state* (s', s'') *of the product process:*

1. $(s', s'') \models (res_1 \wedge rec_1) \quad \Leftrightarrow \quad s' \models_1 (res_1 \wedge rec_1)$
2. $(s', s'') \models \mathcal{S}_{\triangleleft p}(res_1 \wedge rec_1) \quad \Leftrightarrow \quad s' \models_1 \mathcal{S}_{\triangleleft p'}(res_1 \wedge rec_1)$
3. $(s', s'') \models \mathcal{S}_{\triangleleft p} \neg(\neg idle_1 \wedge \neg read_1) \quad \Leftrightarrow \quad s' \models_1 \mathcal{S}_{\triangleleft p'} \neg(\neg idle_1 \wedge \neg read_1)$
4. $(s', s'') \models \mathcal{S}_{\triangleleft p_1}(read_1 \wedge \mathcal{S}_{\triangleleft p_2}(idle_1)) \quad \Leftrightarrow \quad s' \models_1 \mathcal{S}_{\triangleleft p_1'}(read_1 \wedge \mathcal{S}_{\triangleleft p_2'}(idle_1))$

2.3 Compositional model checking for mixed formulae

In order to derive a compositional semantics for the ϕ_{12} formulae (5), we introduce the following function

Definition 1 (function $cond()\colon S \times \Phi_{12} \to \mathcal{B}(\Gamma)$**)**
The function $cond()$ *returns a boolean combination over the set of proposition* Γ *(defined below) which depends on both the state* (s', s'') *and the formula* ϕ_{12} *given as arguments.*

$$
cond\big((s', s''), \phi_{12}\big) = \begin{cases}
s' \models_1 f_t(\phi_1) \ \ \textbf{and} \ \ s'' \models_2 f_t(\phi_2) & \textbf{if} \ \ \phi_{12} \equiv \phi_1 \wedge \phi_2 \\
s' \models_1 f_t(\phi_1) \ \ \textbf{and} \ \ cond\big((s', s''), \phi'_{12}\big) & \textbf{if} \ \ \phi_{12} \equiv \phi_1 \wedge \phi'_{12} \\
cond((s', s''), \phi'_{12}) \ \ \textbf{and} \ \ cond\big((s', s''), \phi''_{12}\big) & \textbf{if} \ \ \phi_{12} \equiv \phi'_{12} \wedge \phi''_{12} \\
\textbf{not} \ \ cond\big((s', s''), \phi'_{12}\big) & \textbf{if} \ \ \phi_{12} \equiv \neg\phi'_{12} \\
 & \\
s' \models_1 \mathcal{S}_{\lhd \frac{p - G \cdot J((\alpha_1, \beta_2), \phi'_{12})}{G \cdot K(\beta_2)}}(\alpha_1) & \textbf{if} \ \ \phi_{12} \equiv \mathcal{S}_{\lhd p}(\phi'_{12}) \ \wedge \\
 & (\alpha_1, \beta_2) \in dec(\phi'_{12})
\end{cases}
$$

where $\mathcal{B}(\Gamma)$ is the set of all boolean combinations of propositions belonging to the set $\Gamma = \Gamma_1 \cup \Gamma_2$ where Γ_i $(i \in 1, 2)$ is defined as:

$$\Gamma_i ::= \{s_i \models_i \phi_i : s_i \in S_i, \phi_i \in \Phi_i\}$$

while G is the product-form *normalisation* constant and for any pair of *single component* formulae α_1 and β_2 and any *mixed* formula ϕ_{12} the constants $K(\beta_2)$ and $J((\alpha_1, \beta_2), \phi_{12})$ are defined as:

$$
\begin{aligned}
J((\alpha_1, \beta_2), \phi_{12}) = & \sum_{(\delta_1, \gamma_2) \in dec(\phi_{12}) (\delta_1, \gamma_2) \neq (\alpha_1, \beta_2)} \Big[\sum_{t' \in Sat_1(\delta_1)} \pi_1(t') \sum_{t'' \in Sat_2(\gamma_2)} \pi_2(t'') \Big] \\
K(\beta_2) = & \sum_{t'' \in Sat_2(\beta_2)} \pi_2(t'')
\end{aligned}
$$

Furthermore the function $dec()$, used by $cond()$, is defined in the following way:

