Deriving Petri Nets from Finite Transition Systems

Jordi Cortadella, *Member*, *IEEE*, Michael Kishinevsky, *Senior Member*, *IEEE*, Luciano Lavagno, *Member*, *IEEE*, and Alexandre Yakovlev, *Member*, *IEEE*

Abstract—This paper presents a novel method to derive a Petri Net from any specification model that can be mapped into a state-based representation with arcs labeled with symbols from an alphabet of events (a Transition System, TS). The method is based on the theory of regions for *Elementary Transition Systems* (ETS). Previous work has shown that, for any ETS, there exists a Petri Net with minimum transition count (one transition for each label) with a reachability graph isomorphic to the original Transition System. Our method extends and implements that theory by using the following three mechanisms that provide a framework for synthesis of *safe* Petri Nets from arbitrary TSs. First, the requirement of isomorphism is relaxed to *bisimulation* of TSs, thus extending the class of synthesizable TSs to a new class called *Excitation-Closed Transition Systems* (ECTS). Second, for the first time, we propose a method of PN synthesis for an arbitrary TS based on mapping a TS event into a set of transition labels in a PN. Third, the notion of *irredundant region set* is exploited, to minimize the number of places in the net without affecting its behavior. The synthesis method can derive different classes of place-irredundant Petri Nets (e.g., pure, free choice, unique choice) from the same TS, depending on the constraints imposed on the synthesis algorithm. This method has been implemented and applied in different frameworks. The results obtained from the experiments have demonstrated the wide applicability of the method.

Index Terms—Petri Nets, transition systems, concurrent systems, asynchronous systems, synthesis.

1 Introduction

Petri nets [48], [41] are a widespread formalism to model concurrent systems. By labeling transitions with symbols from a given alphabet, transitions can be interpreted as the occurrence of events or the execution of tasks in a system.

Labeled Petri Nets have been used in numerous applications: design and specifications of asynchronous circuits [53], [12], [34], [32], resource allocation in operating systems and distributed computation [55], analysis of concurrent programs [50], performance analysis and timing verification [29], [52], and high-level design [23]. Petri Nets are popular due to their inherent ability to express concurrency, choice and causality between events in a system, without explicit enumeration of global states.

Although checking properties of Petri Nets could be difficult in general, for some subclasses of Petri Nets there are efficient verification algorithms. In this paper, we will deal with safe Petri Nets (a place cannot contain more than one token). Safe nets have high expressive power, in particular every finite state system can be expressed as a safe labeled PN. On the other hand, safe nets are also well suited for verification.

- *J. Cortadella is with the Department of Software, Technical University of Catalonia, Jordi Girona Salgado s/n, Campus Nord. Modul C6, 08034-Barcelona, Spain. E-mail: jordic@lsi.upc.es.*
- *M. Kishinevsky is with Intel Corporation, JFT-104, 2111 NE 25th Ave., Hillsboro, OR 97124-5961. E-mail: m.kishine@ichips.intel.com.*
- *L. Lavagno is with the Dipartimento di Elettronica, Politecnical di Torino, C. Duca degli Abruzzi 24, 10129 Torino, Italy. E-mail: lavagno@polito.it.*
- *A. Yakovlev is with the Department of Computing Science, University of Newcastle upon Tyne, Claremont Tower, Claremont Road, Newcastle upon Tyne NE1 7RU, U.K. E-mail: Alex.Yakovlev@newcastle.ac.uk.*

Manuscript received 12 June 1996; revised 5 Aug. 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102029.

1.1 State-Based vs. Event-Based Models

State-based models, like FSMs [22], [33] and burst-mode automata [43], are often used for formal specification and verification of complex systems. Furthermore, the formal semantics of event-based models, like CSP [28] and CCS [37], [38], are defined in terms of states. The drawback of state-based models is that they represent causality, concurrency, and conflict relations between events in terms of state sequences or state configurations (e.g., state diamonds). For more succinct representation, it is very important to identify the set of *causality relations*, *concurrent events*, and *conflict conditions* implicit in the state-based representation because they carry useful information for the designer or / and design algorithms.

