Int. J. Appl. Math. Comput. Sci., 2023, Vol. 33, No. 1, 133–149
DOI: 10.34768/amcs-2023-0011

GENERATION OF SYNCHRONIZING STATE MACHINES FROM A TRANSITION SYSTEM: A REGION–BASED APPROACH

VIKTOR TEREN [a,*], JORDI CORTADELLA [b], TIZIANO VILLA [a]

[a]Department of Computer Science
University of Verona
Strada le Grazie 15, 37134, Verona, Italy
e-mail: {viktor.teren,tiziano.villa}@univr.it

[b]Department of Computer Science
Polytechnic University of Catalonia
Jordi Girona Salgado 1–3, 08034, Barcelona, Spain
e-mail: jordi.cortadella@upc.edu

Transition systems (TSs) and Petri nets (PNs) are important models of computation ubiquitous in formal methods for modeling systems. A crucial problem is how to extract, from a given TS, a PN whose reachability graph is equivalent (with a suitable notion of equivalence) to the original TS. This paper addresses the decomposition of transition systems into synchronizing state machines (SMs), which are a class of Petri nets where each transition has one incoming and one outgoing arc. Furthermore, all reachable markings (non-negative vectors representing the number of tokens for each place) of an SM have only one marked place with only one token. This is a significant case of the general problem of extracting a PN from a TS. The decomposition is based on the theory of regions, and it is shown that a property of regions called excitation-closure is a sufficient condition to guarantee the equivalence between the original TS and a decomposition into SMs. An efficient algorithm is provided which solves the problem by reducing its critical steps to the maximal independent set problem (to compute a minimal set of irredundant SMs) or to satisfiability (to merge the SMs). We report experimental results that show a good trade-off between quality of results vs. computation time.

Keywords: transition system, Petri net, state machine, decomposition, theory of regions, SAT, pseudo-Boolean optimization.

1. Introduction

The decomposition of a transition system (TS) into a synchronous product of state machines gives an intermediate model between a TS and a Petri net (PN). The set of SMs may exhibit fewer distributed states and transitions, exploiting the best of both worlds of TSs and PNs, leading to better implementations (e.g., smaller circuits with probably less power consumption (Benini *et al*., 2001)). Furthermore, the decomposition procedure extracts explicitly the system concurrency: a property identified when given a marking, two or more places have a token and are able to fire independent transitions with an arbitrary order, a PN feature, which is convenient for system analysis and performance improvement. One can get an idea of the efficacy of the decomposition process by comparing Figs. 1 and 2, where the SMs of the latter expose the implicit parallelism of the former.

Notice that each SM is completely concurrent with the others; therefore, any firing order is allowed, except when there are synchronizations on shared events.

The decomposition of a transition system can be seen from the Petri net perspective as the problem of the coverability by S-components of a Petri net (Kemper and Bause, 1992; Desel, 1995; Mattheakis, 2013) or of a connected subnet system (Badouel *et al*., 2015, p. 49) (called S-coverability): each S-component is a strongly connected safe SM, i.e., an SM with only one token, therefore it cannot contain concurrency. The only concurrency of the system takes place in the interaction of the S-components. Carmona *et al*. (2009c) investigated

*Corresponding author

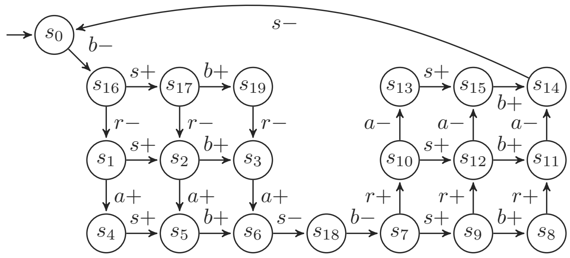


Fig. 1. TS derived from an STG.[1]

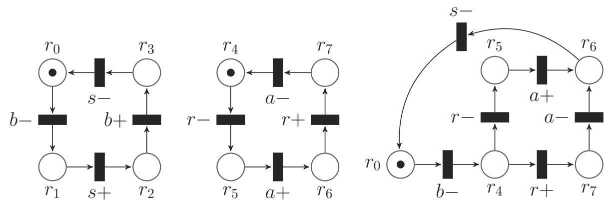


Fig. 2. Set of synchronizing state machines derived from the TS in Fig. 1.

Table 1. List of abbreviations used in this article.

Abbreviation	Explanation
EC	Excitation-closure
ECTS	Excitation-closed transition system
ES	Excitation set
HPC	High performance computing
ILP	Integer linear programming
MIS	Maximal independent set
PN	Petri net
RG	Reachability graph
SAT	Boolean satisfiability
SM	State machine
SS	Switching set
TS	Transition system
UNSAT	Boolean unsatisfiability

synthesis of k-bounded Petri nets, i.e., nets which contain at most k tokens simultaneously in a place. In our case the extension to k-bounded SMs would raise the computational complexity of the decomposition flow. Furthermore, the concurrency, which is only possible between SMs, would become possible also inside single SMs.

In this paper, following the approach of the previous short version (Teren *et al.*, 2021), we start from the theory of regions (Ehrenfeucht and Rozenberg, 1990) to design a procedure which, given a transition system, generates a matching set of interacting SMs, without building an equivalent Petri net, which were the original motivation to define regions (Cortadella *et al.*, 1995). Our approach computes a set of minimal regions with the excitation-closure (EC) property of a given TS, and derives from them an irredundant synchronous product of interacting SMs. Excitation-closure guarantees that the regions extracted from the transition system are sufficient to model its behaviour.

The main steps of the decomposition procedure are: (i) computation of all minimal regions of the given TS, (ii) generation of a set of SMs with the excitation-closure property, (iii) removal of redundant SMs, (iv) merging of regions while preserving the excitation-closure property. The generation of minimal regions is well known from the literature (Cortadella *et al.*, 1998). The generation of SMs with the EC property is reduced to solving instances of the maximal independent set (MIS)[2], where each solution of MIS yields an SM. Some of these SMs may be completely redundant, i.e., they can be removed while the remaining partially redundant SMs still satisfy the EC property. We use a greedy strategy to find a minimal irredundant set of SMs. This step represents our first and most important trade-off, since the search stops when a sufficient number of SMs is found without exploring all of them, so that this set of SMs is an approximation of the optimal result. This trade-off represents also the hardest challenge: decomposing the transition system into the fewest SMs, trying to reach near optimal results, but at the same time without using exact algorithms which are too time-consuming.

In Section 5 we also show the result of performing the search of all possible SMs. The surviving SMs go through a simplification step that merges adjacent regions and removes the edges/labels captured by the merging step. In the extreme case, one can remove all instances of a region except for one SM. The best merging option is selected by encoding both, the constraints of the merging operations and the optimization objective as an ILP[3], solvable by SAT solvers and binary search (Boros and Hammer, 2002), with the goal of keeping the minimum number of labels needed to satisfy the EC property. At the end, the SMs are optimized according to the selected merging operations.

The optimization steps in which the problem is divided may be solved exactly or with heuristics. Experiments have been performed trying various

[1] A signal transition graph (STG) $G = (V, E)$ is an interpreted subset of marked graphs wherein each transition represents either the rising $(x+)$ or falling $(x-)$ of a signal x which has signal levels high and low. V is the set of transitions and E is the set of edges corresponding to places of the underlying marked graph.

[2] Given an undirected graph $G = (V, E)$, an *independent set* is a subset of nodes $U \subseteq V$ such that no two nodes in U are adjacent. An independent set is maximal if no node can be added without violating independence.

[3] *Integer linear programming*, or ILP, investigates linear programming problems in which the variables are restricted to integers: the general problem is to determine $\max\{cx|Ax \leq b; x \text{ integral}\}$ (Schrijver, 1998).

combinations of exact and heuristic algorithms, with the conclusion that the heuristics deliver good results in reasonable computation time.

1.1. State of the art. Kalenkova *et al*. (2014) decomposed a transition system iteratively into an interconnection of n component transition systems with the objective to extract a Petri net from them. This can be seen as a special case of our problem, because by Kalenkova *et al*. (2014) the decomposition allows the extraction of a Petri net, but the decomposed set of transition systems cannot be used as an intermediate model. Their approach is flexible in choosing how to split the original transition system, but it does not provide any minimization algorithm, so that the redundancy due to overlapping states in the component transition systems translates into redundant places of the final Petri net. Another method presented by de San Pedro and Cortadella (2016) is based on the decomposition of transition systems into "slices," where each transition system is separately synthesized into a Petri net, and in the case of Petri nets "hard" to understand the process can be recursively repeated on one or more "slices" creating a higher number of smaller PNs. With respect to the aforementioned methods, our approach yields by construction a set of PNs restricted to only SMs and applies to them minimization criteria. The results of Mokhov *et al*. (2017) instead show how complex processes can be formally represented by process windows, where each window covers a part of the process behaviour. In our case, each SM could be interpreted as a window representing a part of the entire process.

1.1.1. Decomposition in process mining. The aim of de San Pedro and Cortadella (2016) is the mining of comprehensive Petri nets for a better visualization of spaghetti models obtained by process mining. Also decomposition plays an important role in process mining, especially in business process management (BPM) (Van der Aalst, 2012; 2013; Verbeek and Van der Aalst, 2014; Taibi and Systä, 2019), where a decomposed process can be better understood and maybe parallelized. Say that we mined some traces and represented them as a transition system; then the decomposition of the transition system splits it as different concurrent flows which can be analyzed separately. Furthermore, since each SM is completely concurrent with the others, also parallelization of concurrent processes becomes easier. In most cases, the decomposition starts from a Petri net representing the whole behaviour of the system (Van der Aalst, 2012; 2013; Verbeek and Van der Aalst, 2014). Instead of creating a PN from event logs, we can easily create a transition system (Van der Aalst *et al*., 2010; Carmona *et al*., 2009a) and directly decompose it with our algorithm. The application to process decomposition is part of current research that will be reported when completed.