Definition 2 (function $dec() \colon \Phi_{12} \to \Phi_1 \times \Phi_2$)
The function $dec()$ decomposes a ϕ_{12} formula in a pair $(dec_1(\phi_{12}), dec_2(\phi_{12}))$, of single component *formulae, the first element of which, $(dec_1(\phi_{12}))$, is a ϕ_1 formula, while the second is a ϕ_2 formula.*

$$
dec(\phi_{12}) = \begin{cases}
\{(f_t(\phi_1), f_t(\phi_2))\} & \textbf{if} \ \ \phi_{12} \equiv \phi_1 \wedge \phi_2 \\
f_t(\phi_1) \ \ AND_1 \ \ dec(\phi'_{12}) & \textbf{if} \ \ \phi_{12} \equiv \phi_1 \wedge \phi'_{12} \\
 & \\
f_t(\phi_2) \ \ AND_2 \ \ dec(\phi'_{12}) & \textbf{if} \ \ \phi_{12} \equiv \phi_2 \wedge \phi'_{12} \\
dec(\phi'_{12}) \ \ AND \ \ dec(\phi''_{12}) & \textbf{if} \ \ \phi_{12} \equiv \phi'_{12} \wedge \phi''_{12} \\
AND_{(\alpha_1, \beta_2) \in dec(\phi'_{12})} \{(\neg\alpha_1, true), (\alpha_1, \neg\beta_2)\} & \textbf{if} \ \ \phi_{12} \equiv \neg\phi'_{12} \\
\{(true, true)\} & \textbf{if} \ \ \phi_{12} \equiv \mathcal{S}_{\lhd p} \phi'_{12} \wedge \\
 & s' \models_1 \mathcal{S}_{\lhd p'} dec_1(\alpha_1) \\
 & \quad where \ \ (\alpha_1, \beta_2) \in dec(\phi'_{12}) \\
\{(false, false)\} & \textbf{if} \ \ \phi_{12} \equiv \mathcal{S}_{\lhd p} \phi'_{12} \wedge \\
 & s' \not\models_1 \mathcal{S}_{\lhd p'} dec_1(\alpha_1)
\end{cases}
$$

where

$$(\gamma_1\ AND_1\ \Gamma) = \bigcup_{(\alpha_1 \wedge \beta_2) \in \Gamma} (\gamma_1 \wedge \alpha_1, \beta_2)$$

and

$$(\gamma_2\ AND_2\ \Gamma) = \bigcup_{(\alpha_1 \wedge \beta_2) \in \Gamma} (\alpha_1, \beta_2 \wedge \gamma_2).$$

Example 4 *Let us consider some example of decompositional conditions obtained by application of the cond() function to some mixed formulae. Again we refer our arguments to the running example 1.*

1. *if* $\phi_{12} \equiv (gat_1 \wedge read_2)$ *then* $dec(\phi_{12}) = (gat_1, read_2)$*, hence:*

$$cond((s_{11}, s_{21}), \phi_{12}) = s_{11} \models_1 gat_1 \wedge s_{21} \models_2 read_2$$

which shows that the condition equivalent to the satisfiability of the mixed formula $\phi_{12} \equiv (gat_1 \wedge read_2)$ *over the product process, is given by the conjuction of the satisfibility conditions* $s_{11} \models_1 gat_1$ *and* $s_{21} \models_2 read_2$ *over the components.*

2. *if* $\phi_{12} \equiv \mathcal{S}_{\triangleleft p}(gat_1 \wedge read_2)$ *then, since* $dec(gat_1 \wedge read_2) = (gat_1, read_2)$ *and, furthermore,* $Sat_{2_1}(read_2) = \emptyset$ *we are in the 5th case of the definition of cond(), hence:*

$$cond((s_{11}, s_{21}), \phi_{12}) = s_{11} \models_1 \mathcal{S}_{\triangleleft \frac{p+G \cdot J_b}{G \cdot K_b}}(gat_1)$$

with $J_b = 0$*, as* $Sat_{1_2}(gat_1) = \emptyset$*, and* $K_b = \pi_2(s_{21})$*, as* $Sat_{2_2}(read_2) = \{s_{21}\}$*. Thus* $s_{11} \models_1 \mathcal{S}_{\triangleleft \frac{p}{G \cdot \pi_2(s_{21})}}(gat_1)$ *is the satisfiability condition over component* $\mathcal{M}_1$*, which is equivalent to the satisfiability of* $\mathcal{S}_{\triangleleft p}(gat_1 \wedge read_2)$ *over the product process.*