In this paper, we present a method which, given a finite state model, called Transition System (TS) in the sequel, synthesizes a safe, place-irredundant Petri Net with a reachability graph that is *bisimilar* to the original TS. By moving from a TS representation of the system to a PN model which exhibits the same behavior we simplify the representation of concurrency and causality of the system.

The synthesis method provides a technique for transforming specifications. Given a model which can be mapped into a TS, we can derive a PN which is bisimilar to the initial model of the process. In such a way we can create a tool which automatically translates CSP, CCS, FSM, Burst-mode machines, and other models into labeled Petri Nets. We can also use this tool to transform Petri Nets themselves, aiming at optimality under some criterion (place count, transition count, number of places, PN graph complexity, etc.) or for deriving a net belonging to a given class (pure, free choice, unique choice, etc.). This opens up an avenue for building interactive tools where a designer has

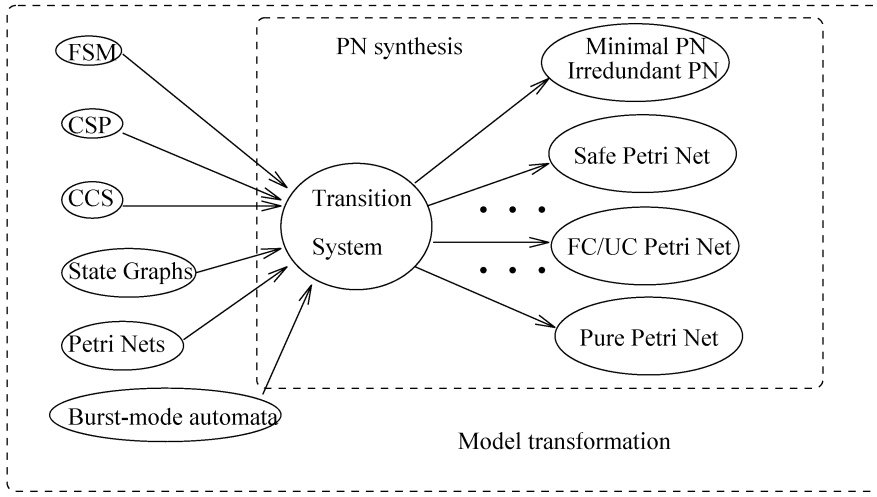


Fig. 1. A framework for synthesizing PNs and transforming specifications.

the possibility to play with a PN-like specification, performing equivalent transformations of PNs, and/or transformations of other specifications into PNs under different design constraints and optimization criteria. Fig. 1 shows our framework for synthesizing PNs and transforming specifications.

1.2 Applications of the Method

There are well-known algorithms (see, e.g., [18], [32], [1]) to extract a finite-state representation of the sequential behavior of a synchronous or asynchronous circuit. But, a really user-friendly interaction can be achieved only by presenting the designer a *timing diagram*-like PN that represents the same behavior, with explicit causality. Section 5.1 contains an example of how different synthesis options (like minimizing transitions, choosing one specific subclass of PN, and so on) result in different degrees of readability of the synthesized PN.

In the same vein, FSM-based formal verification systems ([13], [22], [33]) provide information about property check failures in the form of one specific execution path in a finite state representation. This may not always be meaningful, as it may contain redundant information. Explicitly extracting a *set* of such paths (in the form of a finite state system) and presenting a Petri Net-like representation for it would greatly help the designer in the difficult task of finding out where the problem really lies.

Apart from interaction with the designer, which is a major motivation for this work, we also address the issue of extracting explicit causal relations in order to be able to apply analysis and synthesis techniques which rely on explicit causality and concurrency information ([45], [29]). In [14], [15], we showed that *regions*, sets of states corresponding to places in PNs, are tightly connected with properties that must be preserved across the state encoding process for asynchronous circuits. Hence, regions and their intersections can be efficiently used for state signal insertion and, therefore, for synthesis of digital circuits in general.

Moreover, classical techniques for *Petri Net composition* ([51]) are based on the creation of a "cross-product" between transitions with the same label. Suppose that two PNs N_1 and N_2, with n and m transitions labeled with the same name, respectively, are composed. The resulting net has $m \cdot n$ transitions with the same label, even though some of these are not really needed (i.e., there exists a smaller PN representing the same composed behavior). Section 5.1 describes this application more in detail, with examples.