1.2. Contributions. This is an extended version of the short paper that we presented at the *24th Euromicro Conference on Digital System Design (DSD)* (Teren *et al*., 2021). This paper extends the decomposition algorithm of the conference version by introducing a new mixed strategy to select the components state machines of the decomposition. The new mixed strategy combines exact and heuristic algorithms for the removal of redundant SMs, and operates adaptively according to the number of SMs obtained after the initial extraction step. We report the new related experiments showing the effectiveness of the mixed strategy.

Altogether, the new material includes: additional definitions, complete proofs and detailed examples for each step of the decomposition procedure, a new decomposition strategy with revised experiments. In particular, we added the bisimulation proof, a description of the SAT clause encoding and a step-by-step example of SM set generation.

The paper is organized as follows. Section 2 introduces the background material (including the theory of regions to extract PNs from TSs) and then characterizes the extraction of SMs from TSs. The procedures to extract the SMs are described in Section 3. Section 4 discusses composition of SMs and contains the main theoretical result that the synchronous product of SMs is bisimilar to the original transition system (proof in Appendix). Exhaustive experiments are reported in Section 5, with final conclusions drawn in Section 6.

2. Preliminaries

2.1. Transition systems.

Definition 1. (*TS/LTS (Cortadella* et al.*, 1998)*) A labeled transition system (LTS, or simply TS) is defined as the quadruple (S, E, T, s_0), where

- S is a non-empty set of states,
- E is a set of events/labels,
- $T \subseteq S \times E \times S$ is a transition relation,
- $s_0 \in S$ is an initial state.

Every transition system is supposed to satisfy the following properties:

- it does not contain-self loops: $\forall(s, e, s') \in T$: $s \neq s'$;
- each event has at least one occurrence: $\forall e \in E$: $\exists(s, e, s') \in T$;

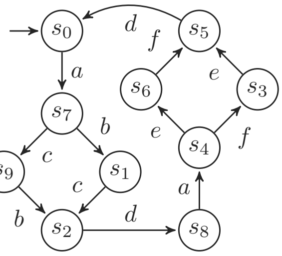


Fig. 3. Example of a transition system.

- every state is reachable from the initial state: $\forall s \in S : s_0 \to^* s$;
- it is deterministic: for each state there is at most one successor state reachable with label e.

An example of a transition system can be seen in Fig. 3.

Definition 2. (*Isomorphism*) Two transition systems $\mathrm{TS}_1 = (S_1, E, T_1, s_{0,1})$ and $\mathrm{TS}_2 = (S_2, E, T_2, s_{0,2})$ are said to be isomorphic (or that there is an isomorphism between TS_1 and TS_2) if there is a bijection $b_S : S_1 \to S_2$, such that

- $b_S(s_{0,1}) = s_{0,2}$,
- $\forall (s, e, s') \in T_1 : \; (b_S(s), e, b_S(s')) \in T_2$,
- $\forall (s, e, s') \in T_2 : \; (b_S^{-1}(s), e, b_S^{-1}(s')) \in T_1$.

Definition 3. (*Bisimulation*) Given two transition systems $\mathrm{TS}_1 = (S_1, E, T_1, \; s_{0,1})$ and $\mathrm{TS}_2 = (S_2, E, T_2, s_{0,2})$, a binary relation $B \subseteq S_1 \times S_2$ is a bisimulation, denoted by $\mathrm{TS}_1 \sim_B \mathrm{TS}_2$, if $(s_{0,1}, s_{0,2}) \in B$ and if whenever $(p, q) \in B$ with $p \in S_1$ and $q \in S_2$:

- $\forall (p, e, p') \in T_1 : \; \exists q' \in S_2$ such that $(q, e, q') \in T_2$ and $(p', q') \in B$,
- $\forall (q, e, q') \in T_2 : \; \exists p' \in S_1$ such that $(p, e, p') \in T_1$ and $(p', q') \in B$.

Two TSs are said to be bisimilar if there is a bisimulation between them.

The operation ‘Ac’ deletes from a TS all the states that are not reachable or accessible from the initial state and all transitions attached to them.

Definition 4. (*Synchronous product*) Given two transition systems $\mathrm{TS}_1 = (S_1, E_1, T_1, s_{0,1})$ and $\mathrm{TS}_2 = (S_2, E_2, \; T_2, s_{0,2})$, the synchronous product is defined as $\mathrm{TS}_1 || \mathrm{TS}_2 = \mathrm{Ac}(S, E_1 \cup E_2, T, (s_{0,1}, s_{0,2}))$ where $S \subseteq S_1 \times S_2$, $(s_{0,1}, s_{0,2}) \in S$, $T \subseteq (S_1 \times S_2) \times E \times (S_1 \times S_2)$ is defined as follows:

- if $a \in E_1 \cap E_2$, $(s_1, a, s_1') \in T_1$ and $(s_2, a, s_2') \in T_2$ then $((s_1, s_2), a, (s_1', s_2')) \in T$,
- if $a \in E_1$, $a \notin E_2$ and $(s_1, a, s_1') \in T_1$ then $((s_1, s_2), a, (s_1', s_2)) \in T$,
- if $a \notin E_1$, $a \in E_2$ and $(s_2, a, s_2') \in T_2$ then $((s_1, s_2), a, (s_1, s_2')) \in T$,
- nothing else belongs to T.

The synchronous product is associative, so we can define the product of a collection of n TSs: $\mathrm{TS}_1 || \mathrm{TS}_2 || \ldots || \mathrm{TS}_n = ((\mathrm{TS}_1 || \mathrm{TS}_2) \ldots) || \mathrm{TS}_n$; as an alternative, we can extend directly the previous definition to more than two TSs.

2.2. Petri nets. We assume the reader to be familiar with Petri nets. We refer to Murata (1989) for a deeper insight on the concepts used in this work. This section introduces the nomenclature related to Petri nets used along the paper.

In this work we will only deal with safe Petri nets, i.e., nets whose places do not contain more than one token in any reachable marking. For this reason, we will model markings as sets of places. This approach could be extended to k-bounded Petri nets, but the extension would increase the computational complexity of the algorithm.

Definition 5. (*Ordinary Petri net (Murata, 1989)*) An ordinary Petri net is the quadruple $\mathrm{PN} = (P, T, \; F, M_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- M_0 is an initial marking,
- $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A Petri net structure $N = (P, T, F)$ without any specific initial marking is denoted by N. A Petri net with an initial marking M_0 is denoted by (N, M_0).

For any $x \in P \cup T$, then ${}^\bullet x = \{y | (y, x) \in F\}$. Similarly, $x^\bullet = \{y \mid (x, y) \in F\}$.

Definition 6. (*Firing rule (Badouel* et al.*, 2015, p. 17)*) Let $N = (P, T, F, M_0)$ be a safe Petri net. A transition $t \in T$ enabled in marking M is represented as $M[t\rangle$. If t is enabled in M, then t can be fired leading to another marking M', denoted as $M[t\rangle M'$, such that $M' = M \backslash {}^\bullet t \cup t^\bullet$.

We call $[M\rangle$ the set of markings that can be reached from M by firing sequences of enabled transitions.

Definition 7. (*Reachability graph (Badouel* et al.*, 2015, p. 20)*) Given a safe Petri net $N = (P, \; T, F, M_0)$, the reachability graph of N is the transition system $\mathrm{RG}(N) = ([M_0\rangle, T, \Delta, M_0)$ defined by $(M, t, M') \in \Delta$ if $M \in [M_0\rangle$ and $M[t\rangle M'$.

Definition 8. (*State machine, SM (Murata, 1989)*) A state machine is an ordinary Petri net, $N = (P, T, F, M_0)$ such that for every transition $t \in T$, $|^\bullet t| = |t^\bullet| = 1$, i.e., it has exactly one incoming and one outgoing edge. In a safe state machine it also holds that $|M_0| = 1$.

For an analysis of safeness in Petri nets, we refer to the work of Wojnakowski *et al*. (2021).

Badouel *et al*. (2015, p. 49) observed that a state machine $M = (P, T, F, M_0)$ can be interpreted as a transition system $\mathrm{TS} = (P, T, \Delta, s_0)$, where the places correspond to the states, the transitions to the events, s_0 corresponds to the unique marked initial place, and $(p, t, p') \in \Delta$ iff $^\bullet t = \{p\}$ and $t^\bullet = \{p'\}$ (in an SM by definition $|^\bullet t| = |t^\bullet| = 1$). Therefore the reachability graph of M is isomorphic to the transition system TS, i.e., RG(M) is isomorphic to TS.

In this paper we consider sets of synchronizing SMs.

2.3. From LTS to Petri nets by regions. In this paper we propose a procedure for the decomposition of transition systems based on the theory of regions (from the work of Cortadella *et al*. (1998)). A region is a subset of states in which all the transitions under the same event have the same relation with the region: either all entering, or all exiting, or some completely inside and some completely outside the region.

Definition 9. (*Region*) Given a TS $= (S, E, T, s_0)$, a region is defined as a non-empty set of states $r \subsetneq S$ such that the following properties hold for each event $e \in E$:

$$
\begin{aligned}
enter(e, r) &\implies \neg in(e, r) \wedge \neg out(e, r) \wedge \neg exit(e, r),\\
exit(e, r) &\implies \neg in(e, r) \wedge \neg out(e, r) \wedge \neg enter(e, r),\\
no\_cross(e, r) &\implies \neg enter(e, r) \wedge \neg exit(e, r),
\end{aligned}
$$

where

$$
\begin{aligned}
in(e, r) &\equiv \exists (s, e, s') \in T : s, s' \in r,\\
out(e, r) &\equiv \exists (s, e, s') \in T : s, s' \notin r,\\
enter(e, r) &\equiv \exists (s, e, s') \in T : s \notin r \wedge s' \in r,\\
exit(e, r) &\equiv \exists (s, e, s') \in T : s \in r \wedge s' \notin r,\\
no\_cross(e, r) &\equiv in(e, r) \vee out(e, r).
\end{aligned}
$$

Definition 10. (*Minimal region*) A region r is called *minimal* if there is no other region r' strictly contained in r ($\nexists r' \mid r' \subset r$).