3. *if* $\phi_{12} \equiv \mathcal{S}_{\triangleleft p}(idle_1 \wedge idle_2)$ *then, since* $dec(idle_1 \wedge idle_2) = (idle_1, idle_2)$ *and, furthermore,* $Sat_{2_1}(idle_2) = \{s_{20}\} \neq \emptyset$ *we are in the 4th case of the definition of cond(), hence:*

$$cond((s_{11}, s_{21}), \phi_{12}) = s_{11} \models_1 \mathcal{S}_{\triangleleft \frac{p-G \cdot J_a}{G \cdot K_a}}(idle_1)$$

with $J_a = 0$*, as* $Sat_{2_2}(idle_2) = \emptyset$*, and* $K_b = \pi_2(s_{20})$*, as* $Sat_{2_1}(idle_2) = \{s_{20}\}$*. Thus* $s_{11} \models_1 \mathcal{S}_{\triangleleft \frac{p}{G \cdot \pi_2(s_{20})}}(idle_1)$ *is the condition over component* $\mathcal{M}_1$*, equivalent* $\mathcal{S}_{\triangleleft p}(gat_1 \wedge read_2)$*.*

Theorem 3 (Compositional model checking of *mixed* formulae)
For any state (s', s'') *of a Boucherie product-process* $\mathcal{M}$ *and any mixed state formula* ϕ_{12} *defined as in (5), the following holds:*

$$(s', s'') \models \phi_{12} \iff cond\big((s', s''), \phi_{12})\big)$$

where cond() is defined as in definition 1.

Proof. the proof is done by structural induction over the definition of the function *cond*() . For the sake of brevity, here we omit it.

Example 5 (satisfiability of mixed formulae) *As a verification of theorem 3 let us consider case 3 of example 4 which is* $\phi_{12} \equiv \mathcal{S}_{\lhd p}(gat_1 \wedge read_2)$. *From CSL semantics, we know that* $(s_{11}, s_{21}) \models \mathcal{S}_{\lhd p}(gat_1 \wedge read_2)$ *if and only if*

$$G[\sum_{(t',t'') \in Sat(gat_1 \wedge read_2)} \pi_1(t')\pi_2(t'')] \lhd p$$

Considering our running example, it is easy to see that $Sat(gat_1 \wedge read_2) = \{(s_{11}, s_{21})\}$, *hence we can re-write the above inequality as:* $G[\pi_1(s_{11})\pi_2(s_{21})] \lhd p$ *or also*

$$\pi_1(s_{11}) \lhd \frac{p}{G \cdot \pi_2(s_{21})}$$

Since $Sat_1(gat_1) = \{s_{11}\}$ *then the above inequality also implies that*

$$s_{11} \models_1 \mathcal{S}_{\lhd \frac{p}{G \cdot \pi_2(s_{21})}}(gat_1)$$

which proves theorem 3 to be true for the considered example.

3 Conclusion and future work

In this paper we have shown that for the simplest possible configuration of the Boucherie framework (the one where the product process is obtained by the composition of two components), a decompositional CSL model checking is possible for all the formulae belonging to a restricted version of the CSL grammar, given by the elimination of the probabilistc path-formulae.

That is achieved by identification of a finite boolean combination of satisfiability conditions for *single component* formulae, which prove to be equivalent to the original formula (from which they are derived).

The objective of future (ongoing) work, is to proceed with the analysis of the more complicated probabilistic path formulae. The main issue in that case is to find a way to relate paths satisfying a given *path formula* over the composed process with paths satisfying the same formula on the components.

Finally, the complexity analysis of the decompositional approach (computational costs of the functions $f_t()$, *cond*() and *dec*()), need to be performed in order to be able to evaluate the cost of the decompositional methodology.

References

1. K. Sanwal V. Singhal A. Aziz and R. Brayton. Verifying continuous time markov chains. In *in CAV LNCS 939, pp. 155-165*, 1996.
2. R.J. Boucherie. A characterisation of indipendence for competing markov chains with applications to stochastic petri nets. In *IEEE Trans. on Software EEngineering 20(7), pp. 536-544*, 1994.

3. J. P. Katoen C. Baier and H. Hermann. Approximate symbolic model checking of continuos-time markov chains. In *in Concurrency Theory, LNCS 1664: 146-162*, 1999.
4. E. Emerson E. Clarke and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *ACM Trans. on Prog. Lang. and Sys., 244-263*, 1986.
5. H. A. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *In Proc. 10th IEEE Real -Time Systems Symposium, Santa Monica, Ca.*, 1989.
6. A. Pnueli. A temporal logic of programs. In *Theoretical Computer Science,13, pp.45-60*, 1981.