Reference [50] shows that PNs can be useful for analysis of concurrent programs. There, a PN is derived from a task flow-graph for an Ada program. No efficient technique for deriving a PN is used. We think that our technique for synthesis of PNs can provide an efficient machinery for the analysis of concurrent programs.

1.3 Related Work

The concept of regions was introduced in [24] (and developed in [42], [3], [6], [21], [40]) as a fundamental link between state-based and event-based specifications. A comprehensive review of the theory of regions can be found in [4]. "State" in safe Petri Nets is distributed among places: Each state is a set of marked places, and each place is marked in a set of states that form the corresponding region. A region in a Transition System is exactly a set of states, such that transitions coming in and going out of it "mimic" the PN firing behavior.

These papers provide the formal framework for our contribution, but suffer from a series of problems:

- Their contribution was mainly category-theoretical, aimed at obtaining a *canonical* representation of the PN, with many places (actually, as many places as could be added without changing the behavior of the net). On the other hand, we strive to *minimize* the number of places in order to make the final Petri Net easier for the designer to understand. This problem was also tackled in [21], aiming at deriving a place-irredundant Petri Net, but not at minimizing the number of places. Two open problems are formulated in [21]:
  1) if the existence of an optimal net can be characterized in terms of the TSs and
  2) if there always exists at most one optimal net which could be considered canonical.

Our paper gives a positive answer to the first problem and a negative answer to the second problem.

- They did not address the problem of *merging* and *splitting* "equivalent" labels, which model the same event, but must be split in order to yield a valid or/and efficient Petri Net. We, for the first time, provide a method for label splitting which seems to satisfactorily solve the problem.[1]
- They were limited to *elementary* TSs, which are quite restricted, while we can handle the full class of TSs by means of transition splitting.
- They produce a PN with an RG *isomorphic* to the TS, which appears to be too strong a degree of correspondence. Our method extends this result to *excitation-closed* TSs (ECTS), which include not only ETSs, but also all those TSs that have *bisimilar* ETS.

1.4 Contributions

In this paper, we present an algorithm for generating a complete set of *minimal* regions (which are analogous to prime implicants in Boolean minimization) and, further, for removing redundant regions (which is similar to generating a prime irredundant cover in Boolean minimization). We can either generate all irredundant nets and take the minimum one among them (an exact minimization of places in PNs) or we can heuristically select a minimal prime and place-irredundant net if searching for the minimum is too time consuming.

We use different cost functions for minimization of PNs, depending on whether the designer requires minimizing the number of places, transitions, or arcs in the PN graph. We can also restrict the resulting PN to belong to a specific subclass (e.g., free-choice) if required by the designer, as shown in Section 5.

Our technique of global optimization is complementary to the local optimization and reduction technique that is based on applying a set of local rules for replacing a more complex fragment of a PN with a simpler one, while preserving certain semantic, behavioral, or structural properties [7], [54], [11], [41], [20]. In fact, as discussed in Section 4.5, we apply those structural reduction techniques in order to reduce the complexity of the TS representation when we derive the TS from a PN with the objective of optimizing it.

It is known [3] that, for ETS, the complexity of synthesis of PNs is polynomial in the size of the TS. However, direct application of methods from [3] would require explicit enumeration of states of the TS. To avoid explicit enumeration, we have defined synthesis conditions (called excitation closure conditions) in a form that allows us to use symbolic techniques based on Reduced Ordered Binary Decision Diagrams for representing sets of states and checking synthesis conditions. Thus, many real-life examples with large TSs become manageable. For nonexcitation closed TSs, an additional amount of computation is required for splitting labels and calculating minimal regions for the TS after splitting. In the worst case of complete splitting, there is one label per arc and, hence, the number of minimal regions is equal to the number of states.