The minimal regions of the TS in Fig. 3 are shown in Table 2.

Definition 11. (*Pre-region (resp. post-region)*) A region r is a pre-region (resp. post-region) of an event e if there is a transition labeled with e which exits from r (resp. enters into r). The set of all pre-regions (resp. post-regions) of the event e is denoted by $^\circ e$ (e°).

Table 2. Minimal regions of the TS in Fig. 3.

Region	States of the TS
r_1	$\{s_0, s_8\}$
r_2	$\{s_0, s_1, s_3, s_5, s_7\}$
r_3	$\{s_0, s_5, s_6, s_7, s_9\}$
r_4	$\{s_1, s_2, s_3, s_4, s_8\}$
r_5	$\{s_1, s_2, s_3, s_5\}$
r_6	$\{s_1, s_4, s_6, s_7\}$
r_7	$\{s_2, s_4, s_6, s_8, s_9\}$
r_8	$\{s_2, s_5, s_6, s_9\}$
r_9	$\{s_3, s_4, s_7, s_9\}$
r_{10}	$\{s_0, s_1, s_5, s_6, s_7\}$
r_{11}	$\{s_0, s_3, s_5, s_7, s_9\}$
r_{12}	$\{s_1, s_2, s_4, s_6, s_8\}$
r_{13}	$\{s_1, s_2, s_5, s_6\}$
r_{14}	$\{s_1, s_3, s_4, s_7\}$
r_{15}	$\{s_2, s_3, s_4, s_8, s_9\}$
r_{16}	$\{s_2, s_3, s_5, s_9\}$
r_{17}	$\{s_4, s_6, s_7, s_9\}$

Table 3. Pre-regions and ESs for each event of the TS in Fig. 3.

Event	Pre-regions	ES(event)
a	$\{r_1\}$	$\{s_0, s_8\}$
b	$\{r_3, r_9, r_{11}, r_{17}\}$	$\{s_7, s_9\}$
c	$\{r_2, r_6, r_{10}, r_{14}\}$	$\{s_1, s_7\}$
d	$\{r_5, r_8, r_{13}, r_{16}\}$	$\{s_2, s_5\}$
e	$\{r_4, r_9, r_{14}, r_{15}\}$	$\{s_3, s_4\}$
f	$\{r_6, r_7, r_{12}, r_{17}\}$	$\{s_4, s_6\}$

By definition, if $r \in {}^\circ e$ (resp. $r \in e^\circ$) all the transitions labeled with e are exiting from r (resp. entering into r), furthermore, if the transition system is strongly connected, all the regions are also pre-regions of some event.

Definition 12. (*Excitation set/switching set*) The *excitation (resp. switching) set* of event e, ES(e) (SS(e)), is the maximal set of states such that for every $s \in$ ES(e) (resp. $s \in$ SS(e)) there is a transition $t \in T$ such that $t = (s', e, s)$ (resp. $t = (s, e, s')$).

The excitation sets of the TS in Fig. 3 are reported in Table 3.

Definition 13. (*Excitation-closed transition system, ECTS*) A TS with the set of labels E and the pre-regions $^\circ e$ is an ECTS if the following conditions are satisfied:

- excitation-closure: $\forall e \in E : \bigcap_{r \in {}^\circ e} r = \mathrm{ES}(e)$,
- event effectiveness: $\forall e \in E : {}^\circ e \neq \emptyset$.

If the initial TS does not satisfy the excitation-closure (EC) or event effectiveness property, *label split-*

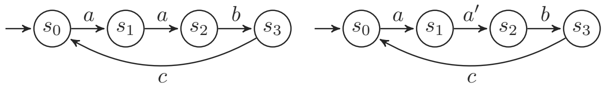


Fig. 4. TS before label splitting (a) and ECTS after label splitting (b).

ting (Cortadella *et al.*, 1998) can be performed to obtain an ECTS.

An example can be seen in Fig. 4: the initial TS has two regions $r_1 = \{s_0, s_1, s_2\}$ and $r_2 = \{s_3\}$. Label a satisfies the *no-cross* property, and so it is not an ECTS, because, e.g., event effectiveness is not satisfied for the event a: $^\circ a = \emptyset$. Also excitation-closure is not satisfied for the event b: $\bigcap_{r\in {^\circ b}} r = r_1 \neq \mathrm{ES}(b)$.

After label splitting, label a is split into a and a' yielding the following smaller minimal regions: $r_0 = \{s_0\}$, $r_1 = \{s_1\}$, $r_2 = \{s_2\}$ and $r_3 = \{s_3\}$. After label splitting, both excitation-closure and event effectiveness are satisfied.

The EC property also ensures that if two states, s_1 and s_2, cannot be *separated* by any region, i.e., there is no minimal region r such that $s_1 \in r$ and $s_2 \notin r$, then s_1 and s_2 are bisimilar.

The synthesis of a Petri net from an ECTS, proposed by Cortadella *et al.* (1998), can be summarized by the following steps:

1. *Generation of all minimal regions.*
   All the excitation sets are expanded until they become regions, i.e., all events satisfy one of the *enter/exit/no_cross* conditions with respect to the regions. The non-minimal regions can be removed by comparing them with the other regions.

2. *Removal of redundant regions.*
   Some minimal regions may be redundant, meaning that they can be removed while the excitation-closure property still holds.

3. *Merging minimal regions.*
   In order to obtain a place-minimal PN, subsets of disjoint minimal regions can be merged into non-minimal regions, thus reducing the number of places. This merging must preserve the excitation-closure of the final set of regions.

2.4. From LTS to SMs by regions. We now show how to decompose an ECTS into a set of synchronizing SMs.

From the set of all minimal regions obtained from an ECTS we can extract subsets of regions representing state machines. A set of regions R represents a state machine if R covers all the states S of the transition system and all the regions are disjoint, i.e.,

$$(\forall r \in R, \nexists r' \in R : r \cap r' \neq \emptyset) \wedge (\forall s \in S, \exists r \in R : s \in r)$$

Given a set of regions satisfying the previous properties we obtain a state machine whose places correspond to the regions, with a transition $r_i \xrightarrow{e} r_j$ when r_i and r_j are pre- and post-regions of e, respectively. Since the regions of an SM are disjoint, each derived SM has only one marked place, which corresponds to the regions that cover the initial state. Notice that *only* the events that cross some region appear in the SM. Notice also that the reachability property of the original TS is inherited by the SMs obtained by this construction.

Theorem 1. *Given an ECTS TS $= (S, E, T, s_0)$ and the set of all its minimal regions, a subset of regions R represents an SM if and only if the set covers all the states of TS and all its regions are pairwise disjoint.*

Proof. The proof is based on the fact that every event appearing in one SM can only have one pre-region and one post-region in the SM. Therefore, each event has one incoming and one outgoing edge in the SM.

Given a collection R of disjoint regions that cover all states of TS, each element $r_i \in R$ has entering, exiting and no-crossing events. We claim the following:

1. If event e exits (enters) region $r_i \in R$, it cannot exit (enter) region $r_j \in R, j \neq i$.

2. If event e exits (enters) region $r_i \in R$, there must be a region $r_j \in R, j \neq i$, such that event e enters (exits) $r_j \in R, j \neq i$.

We prove the first claim. Given a region r_i with e as exiting event, there cannot be another region r_j such that e is an exiting event also for r_j. Otherwise, i.e., if $r_i \in {^\circ e}$ and $r_j \in {^\circ e}$, $j \neq i$, there are two transitions $s_a \xrightarrow{e} s_b$ and $s_c \xrightarrow{e} s_d$ with $s_a \in r_i$ and $s_c \in r_j$. There are two options for s_b: either it is inside or outside r_j, i.e., $s_b \in r_j$ or $s_b \notin r_j$, which means that e would either be entering or no-crossing for r_j, contradicting that by construction r_j is a region with e as an exiting arc. The same reasoning applies when e is an entering event.

We prove the second claim: if event e appears as exiting (entering) event of $r_i \in R$, it must appear as entering (exiting) event of $r_j \in R$. Indeed, suppose that $r_i \in {^\circ e}$, then there is a transition $s_a \xrightarrow{e} s_b$ with $s_a \in r_i$ and $s_b \notin r_i$, but then there must exist a region $r_j \in R, j \neq i$, such that $s_b \in r_j$, because the union of the regions in R covers all the states of the original TS, and so $r_j \in e^\circ$. The case $r_i \in e^\circ$ is proved similarly.

Notice that we use also the fact that in our definition of TS we rule out self-loops. ∎

The property of excitation-closure can be inherited by the SMs, as stated in the following definition.

Definition 14. (*Excitation-closed set of state machines derived from an ECTS*) Given a set of SMs S derived from an ECTS TS, the set of all regions R of S, the set of labels E of TS, and the sets of pre-regions $^\circ e$ of the TS for all $e \in E$, we have that S is excitation-closed with respect to the regions of TS if the following conditions are satisfied:

- EC: $\forall e \in E : \bigcap_{r \in (^\circ e \cap R)} r = \text{ES}(e)$,
- event effectiveness: $\forall e \in E : \exists r \in R \mid r \in {}^\circ e$.

3. Decomposition algorithm

The first step to decompose a transition system is to enumerate all the minimal regions of the original TS. Each collection of disjoint regions covering all the states of the TS represents a state machine, such that the regions are mapped to places of the SM, i.e., each such SM includes a subset of regions of the original TS and represents only the behavior related to the transitions entering into these regions or exiting from them (instead, internal and external events are missing).