The paper is organized as follows. Section 2 formally introduces Transition Systems, Petri Nets, and regions. Section 3 introduces the new definitions and theoretical results required by our synthesis algorithm. Section 4 describes the synthesis algorithms in detail and formally proves their correctness. It also provides a number of modifications to the basic method. Section 5 presents applications and some experimental results. Section 6 concludes the paper. Due to lack of space, all statements in Sections 3 and 4 (except for Theorem 3.3 which is fully proven) are only given with only informal proofs. The details can be found in the technical report [17], which is the extended version of this paper.

2 Models

2.1 Transition Systems

A *transition system* (TS) is a quadruple [42] $TS = (S, E, T, s_{in})$, where S is a *nonempty* set of *states*, E is a set of *events*, $T \subseteq S \times E \times S$ is a *transition* relation, and s_{in} is an *initial state*. The elements of T are called the *transitions* of TS and will be often denoted by $s \xrightarrow{e} s'$ instead of (s, e, s').

A transition system is *finite* if S and E are finite. In the sequel, only finite transition systems will be considered. A TS is called *deterministic* if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. Otherwise, a TS is called *nondeterministic*.

The transitive closure of the transition relation T is called the *reachability relation* between states and is denoted by T^*. In other words, state s' is reachable from state s if there is a (possibly empty) sequence of transitions from T: $\sigma = (s, e_1, s_1), ..., (s_k, e_k, s')$. This is denoted by $s \xrightarrow{\sigma} s'$ or simply by $s \xrightarrow{*} s'$ if the sequence is not important. We also write $s \xrightarrow{e}$, $s \xrightarrow{\sigma}$, $\xrightarrow{e} s$, and $\xrightarrow{\sigma} s$ if there is a state $s' \in S$ such that $s \xrightarrow{e} s'$, $s \xrightarrow{\sigma} s'$, $s' \xrightarrow{e} s$, or $s' \xrightarrow{\sigma} s$, correspondingly. Each state is reachable from itself, since we allow empty sequences in the definition.

Two states, s and s', are *confluent* if there is a state s'' which is reachable both from s and s' [30]. Note that, according to the definition of reachability, s'' can coincide with s or s'.

Every transition system $TS = (S, E, T, s_{in})$ is assumed to satisfy the following axioms:

A.1) No self-loops: $\forall (s, e, s') \in T : s \neq s'$;
A.2) Every event has an occurrence: $\forall e \in E : \exists (s, e, s') \in T$;
A.3) Every state is reachable from the initial state:

$$\forall s \in S : s_{in} \xrightarrow{*} s.$$

An additional axiom is assumed in [42] (no multiple arcs between pairs of states):

$$\forall (s, e_1, s_1), (s, e_2, s_2) \in T : [s_1 = s_2 \Rightarrow e_1 = e_2].$$

This axiom is required in [42] so that behavior-preserving morphisms between TSs can be partially defined. A partial

1. See Section 5.1 for an example of how label splitting can be used also to increase the readability of the synthesized PN, by forcing it to belong to the subclass known as *free-choice*.

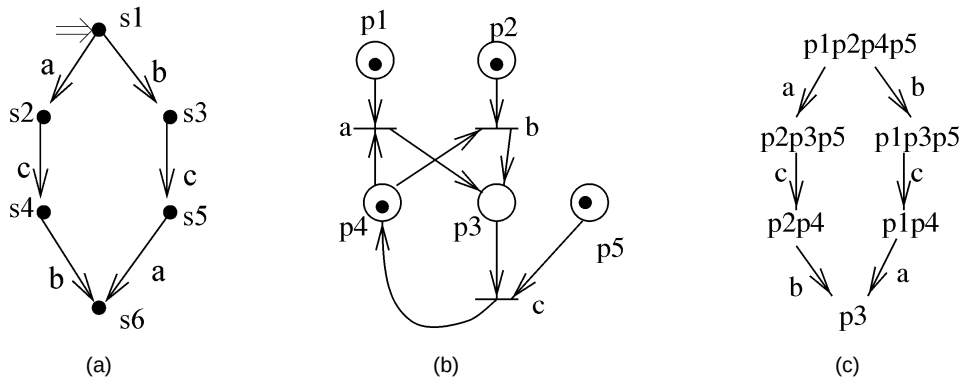


Fig. 2. An example of Transition System (a), a corresponding PN (b), and its RG (c).

mapping implies that some of the "twin" events which do not satisfy the above axiom become nonobservational. This can lead to discrepancies between the behaviors of the original and mapped TSs. In our framework of applications, we are only interested in transformations and equivalences of TSs in which events are *totally* mapped.

Therefore, the above axiom can be safely omitted (see Definitions 2.4, 2.5, and 2.6 for further details).

A TS can be represented by an arc-labeled directed graph. A simple example of a TS without cycles is shown in Fig. 2a.

2.2 Petri Nets

A Petri Net [48], [41] is a quadruple $N = (P, T, F, m_0)$, where P is a *finite* set of places, T is a *finite* set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and m_0 is the initial marking. A transition $t \in T$ is enabled at marking m_1 if all its input places are marked. An enabled transition t may fire, producing a new marking m_2 with one less token in each input place and one more token in each output place ($m_1 \stackrel{t}{\rightarrow} m_2$).

A PN expressing the same behavior as the TS from Fig. 2a is shown in Fig. 2b.

The sets of input and output places of transition t are denoted by $\bullet t$ and $t\bullet$. The sets of input and output transitions of place p are denoted by $\bullet p$ and $p\bullet$. The set of all markings of N reachable from the initial marking m_0 is called its Reachability Set.

A *labeled PN* is a PN with a labeling function $\lambda: T \rightarrow A$ which puts into correspondence every transition of the net with a symbol (called label) from the alphabet A. If no two transitions have the same label (unique labeling), then each transition in the net can be uniquely identified by its label. In such a case, we can use the label as the name of the transition. The *Reachability Graph* (RG) of a PN is a transition system in which the set of states is the Reachability Set, the events are the transitions of the net and a transition (m_1, t, m_2) exists if and only if $m_1 \stackrel{t}{\rightarrow} m_2$. (A formal definition of RG is given in Section 2.5).

One can easily check that the RG in Fig. 2c, derived from the PN in Fig. 2b, is isomorphic to the TS (Fig. 2a).

A net is called *safe* if no more than one token can appear in a place. Safe nets are especially widely used in many applications, since they have simple verification algorithms [25] and simple semantics. A net is called a *pure* net if $(p, t) \in F$ implies that $(t, p) \notin F$, i.e., for each transition t the following condition is satisfied: $t\bullet \cap \bullet t = \emptyset$. A net is called *simple* if no two transitions t_1 and t_2 have the same sets of input and output places (i.e., $\forall t_1$, either $t_2 \bullet t_1 \neq \bullet t_2$ or $t_1 \bullet \neq t_2 \bullet$).

2.3 Regions

Let S' be a subset of the states of a TS, $S' \subseteq S$. If $s \notin S'$ and $s' \in S'$, then we say that transition $s \stackrel{a}{\rightarrow} s'$ *enters* S'. If $s \in S'$ and $s' \notin S'$, then transition $s \stackrel{a}{\rightarrow} s'$ *exits* S'. Otherwise, transition $s \stackrel{a}{\rightarrow} s'$ *does not cross* S'. In particular, if $s \in S'$ and $s' \in S'$, then the transition is said to be *internal* to S', and if $s \notin S'$ and $s' \notin S'$, then the transition is *external* to S'.

DEFINITION 2.1. *Let $TS = (S, E, T, s_{in})$ be a TS. Let $S' \subseteq S$ be a subset of states and $e \in E$ be an event. The following conditions (in the form of predicates) are defined for S' and e:*

$$\mathbf{in}(e, S') \equiv \exists (s, e, s') \in T : s, s' \in S'$$
$$\mathbf{out}(e, S') \equiv \exists (s, e, s') \in T : s, s' \notin S'$$
$$\mathbf{enter}(e, S') \equiv \exists (s, e, s') \in T : s \notin S' \wedge s' \in S'$$
$$\mathbf{exit}(e, S') \equiv \exists (s, e, s') \in T : s \in S' \wedge s' \notin S'.$$

The notion of a *region* is central for the synthesis of PNs. Intuitively, each region corresponds to a place in the synthesized PN, so that there is a one-to-one correspondence between states of the region and markings of the PN in which this place has a token.