The example in Section 4 shows also that we do not need all the SMs to reconstruct the original LTS, so the question is how many of them we need and which is the "best" (in some sense) subset of SMs sufficient to represent the given LTS. Therefore, we may set up a search to obtain a subset of SMs, which are excitation-closed and cover all events, to yield a composition equivalent to the original TS. An easy strategy to guarantee the complete coverage of all events is to add new SMs until all regions are used. However, the resulting collection of SMs may contain completely or partially redundant SMs (see Sections 3.2 and 3.3), which can be removed exactly or greedily by verifying the excitation-closure property. Moreover, the size of the selected SMs can be reduced through removing redundant labels by merging regions. Summarizing, (i) minimal regions are computed, (ii) from which a set of SMs with EC is generated, (iii) redundant SMs are removed and, lastly, (iv) regions are merged preserving the EC property.

The first step of the algorithm can be achieved by a greedy algorithm from the literature, which checks minimality while creating regions (Cortadella *et al.*, 1998; 1997; Badouel *et al.*, 2015, p. 103).

The second step of the decomposition algorithm is performed by reducing it to an instance of maximal independent set (MIS), and by calling an MIS solver on the graph whose vertices correspond to the minimal regions with edges which connect intersecting regions. Each *maximal independent set* of the aforementioned graph corresponds to a set of disjoint regions that define an SM.

A greedy algorithm is used for the computation of the third step: starting from the SM with the highest number of regions, one removes each SM whose removal does not invalidate the ECTS properties.

Algorithm 1. Generation of excitation-closed set of SMs.

Require: Set of minimal regions of an ECTS
Ensure: An excitation-closed set of SMs
1: Create the graph G where each node is a region and there is an edge between intersecting regions
2: $G_0 \leftarrow G$
3: $M \leftarrow \emptyset$, $F \leftarrow \emptyset$
4: **do**
5: Compute $m = \text{MIS}(G)$
6: $M \leftarrow M \cup \{m\}$
7: $G \leftarrow G \setminus M$
8: **while** $G \neq \emptyset$
9: **for** $m \in M$ **do**
10: Compute $\tilde{m} = \text{MIS}(G_0)$ with the constraint $\tilde{m} \supseteq m$
11: Build state machine $s\tilde{m}$ induced by set of regions $\tilde{m}$
12: $F \leftarrow F \cup \{s\tilde{m}\}$
13: **end for**
14: **return** F

The last step of merging is reduced to a SAT instance, by encoding all the regions of each SM and also the events implied by the presence of one or more regions. Solving this SAT instance by a SAT solver, the number of labels can be minimized by merging the regions which occur multiple times in different SMs.

3.1. Generation of a set of SMs with excitation-closure.

Given a set of minimal regions of an excitation-closed TS, Algorithm 1 returns an excitation-closed set of SMs, by associating sets of non-overlapping regions to SMs as mentioned below. Notice that in Definition 14 we extended Definition 13 of an excitation-closed transition system (ECTS) to an excitation-closed set of SMs, by requiring that the two properties of excitation-closure and event-effectiveness hold on the union of regions underlying the SMs.

Initially, Algorithm 1 converts the minimal regions of the TS into a graph G, where intersecting regions define edges between the nodes of G (line 1). As long as G is not empty, the search of the maximal independent sets is performed on it by invoking the procedure MIS on G (MIS(G), line 5), storing the results in M (line 6) and removing the vertices selected at each iteration (line 7). In this way, each vertex will be included in one MIS solution. Notice that the maximal independent sets computed after the first one are not maximal with respect to the original graph G_0, because the MIS procedure is run on a subgraph of G_0 without the previously selected nodes. To be sure that we obtain maximal independent sets with respect to the original G_0, we expand to

Table 4. Adjacency matrix representing the edges (value 1) between vertices of the graph G created from the regions of the TS in Fig. 3.

	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}
r_1	1	1	1	0	0	1	0	0	1	1	1	0	0	1	0	1
r_2		1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
r_3			0	1	1	1	1	1	1	1	0	1	1	1	1	1
r_4				1	1	1	1	1	0	1	1	1	1	1	1	1
r_5					1	1	1	1	1	1	1	1	1	1	1	0
r_6						1	1	1	1	1	1	1	1	1	0	1
r_7							1	1	1	1	1	1	1	1	1	1
r_8								1	1	1	1	1	0	1	1	1
r_9									1	1	1	0	1	1	1	1
r_{10}										1	1	1	1	0	1	1
r_{11}											0	1	1	1	1	1
r_{12}												1	1	1	1	1
r_{13}													1	1	1	1
r_{14}														1	1	1
r_{15}															1	1
r_{16}																1

maximality the independent sets in M, by invoking the MIS procedure on each independent set $m \in M$ constrained to obtain a maximal independent set $\tilde{m} \supset m$ on G_0 (from line 9). Then from the maximal independent sets we obtain the induced state machines to be stored in F (from line 12). The motivation behind this step to enlarge the independent sets is to increase the number of regions for each SM, in order to widen the space of solutions for the successive optimizations of redundancy elimination and merging. The set of SMs derived from Algorithm 1 satisfies the EC and event-effectiveness properties because by construction each region is included in at least one independent set.

Consider a step-by-step execution of Algorithm 1 on the TS in Fig. 3. Initially one builds the graph G connecting the regions with common states (see Table 4).

Then the set of independent sets M is populated by the first cycle starting at line 5 as follows:

1. MIS $= \{r_1, r_6, r_{16}\}$,

   $M = \{\{r_1, r_6, r_{16}\}\}$,

   Nodes$(G) = \{r_1 - r_{17}\} \setminus \{r_1, r_6, r_{16}\}$.

2. MIS $= \{r_2, r_7\}$,

   $M = \{\{r_1, r_6, r_{16}\}, \{r_2, r_7\}\}$,

   Nodes$(G) = \{r_2, r_3, r_4, r_5, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\} \setminus \{r_2, r_7\} = \{r_3, r_4, r_5, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{17}\}$.

3. ....

The last cycle of the procedure checks, for each element m of M, if there is a larger independent set $\tilde{m} \supseteq m$ in G_0. The only independent sets which are extended are $\mathrm{SM}_4 = \{r_1, r_8, r_{14}\}$, $\mathrm{SM}_6 = \{r_1, r_5, r_{17}\}$ and $\mathrm{SM}_7 = \{r_1, r_9, r_{13}\}$.

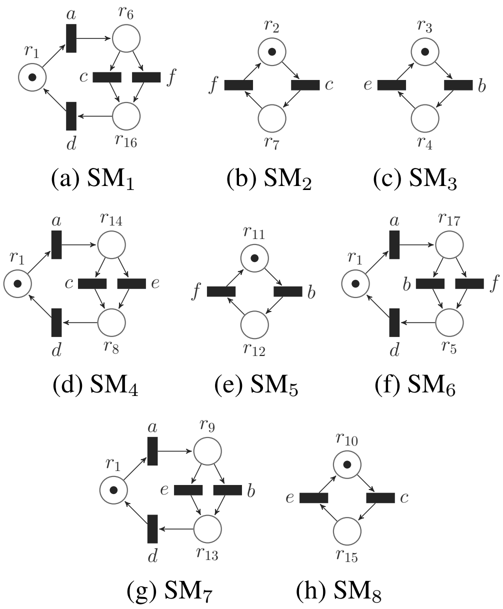


Fig. 5. All SMs created from the TS in Fig. 3.

$\mathrm{SM}_4 = \{r_1, r_8, r_{14}\}$ because $\{r_8, r_{14}\}$ is not a MIS on G_0.

$\mathrm{SM}_6 = \{r_1, r_5, r_{17}\}$ because $\{r_5, r_{17}\}$ is not a MIS on G_0.

$\mathrm{SM}_7 = \{r_1, r_9, r_{13}\}$ because $\{r_9, r_{13}\}$ is not a MIS on G_0.

Figure 5 shows the resultant SMs derived from the TS in Fig. 3.

3.2. Removal of redundant SMs. The set of SMs generated by Algorithm 1 may be redundant, i.e., it may contain a subset of SMs which still define an ECTS. We describe a greedy search algorithm to obtain an irredundant set of SMs: we order all the SMs by size and try to remove them one by one starting from the largest to the smallest, by checking that the union of the remaining regions satisfies *excitation-closure* and *event effectiveness*. If excitation-closure and event effectiveness are preserved, then the given SM can be removed. This algorithm is not optimal, because the removal of an SM may prevent the removal of a set of smaller SMs whose sum of places is greater than the number of places of the removed SM. However, this approach guarantees good performance having linear complexity in the number of SMs.

To check if the excitation-closure property is still valid after the removal of an SM, we consider the excitation sets and the pre-regions (see Table 3) for each event of the original transition system. We notice that SM_2 (whose nodes are $\{r_2, r_7\}$) affects only the events c and f (see Fig. 5(b)). Indeed, in the graph of SM_2 there is an edge from r_2 to r_7 under c because r_2 is a pre-region of

Table 5. Minimal regions of the transition system in Fig. 6.

Region	States of the TS
r_1	$\{s_1, s_3, s_5\}$
r_2	$\{s_2, s_4, s_6\}$
r_3	$\{s_7\}$
r_4	$\{s_8\}$
r_5	$\{s_1, s_2\}$
r_6	$\{s_3, s_4\}$
r_7	$\{s_5, s_6\}$

Table 6. Pre-regions for each event of the transition system in Fig. 6.

Event	Pre-regions
a	$\{r_1\}$
b	$\{r_1, r_7\}$
b'	$\{r_5\}$
c	$\{r_2, r_6\}$
d	$\{r_2, r_7\}$
e	$\{r_3\}$
f	$\{r_4\}$

c since c exits from $\{s_1, s_7\} \subseteq r_2 = \{s_0, s_1, s_3, s_5, s_7\}$, and r_7 is a post-region of c since c enters into $\{s_2, s_9\} \subseteq r_7 = \{s_2, s_4, s_6, s_8, s_9\}$; similarly, there is an edge from r_7 to r_2 under f because r_7 is a pre-region of f since f exits from $\{s_4, s_6\} \subseteq r_7\}$, and r_2 is a post-region of f since f enters into $\{s_3, s_5\} \subseteq r_2\}$. After the removal of event c, the intersection of the pre-regions is: $r_6 \cap r_{10} \cap r_{14} = \{s_1, s_4, s_6, s_7\} \cap \{s_0, s_1, s_5, s_6, s_7\} \cap \{s_1, s_3, s_4, s_7\} = \{s_1, s_7\} = \mathrm{ES}(c)$; after the removal of event f it is: $r_6 \cap r_{12} \cap r_{17} = \{s_1, s_4, s_6, s_7\} \cap \{s_1, s_2, s_4, s_6, s_8\} \cap \{s_4, s_6, s_7, s_9\} = \{s_4, s_6\} = \mathrm{ES}(f)$. For the other events the intersection of pre-regions is unchanged. Thus, SM_2 can be removed. Subsequently, following the same reasoning for the other events, also SM_1, SM_3 and SM_7 can be removed. Consequently, after the removal of the redundant SMs from the set shown in Fig. 5 only SM_4, SM_5, SM_6 and SM_8 are left.

3.3. Merge between regions preserving excitation-closure. We will use the transition system in Fig. 6 as running example to illustrate this subsection. By the procedure discussed so far, it can be decomposed as the synchronous product of two SMs shown in Fig. 7.

The third step of the procedure merges pairs of regions with the objective to minimize the size of the sets of SMs: edges carrying labels are removed and, in consequence, the two nodes connected to them are merged decreasing their number. For example, in Fig. 7, both SMs contain an instance of label e connected by regions r_3 and r_4. This means that an edge carrying label e can be removed in one of the SMs. The result of removing the edge with label e in SM_b and merging the regions r_3^2 and r_4^2 replacing them with the region r_{34} is shown in Fig. 8.

All instances of a region except one can be removed, because removing all of them would change the set of regions used for checking the excitation-closure property, whereas keeping at least one guarantees the preservation of the property.

We formulated the merging problem as solving an instance of SAT. We now describe the problem encoding. We introduce next three types of SAT clauses required to represent the problem.

1. A set of SAT clauses states that we cannot remove all instances of a given region r from all the SMs where it appears. If we define r_i^k to be true if region r_i appears in SM_k, the constraint that each region must appear in at least one SM is modelled by the following equality:

   $$\forall_i \exists_k \; r_i^k = 1$$

   which is then encoded with SAT clauses.

   Therefore, the SAT model will contain a clause for each region r_i to represent all instance of a given r_i in all SMs. In the running example the clauses will be

   $$r_1^1 \wedge r_2^1 \wedge (r_3^1 \vee r_3^2) \wedge (r_4^1 \vee r_4^2) \wedge r_5^2 \wedge r_6^2 \wedge r_7^2.$$

2. Another set of SAT clauses states, for each SM, that if a label on a given edge is removed, then also the two regions connected by the edge are removed, i.e., if label l is on the edge connection regions r_1 and r_2, the clause template is $(r_1 \vee r_2) \to l$, i.e., $(\neg r_1 \wedge \neg r_2) \vee l$, i.e., $(\neg r_1 \vee l) \wedge (\neg r_2 \vee l)$. In the running example, the clauses for SM_a are the following (l replaced by the actual labels):

   $$(\neg r_1^1 \vee a) \wedge (\neg r_2^1 \vee a) \wedge (\neg r_1^1 \vee b) \wedge (\neg r_3^1 \vee b) \wedge (\neg r_1^1 \vee c) \wedge (\neg r_2^1 \vee c) \wedge (\neg r_2^1 \vee d) \wedge (\neg r_4^1 \vee d) \wedge (\neg r_3^1 \vee e) \wedge (\neg r_4^1 \vee e) \wedge (\neg r_1^1 \vee f) \wedge (\neg r_4^1 \vee f);$$

   the clauses for SM_2 are (l replaced by the actual labels):

   $$(\neg r_5^2 \vee b') \wedge (\neg r_6^2 \vee b') \wedge (\neg r_6^2 \vee c) \wedge (\neg r_7^2 \vee c) \wedge (\neg r_4^2 \vee d) \wedge (\neg r_7^2 \vee d) \wedge (\neg r_3^2 \vee e) \wedge (\neg r_4^2 \vee e) \wedge (\neg r_4^2 \vee f) \wedge (\neg r_5^2 \vee f).$$

3. Finally, we must express the optimization objective: keep the minimum number of labels needed to satisfy the excitation-closure property. This is expressed by

   $$\min\Big(\sum_{\forall k \forall j} l_j^k\Big) \tag{1}$$

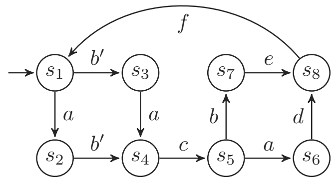
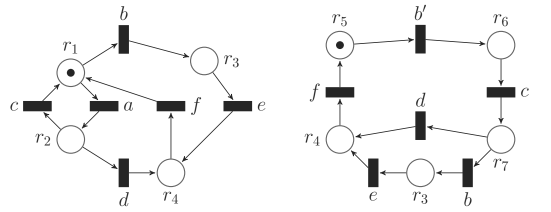
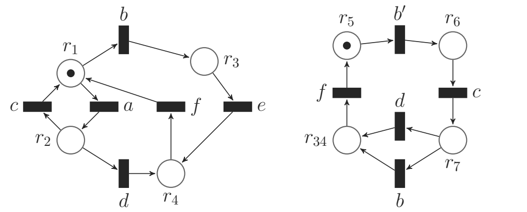


Fig. 6. ECTS.



(a) SM_a (b) SM_b

Fig. 7. SMs obtained with the MIS solver from the TS of Fig. 6.



(a) SM_a (b) SM_b

Fig. 8. SMs of Fig. 7 after the removal of label e in SM_b.

where l_j^k is true if there is an instance of label j in SM_k.

Setting x as the total number of labels in all SMs (in the running example $x = \sum l_j^k = 6 + 6 = 12$), this constraint is rewritten as

$$\sum_{\forall k \forall j} l_j^k \leq x. \tag{2}$$

Then (2) is converted into a set of SAT clauses using the library PBLib (Philipp and Steinke, 2015). The first assignment of x yields a trivially true SAT instance because it corresponds to the initial situation, as stated by

$$\sum_{\forall k \forall j} l_j^k = x. \tag{3}$$

Therefore, a solution of Eqn. (1) can be found by solving a sequence of SAT instances whose clauses are the ones previously defined (clauses to represent regions, clauses encoding the relation between regions and labels, and clauses from the conversion of Eqn. (2), and where x decreases from the initial largest value down, until an UNSAT model[4] is reached. The solution of the last satisfiable SAT instance encountered represents the best decomposition of the initial transition system. As a matter of fact, the linear search on x is sped up by transforming it into a logarithmic binary search on x (in the running example, we solve for $x = 12$, $x = 6$, $x = 9$ till we converge for $x = 11$).

At the end, according to the SAT solution, the SMs are restructured by removing arcs and nodes to be deleted and adding merged nodes, and redirecting arcs as appropriate. In the running example, in SM_b we merge the nodes r_3, r_4 into node r_{34}, remove the edge labeled e between the deleted nodes r_3 and r_4, and redirect to r_{34} the edges pointing to r_3 or r_4.

4. Composition of SMs and equivalence to original TS

Intuitively, the SMs derived from an LTS interact running in parallel with the same rules of the synchronous product of transition systems (see Definition 4). Indeed, if we interpret the reachability graphs of the SMs as LTSs and execute the synchronous product deriving a single LTS which models the interaction of the SMs, it turns out that the result of the composition is equivalent to the original LTS, as proved in Appendix. For example, consider the composition of reachability graphs of SMs SM_4 and SM_5 in Fig. 9, it generates a superset of behaviors of the original LTS in Fig. 3, it produces the sequence "acbdaefd" which is in the original LTS, but also new behaviors, like the sequences starting by the event b (e.g., "bacfd"), which are not in the original LTS because some constraints of the original LTS are missing; indeed, these two SMs are not enough to satisfy the excitation-closure property, whereas event effectiveness is satisfied by them because all events are included in the composition. In this example, by considering a single SM, even event effectiveness may fail, when some events are hidden because they are completely inside or outside some regions: e.g., considering only SM_4, event effectiveness is not satisfied because the events b and f are missing (in this case sequences containing the aforementioned events cannot be produced, for example the previously cited sequence "acbdaefd"). The composition of SMs can exhibit these hidden behaviors by including new regions. For example, the composition of SM_4 with SM_5 includes two new regions r_{11} and r_{12} so that the events b and f show up in the composition.

Theorem 2. *Given an excitation-closed set $\{SM_1, \ldots, SM_n\}$ of SMs derived from the ECTS TS, there is a bisimulation B such that TS $\sim_B ||_{i=1,\ldots,n} RG(SM_i)$.*

[4] A model for which no satisfiable SAT solution can be found.

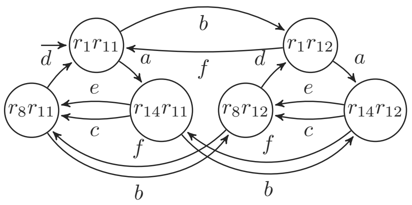


Fig. 9. Composition between RG(SM_4) and RG(SM_5) of Fig. 5.

For a proof, see Appendix.

Theorem 2 states that, given a set of SMs, the excitation-closure and event effectiveness of the union of their regions is a necessary and sufficient condition to guarantee that their synchronous product is equivalent to the original TS.

5. Experimental results

We implemented the procedure described in Sec. 3 and performed experiments on an Intel core running at 2.80 GHz with 16 GB of RAM. Our software is written in C++ and uses *PBLib* (Philipp and Steinke, 2015) for the resolution of SAT. The resolution of the MIS problem is performed by the *NetworkX* library (Hagberg *et al.*, 2008). For our tests, we used two sets of benchmarks, both from the world of asynchronous controllers: the first set (the same as in the work of Cortadella *et al.* (1995)), with smaller transition systems is listed in the first rows of Table 7 and denoted as “Small-sized set”; the second one containing large transition systems is listed in the second part of Table 7, denoted as “Large-sized set.” “Large-sized set” contains parametrized controllers (`art_m_n`) from the work of Carmona *et al.* (2006) and the biggest parametrized controller computable by our software (`pparb_2_6`) from the set of Khomenko *et al.* (2004). Differently from the miscellaneous small benchmarks, the large set mainly contains controllers with m pipelines (`art_m_n, pparb_m_n`), a suitable type of input for our algorithm, being highly concurrent. However there is also a case with a completely sequential circuit (`seq40`) in order to show also the worst case where each region contains only one state.

The software used for the synthesis of Petri nets is Petrify[5] (Cortadella *et al.*, 1997). Even if the core of this software did not change for many years, it is still a reference point for PN synthesis using the theory of regions. Genet (Carmona *et al.*, 2009b) is the only alternative used nowadays (still based on the theory of regions).

Table 7 shows the absolute and relative runtimes of the steps of the flow: region generation, decomposition into SMs, irredundancy, place merging. The generation of minimal regions is the dominating operation taking more than 60% of the overall time spent; it is exponential in the number of events and with an increase in the input dimensions it becomes a bottleneck shadowing the remaining computations. However, it is still possible to decompose quite large transition systems with about 10^6 states and $3 \cdot 10^6$ transitions.

Table 8 compares the states and transitions of transition systems vs. the places/transitions/crossing arcs of the Petri nets derived by Petrify (columns under PN), and vs. our product of state machines for the first benchmark set. The number of crossing arcs is reported by the *dot* algorithm of *graphviz* (Gansner *et al.*, 1993) and can be considered as a metric of structural simplicity of the model (i.e., fewer crossings implies a simpler structure). Our results from synchronized state machines have similar sizes compared with those from Petri nets, but they have fewer crossings, which is a significant advantage in supporting a visual representation for “large systems.” Therefore the plots, in a two-dimensional graphical representation of synchronizing SMs, are substantially more *readable* than the ones of Petri nets: see the inputs *intel_edge* and *pe-rcv-ifc* witnessing that peaks of edge crossings are avoided. The example *master-read* instead is an impressive case of how our decomposition tames the state explosion of the original transition system derived from a highly concurrent environment, since from 8932 states we go down to 8 SMs with an average number of 5 states each.

We implemented also an exact search of all SMs derived from the original TS, to gauge our heuristics, when it is possible to find a near exact solution. We compare the times taken by the exact and heuristic SM generation steps: the exponential behaviour of the exact algorithm makes it hardly affordable for about 15 regions and run out of 16 GB of memory for more than 20 regions (Table 9). Instead, the approximate algorithms presented in Section 3 can handle very large transition systems. Even though the result is not guaranteed to be a minimum one, the irredundancy procedure guarantees a form of minimality, yielding a compact representation that avoids state explosion and exhibits concurrency explicitly.

5.1. Creation of a new mixed strategy. We performed a set of three experiments on top of those reported by Teren *et al.* (2021). The first experiment consists in the execution of the exact algorithm for both phases: search of the new SMs and the removal of redundant ones. Previously an experiment had been performed running the exact algorithm only to generate the SMs; as reported by Teren *et al.* (2021), the execution of the exact algorithm for this task followed by an approximate removal of SMs requires a lot of effort without bringing interesting results. The removal of redundant SMs with an exact algorithm too provides a lower bound of the

[5] Version 5.2, May 2019.

Table 7. TS statistics and CPU time for each decomposition step including the time spent to generate the regions.

	Input	States	Transitions	Events	Regions	Time region generation [s]	Time decomposition [s]	Time Greedy [s]	Time Merge [s]	Total time [s]	Time region generation [%]	Time decomposition [%]	Time Greedy [%]	Time Merge [%]
"Small-sized" set	alloc-outbound	17	18	14	15	0.00	0.25	0.00	0.06	0.31	0.36	80.36	0.07	19.21
	clock	10	10	4	11	0.01	0.20	0.00	0.03	0.24	3.02	85.71	0.13	11.14
	dff	20	24	7	20	0.29	0.20	0.00	0.77	1.27	23.28	15.50	0.08	61.14
	espinalt	27	31	20	23	0.00	0.21	0.00	0.49	0.70	0.37	29.54	0.07	70.02
	fair_arb	13	20	8	11	0.02	0.20	0.00	0.03	0.25	8.80	80.41	0.04	10.74
	future	36	44	16	19	0.03	0.21	0.00	0.11	0.35	9.40	60.35	0.20	30.05
	intel_div3	8	8	4	8	0.00	0.23	0.00	0.01	0.24	0.75	94.73	0.04	4.48
	intel_edge	28	36	6	27	1.60	0.20	0.00	1.30	3.11	51.58	6.41	0.14	41.86
	isend	53	66	15	128	57.67	0.31	0.32	1.04	59.33	97.21	0.51	0.53	1.75
	lin_edac93	20	28	8	10	0.00	0.19	0.00	0.01	0.21	1.16	93.38	0.10	5.36
	master-read	8932	36	26	33	6.71	0.53	0.12	1.03	8.39	80.00	6.28	1.45	12.28
	pe-rcv-ifc	46	62	16	7	8.80	0.19	0.00	1.21	10.21	86.20	1.90	0.01	11.90
	pulse	12	12	6	33	0.00	0.19	0.00	0.01	0.19	0.36	96.62	0.05	2.96
	rcv-setup	14	17	10	11	0.00	0.19	0.00	0.04	0.23	1.41	81.36	0.09	17.14
	vme_read	255	668	26	44	0.53	0.20	0.01	15.17	15.91	3.36	1.23	0.04	95.37
	vme_write	821	2907	30	51	2.78	0.24	0.03	30.03	33.08	8.39	0.73	0.10	90.77
"Large-sized" set	art_3_10	32000	93200	60	64	154.96	2.02	0.07	1.18	158.23	97.93	1.28	0.04	0.75
	art_3_11	42592	124388	66	70	105.51	3.02	0.31	1.71	110.55	95.44	2.73	0.28	1.55
	art_3_12	55296	161856	72	76	133.58	4.21	0.39	2.06	140.24	95.25	3.01	0.27	1.47
	art_3_13	70304	206180	78	83	1153.20	6.97	1.52	2.84	1164.54	99.03	0.60	0.13	0.24
	art_3_14	87808	257936	84	88	2062.91	9.09	0.94	3.49	2076.43	99.35	0.44	0.05	0.17
	art_3_15	108000	317700	90	94	2240.17	10.20	0.77	4.11	2255.25	99.33	0.45	0.03	0.18
	art_3_16	131072	386048	96	100	971.23	12.70	0.56	5.75	990.23	98.08	1.28	0.06	0.58
	art_3_17	157216	463556	102	108	6068.14	15.84	4.81	0.43	6089.22	99.65	0.26	0.08	0.01
	art_3_18	186624	550800	108	112	5133.03	16.57	0.95	0.47	5151.01	99.65	0.32	0.02	0.01
	art_3_19	219488	648356	114	118	904.41	18.84	1.11	0.57	924.93	97.78	2.04	0.12	0.06
	art_3_20	256000	756800	120	124	11 915.93	30.30	1.97	0.65	11 948.85	99.72	0.25	0.02	0.01
	art_4_04	32768	120832	32	38	65.30	2.23	0.55	0.27	68.35	95.54	3.26	0.81	0.40
	art_4_05	80000	300000	40	46	232.23	5.95	0.49	0.67	239.34	97.03	2.49	0.21	0.28
	art_4_06	165888	628992	48	55	768.95	16.57	5.61	0.96	792.09	97.08	2.09	0.71	0.12
	art_4_07	307328	1174432	56	62	2151.37	28.12	4.53	1.06	2185.07	98.46	1.29	0.21	0.05
	art_4_08	524288	2015232	64	70	3373.92	61.78	10.88	1.60	3448.17	97.85	1.79	0.32	0.05
	art_4_09	839808	3242592	72	78	4293.87	57.98	4.95	2.07	4358.87	98.51	1.33	0.11	0.05
	seq_40	164	164	164	164	0.04	0.23	0.00	1.47	1.75	2.54	13.19	0.01	84.26
	pparb_2_6	69632	321536	34	77	886.54	25.51	30.27	8.32	950.65	93.26	2.68	3.18	0.88
	AVERAGE										63.70	20.69	0.28	15.33

Table 8. Number of places (P), transitions (T) and arc crossings (C) of the original transition systems vs. derived Petri nets vs. product of SMs and SM details.

Input	Size comparison													SM details				
	TS		PN			PN*[6a]			Synchronizing SMs			Number of SMs	Avg. places per SM	Avg. alphabet per SM	Places largest SM	Alphabet largest SM
	States	T	P	T	C	P	T	C	P	T	C					
alloc-outbound	21	18	14	14	3	17	18	0	17	21	0	2	8.50	10.50	10	11
clock	10	10	8	5	4	10	10	0	11	15	0	3	3.67	5.00	4	4
dff	20	24	13	14	21	20	20	0	25	41	0	3	8.33	13.33	13	7
espinalt	27	31	22	20	5	27	25	1	29	32	0	3	9.33	11.00	11	13
fair_arb	13	20	11	10	4	11	10	4	12	18	0	2	6.00	9.00	6	6
future	36	44	18	16	1	30	28	0	21	22	0	3	7.00	7.33	13	14
intel_div3	8	8	7	5	2	8	8	0	10	11	0	2	5.00	5.50	6	4
intel_edge	28	36	11	15	22	21	30	56	35	68	1	4	8.50	16.75	13	6
isend	53	66	25	27	106	54	43	5	80	138	4	13	6.31	11.85	12	11
lin_edac93	20	28	10	8	1	14	12	0	13	14	0	3	4.33	4.67	5	6
master-read	8932	36226	33	26	0	33	26	0	38	38	0	8	4.75	4.75	10	10
pe-rcv-ifc	46	62	23	20	96	43	37	13	39	57	2	2	19.00	28.50	21	13
pulse	12	12	7	6	2	12	12	0	7	10	0	2	3.50	5.00	3	6
rcv-setup	14	17	10	10	5	14	14	4	12	14	0	2	6.00	7.00	9	10
vme_read	255	668	38	29	18	41	32	2	50	67	1	9	6.11	7.67	12	13
vme_write	821	2907	46	33	31	49	36	6	57	74	1	11	6.18	7.36	9	11

[a] PN* is a representation of the PN after splitting disconnected ERs, thus producing multiple labels (transitions) for the same event. This results in a PN with more transitions and a simpler structure.

Table 9. CPU time and results of the exact decomposition algorithm.

Input	Decomposition [s]	Greedy [s]	Merge [s]	States after decomposition	States after greedy	States after merge	Trans. after decomposition	Trans. after greedy	Trans. after merge	Number of regions TS
alloc-outbound	14.01	0.0009	0.06	42	21	17	50	25	21	15
clock	0.55	0.0003	0.02	18	14	11	22	18	15	11
fair_arb	0.58	0.0007	0.03	24	12	12	36	18	18	11
future	1881.00	0.0012	0.10	41	29	22	43	30	23	19
intel_div3	0.21	0.0001	0.01	12	12	10	13	13	11	8
lin_edac93	0.33	0.0002	0.01	13	13	13	14	14	14	10
pulse	0.20	0.0000	0.01	7	7	7	10	10	10	7
rcv-setup	0.36	0.0002	0.04	18	18	12	22	22	14	11

Table 10. Number of final SMs derived using an approximate algorithm for the search of new SMs and different approaches for the removal of redundant SMs.

	Greedy algorithm (approximate)	Exact algorithm	Mixed strategy
alloc-outbound	2	2	2
clock	3	3	3
dff	3	3	3
espinalt	3	3	3
fair_arb	2	2	2
future	3	3	3
intel_div3	2	2	2
intel_edge	4	3	3
isend	13	-	13
lin_edac93	3	3	3
master-read	8	8	8
pe-rcv-ifc	2	2	2
pulse	2	2	2
rcv-setup	2	2	2
vme_read	9	9	9
vme_write	11	10	10
AVERAGE	4,5		4,375

decomposition (in terms of the number of final SMs), since both steps are performed with an exact algorithm; moreover, it hits the scalability threshold of the exact algorithm, since the computation of many benchmarks did not finish. Consequently, the fully exact computation can be performed only on very tiny benchmarks where the number of SM combinations is very restricted. Notice that for each available result of the completely exact flow also the completely approximate one, and the combination of exact SM search and approximate SM removal, found the same number of SMs.

The second experiment consists in the execution of the approximate SM search followed by an exact algorithm for SM removal (column "Exact algorithm" in Table 10). For some benchmarks we got better results compared with a completely approximate approach (`intel_edge`, `vme_write`), but in other cases (`isend`) the computation did not finish due to the high number of SMs available for the removal algorithm (more than 50). Indeed, the complexity of the exact removal of redundant SMs is $O(2^n)$, where n is the number of SMs.

The third experiment explored a mixed strategy and represents the main algorithmic improvement with the respect to the previous conference version. This approach is based on the number of derived SMs after the approximate search, given that between the two computation steps we know the exact number of derived SMs. The mixed strategy works as follows: Let n be the initial number of SMs found with the approximate search; then for a "small" n we apply the exact removal algorithm whose computational times are affordable; otherwise we apply the approximate removal algorithm. In our experiments we have chosen $n = 20$. Column "Mixed strategy" in Table 10 represents the result of this combination between the exact and approximate algorithms for the removal of redundant SMs. On the average, this combination yields slightly better results than the previously proposed completely approximate solution (column "Greedy algorithm"), but without significant improvements.

6. Conclusions

In this paper, we described a method for the decomposition of transition systems into a synchronous composition of state machines (a restricted class of Petri nets). We provided a complete exposition of the underlying theory, clarifying the computational steps with detailed running examples. The experimental results demonstrate that the decomposition algorithm can be run on transition systems with up to one million states; therefore, it is suitable to handle real cases. In this extended version, we reported also a new mixed strategy leveraging in some cases the exact algorithm for the removal of redundant SMs, which allowed us to improve the decomposition results by decreasing the average number of SMs in our benchmark set from 4,5 to 4,375.

Since the generation of minimal regions is currently a computational bottleneck, future work will address this limitation, while it will leverage the improvements in efficiency of last-generation MIS and SAT solvers, and the power of HPC[7] since the generation of minimal regions is highly parallelizable. HPC can be exploited also in other steps of the decomposition algorithm, e.g., different MIS computations could be performed simultaneously applying constraints to each parallel computation (e.g., assigning a state to each thread and forcing it to be in the MIS result).

As future work, we want to apply this decomposition paradigm to process mining. Rather than synthesizing intricate "spaghetti" Petri nets from logs, we aim at distilling loosely coupled concurrent threads (SMs) that can be easily visualized, analyzed and optimized individually, while preserving the synchronization with the other threads. Optionally, a new Petri net can be obtained by composing back the optimized threads and imposing some structural constraints, e.g., to be a free-choice Petri net, thus providing a tight approximation of the original behavior with a simpler structure.

[7] High performance computing: aggregation of computing power to solve problems too complex to be solved by a normal desktop computer or workstation.

References

Badouel, E., Bernardinello, L. and Darondeau, P. (2015). *Petri Net Synthesis*, Springer, Berlin.

Benini, L., De Micheli, G. and Macii, E. (2001). Designing low-power circuits: Practical recipes, *IEEE Circuits and Systems Magazine* **1**(1): 6–25.

Boros, E. and Hammer, P.L. (2002). Pseudo-Boolean optimization, *Discrete Applied Mathematics* **123**(1–3): 155–225.

Carmona, J., Colom, J-M, Cortadella, J. and García-Vallés, F. (2006). Synthesis of asynchronous controllers using integer linear programming, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **25**(9): 1637–1651.

Carmona, J., Cortadella, J. and Kishinevsky, M. (2009a). Divide-and-conquer strategies for process mining, *International Conference on Business Process Management, Ulm, Germany*, pp. 327–343.

Carmona, J., Cortadella, J. and Kishinevsky, M. (2009b). Genet: A tool for the synthesis and mining of Petri nets, *9th International Conference on Application of Concurrency to System Design, Augsburg, Germany*, pp. 181–185.

Carmona, J., Cortadella, J. and Kishinevsky, M. (2009c). New region-based algorithms for deriving bounded Petri nets, *IEEE Transactions on Computers* **59**(3): 371–384.

Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L. and Yakovlev, A. (1997). Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers, *IEICE Transactions on Information and Systems* **80**(3): 315–325.

Cortadella, J., Kishinevsky, M., Lavagno, L. and Yakovlev, A. (1995). Synthesizing Petri nets from state-based models, *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD), San Jose, USA*, pp. 164–171.

Cortadella, J., Kishinevsky, M., Lavagno, L. and Yakovlev, A. (1998). Deriving Petri nets from finite transition systems, *IEEE Transactions on Computers* **47**(8): 859–882.

de San Pedro, J. and Cortadella, J. (2016). Mining structured Petri nets for the visualization of process behavior, *Proceedings of the 31st Annual ACM Symposium on Applied Computing, New York, USA*, pp. 839–846.

Desel, J. (1995). *Free Choice Petri Nets*, Cambridge University Press, Cambridge.

Ehrenfeucht, A. and Rozenberg, G. (1990). Partial (set) 2-structures, *Acta Informatica* **27**(4): 343–368.

Gansner, E., Koutsofios, E., North, S. and Vo, K.-P. (1993). A technique for drawing directed graphs, *IEEE Transactions on Software Engineering* **19**(3): 214–230.

Hagberg, A., Swart, P. and S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX, *Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, USA*, pp. 11–15.

Kalenkova, A.A., Lomazova, I.A. and Van der Aalst, W.M. (2014). Process model discovery: A method based on transition system decomposition, *International Conference on Applications and Theory of Petri Nets and Concurrency, Tunis, Tunisia*, pp. 71–90.

Kemper, P. and Bause, F. (1992). An efficient polynomial-time algorithm to decide liveness and boundedness of free-choice nets, *in* K. Jensen (Ed.), *Application and Theory of Petri Nets,* Springer, Berlin/Heidelberg, pp. 263–278.

Khomenko, V., Koutny, M. and Yakovlev, A. (2004). Detecting state encoding conflicts in STG unfoldings using SAT, *Fundamenta Informaticae* **62**(2): 221–241.

Mattheakis, P.M. (2013). *Logic Synthesis of Concurrent Controller Specifications*, PhD thesis, University of Crete, Rethymnon, `https://thesis.ekt.gr/thesisBookReader/id/29912#page/1/mode/2up`.

Mokhov, A., Cortadella, J. and de Gennaro, A. (2017). Process windows, *17th International Conference on Application of Concurrency to System Design (ACSD), Zaragoza, Spain*, pp. 86–95.

Murata, T. (1989). Petri nets: Properties, analysis and applications, *Proceedings of the IEEE* **77**(4): 541–580.

Philipp, T. and Steinke, P. (2015). PBLib—A library for encoding pseudo-Boolean constraints into CNF, *in* M. Heule and S. Weaver (Eds), *Theory and Applications of Satisfiability Testing, SAT 2015*, Lecture Notes in Computer Science, Vol. 9340, Springer, Cham, pp. 9–16.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*, Wiley, Amsterdam.

Taibi, D. and Systä, K. (2019). From monolithic systems to microservices: A decomposition framework based on process mining, *Proceedings of the 9th International Conference on Cloud Computing and Services Science—CLOSER, Heraklion, Greece*, pp. 153–164, DOI: 10.5220/0007755901530164.

Teren, V., Cortadella, J. and Villa, T. (2021). Decomposition of transition systems into sets of synchronizing state machines, *2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, Italy*, pp. 77–81, DOI: 10.1109/DSD53832.2021.00021.

Van der Aalst, W.M. (2012). Decomposing process mining problems using passages, *International Conference on Application and Theory of Petri Nets and Concurrency, Hamburg, Germany*, pp. 72–91.

Van der Aalst, W.M. (2013). Decomposing Petri nets for process mining: A generic approach, *Distributed and Parallel Databases* **31**(4): 471–507.

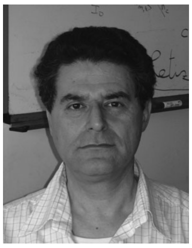
Van der Aalst, W.M., Rubin, V., Verbeek, H., van Dongen, B.F., Kindler, E. and Günther, C.W. (2010). Process mining: A two-step approach to balance between underfitting and overfitting, *Software & Systems Modeling* **9**(1): 87.

Verbeek, H. and Van der Aalst, W.M. (2014). Decomposed process mining: The ILP case, *International Conference on Business Process Management, Eindhoven, The Netherlands*, pp. 264–276.

Wojnakowski, M., Wiśniewski, R., Bazydło, G. and Popławski, M. (2021). Analysis of safeness in a Petri net-based specification of the control part of cyber-physical systems, *International Journal of Applied Mathematics and Computer Science* **31**(4): 647–657, DOI: 10.34768/amcs-2021-0045.

Viktor Teren was born in 1993. He is currently working toward his PhD in computer science at the University of Verona. At present his research interest is focused on the decomposition of transition systems into different subclasses of interoperable Petri nets.

Jordi Cortadella (IEEE Fellow) received his PhD degree in computer science from Universitat Politècnica de Catalunya, Barcelona, Spain, in 1987. He is a professor at the Computer Science Department there. His current research interests include formal methods and computer-aided design of VLSI systems, with a special emphasis on asynchronous circuits, concurrent systems, and logic synthesis. Prof. Cortadella received Best Paper Awards at *ASYNC 2004* and *2016*, *DAC* in 2004, *ACSD* in 2009, and *FPGA* in 2020. He has served on technical committees of several international conferences in the field of design automation and concurrent systems.



Tiziano Villa received his PhD in electrical engineering and computer in 1995 from the University of California, Berkeley. Since 2006 he has been a professor with the Department of Computer Science (DI), Università di Verona, Italy. His research interests are in formal methods for electronic design automation, including logic synthesis, formal verification, models of computation, discrete-event dynamic systems, cyber-physical systems. He has co-authored three books: *Synthesis of Finite State Machines: Functional Optimization* (Kluwer/Springer), *Synthesis of Finite State Machines: Logic Optimization* (Kluwer/Springer), *The Unknown Component Problem: Theory and Applications* (Springer), and has co-edited the book *Coordination Control of Distributed Systems* (Springer).

Appendix

Proof of Theorem 2

The equivalence between an ECTS and the derived set of SMs is proved by defining a bisimulation between the original TS, defined as $\mathrm{TS} = (S, E, T, s_0)$, and the synchronous product of the reachability graphs of the derived state machines $\mathrm{RG}(\mathrm{SM}_1)||\mathrm{RG}(\mathrm{SM}_2)||\dots||\mathrm{RG}(\mathrm{SM}_n)$, denoted by $||_{i=1,\dots,n}\mathrm{RG}(\mathrm{SM}_i) = (S_{||}, E, T_{||}, s_{0,||})$. Notice that each $\mathrm{RG}(\mathrm{SM}_i) = (R_i, E_i, T_i, r_{0,i})$, with $T_i \subseteq R_i \times E_i \times R_i$, is defined on a subset E_i of events of TS, its states r_i correspond to regions of the states of TS, and the initial state is a region $r_{0,i}$ containing the initial state of TS. To prove the existence of a bisimulation, we require that the union of $\mathrm{RG}(\mathrm{SM}_i)$ satisfies ECTS, where event-effectiveness guarantees that $\cup E_i = E$, and excitation-closure guarantees that the two transition systems simulate each other, i.e., the transition relations allow to match each other's moves.

Proof. We define the binary relation B as follows:

$$(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B \iff s_j \in \bigcap_{i=1}^{n} r_{j,i},$$

where $s_j \in S$ and $r_{j,i} \in R_i$, for $i \in \{1, \dots, n\}$.

Notice that writing $(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B \iff \{s_j\} = \bigcap_{i=1}^{n} r_{j,i}$ would be wrong, because the intersection of regions could have two or more bisimilar (i.e., behaviourally equivalent) states, as in the TS $s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_3 \xrightarrow{b} s_0$.

A region $r_{j,i}$ may appear in two or more sets of regions R_i. Now we prove that B is a bisimulation in three steps:

1. $(s_0, (r_{0,1}, r_{0,2}, \dots, r_{0,n})) \in B$.
2. If $(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B$ and $(s_j, e, s_k) \in T$, then there is $(r_{k,1}, r_{k,2}, \dots, r_{k,n}) \in S_{||}$ such that $((r_{j,1}, r_{j,2}, \dots, r_{j,n}), e, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in T_{||}$ and $(s_k, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in B$.
3. If $(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B$ and, moreover, $((r_{j,1}, r_{j,2}, \dots, r_{j,n}), e, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in T_{||}$, then there is $s_k \in S$ such that $(s_j, e, s_k) \in T$ and $(s_k, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in B$.

Let us now proceed with the proof.

1. Since TS has a unique initial state s_0, each state machine SM_i has exactly one initial region $r_{0,i}$ such that $s_0 \in r_{0,i}$ because all the regions of an SM are disjoint. Therefore, $s_0 \in \bigcap_{i=1}^{n} r_{0,i}$ and we have that $(s_0, (r_{0,1}, r_{0,2}, \dots, r_{0,n})) \in B$.
2. Since $(s_j, e, s_k) \in T$ and, moreover, $(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B$, we get $s_j \in \bigcap_{i=1}^{n} r_{j,i}$. Now we will prove that there is s_k such that $s_k \in \bigcap_{i=1}^{n} r_{k,i}$, so that we can have $(s_k, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in B$.

   Since e is enabled in s_j, none of the $r_{j,i}$'s can be a post-region of e. If one $r_{j,i}$ in $\{r_{j,1}, \dots, r_{j,n}\}$ were a post-region, then $s_j \notin \bigcap_{i=1}^{n} r_{j,i}$. Therefore, the following three cases can be distinguished for each $r_{j,i} \in \{r_{j,1}, r_{j,2}, \dots, r_{j,n}\}$:

   - e is not an event of SM_i. Thus, $r_{k,i} = r_{j,i}$.
   - e is an event of SM_i and $r_{j,i}$ is a no-cross region for e. Thus, $r_{k,i} = r_{j,i}$.
   - e is an event of SM_i and $r_{j,i}$ is a pre-region of e. Thus, $r_{k,i} \neq r_{j,i}$ is a post-region of e.

For the first and second cases, SM_i will not change state and TS will not change region when moving from s_j to s_k. Therefore, $s_k \in r_{j,i} = r_{k,i}$.

For the third case, e will exit $r_{j,i}$ and will enter $r_{k,i}$ in *TS*, which means that $s_k \in r_{k,i}$. Therefore, $((r_{j,1}, r_{j,2}, \dots, r_{j,n}), e, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in T_{||}$.

For all cases we have that $s_k \in r_{k,i}$ and therefore $s_k \in \bigcap_{i=1}^{n} r_{k,i}$.

3. Since $(s_j, (r_{j,1}, r_{j,2}, \dots, r_{j,n})) \in B$, we have that $s_j \in \bigcap_{i=1}^{n} r_{j,i}$. Given the existence of the transition $((r_{j,1}, r_{j,2}, \dots, r_{j,n}), e, (r_{k,1}, r_{k,2}, \dots, r_{k,n}))$, and knowing that the EC property holds, we know that $s_j \in \bigcap_{i=1}^{n} r_{j,i} \subseteq \mathrm{ES}(e)$. The latter inequality holds because by Theorem 1 we have (i) $\forall i, i = 1, \dots, n$, label e appears once in SM_i or it does not appear, and (ii) $\forall i, i = 1, \dots, n$, if label e appears in SM_i then $r_{j,i} \in ({}^{\circ}e \cap R)$, by which $\bigcup_{i=1}^{n} \{r_{j,i}\} \supseteq \bigcup_{r \in ({}^{\circ}e \cap R)} \{r\}$ and so by intersection of the regions considered as sets of states $\bigcap_{i=1}^{n} r_{j,i} \subseteq \bigcap_{r \in ({}^{\circ}e \cap R)} r = \mathrm{ES}(e)$.

Therefore, there is s_k such that $(s_j, e, s_k) \in T$. We can also see that $s_k \in \bigcap_{i=1}^{n} r_{k,i}$, using the same reasoning as in Step 2, since all the pre-regions $r_{j,i}$ of e in $\{r_{j,1}, \dots, r_{j,n}\}$ are exited by entering $r_{k,i}$, whereas the no-crossing regions remain the same. We can then conclude that $(s_k, (r_{k,1}, r_{k,2}, \dots, r_{k,n})) \in B$.

■

Received: 28 January 2022
Revised: 24 June 2022
Re-revised: 6 October 2022
Accepted: 25 October 2022