DEFINITION 2.2 (region). *A set of states $r \subseteq S$ in $TS = (S, E, T, s_{in})$ is called a* region *if the following two conditions are satisfied for each event $e \in E$:*

1) $\mathbf{enter}(e, r) \Rightarrow \neg\mathbf{in}(e, r) \wedge \neg\mathbf{out}(e, r) \wedge \neg\mathbf{exit}(e, r)$
2) $\mathbf{exit}(e, r) \Rightarrow \neg\mathbf{in}(e, r) \wedge \neg\mathbf{out}(e, r) \wedge \neg\mathbf{enter}(e, r)$

A region is a subset of states with which *all* transitions labeled with the same event e have exactly the same "entry/exit" relation. This relation will become the predecessor/successor relation in the Petri Net. The event may either always be an *enter* event for the region (Case 1 in the previous definition), or always be an *exit* event (Case 2), or never "cross" the region's boundaries (each transition labeled with e is *internal* or *external* to the region if the antece-

dents of neither Case 1 nor Case 2 hold). The transition corresponding to the event will be successor, predecessor or unrelated with the corresponding place, respectively.

Let us consider the TS shown in Fig. 2. The set of states $r_3 = \{s_2, s_3, s_6\}$ is a region, since all transitions labeled with a and with b enter r_3, and all transitions labeled with c exit r_3. On the other hand, $\{s_2, s_3\}$ is not a region since transition $s_1 \xrightarrow{b} s_3$ enters this set, while another transition also labeled with b, $s_4 \xrightarrow{b} s_6$, does not. Similar violations of the region conditions exist for two transitions labeled with a. However, there are no violations for c since both transitions labeled with c exit this set of states.

Each TS has two *trivial regions*: the set of all states, S, and the empty set. Further on we will always consider only nontrivial regions. The set of nontrivial regions of TS will be denoted by R_{TS}. For each state $s \in S$, we define the set of nontrivial regions containing s, denoted by R_s.

A region r is a *preregion* of event e if there is a transition labeled with e which exits r. A region r is a *postregion* of event e if there is a transition labeled with e which enters r. The set of all preregions and postregions of e is denoted with $^\circ e$ and e°, respectively. By definition, it follows that if $r \in {}^\circ e$, then all transitions labeled with e exit r. Similarly, if $r \in e^\circ$, then all transitions labeled with e enter r. Let r and r' be regions of a TS. A region r' is said to be a *subregion* of r iff $r' \subset r$. A region r is a *minimal* region if there is no other region r' which is a subregion of r.

There are eight nontrivial regions in the TS from Fig. 2: $r_1 = \{s_1, s_3, s_5\}$; $r_2 = \{s_1, s_2, s_4\}$; $r_3 = \{s_2, s_3, s_6\}$; $r_4 = \{s_1, s_4, s_5\}$; $r_5 = \{s_1, s_2, s_3\}$; $r_6 = \{s_4, s_5, s_6\}$; $r_7 = \{s_2, s_4, s_6\}$; $r_8 = \{s_3, s_5, s_6\}$. All of these regions are minimal. Preregions and postregions are defined as follows: $^\circ a = \{r_1, r_4\}$; $^\circ b = \{r_2, r_4\}$; $^\circ c = \{r_3, r_5\}$; $a^\circ = \{r_3, r_7\}$; $b^\circ = \{r_3, r_8\}$; $c^\circ = \{r_4, r_6\}$.

2.4 Properties of Regions

The following propositions state a few important properties of regions [6], [42], [17].

PROPERTY 2.1. *If r and r' are two different regions such that r' is a subregion of r, then $r - r'$ is a region.*

PROPERTY 2.2. *A set of states, r, is a region, if and only if its coset $\bar{r} = S - r$ is a region, where S is a set of all states of the* TS.

PROPERTY 2.3 [6], [17]. *Every region can be represented as a union of disjoint minimal regions.*

Property 2.1 has been mentioned in [6] for the subclass of elementary TSs. We generalize it for the complete class of TSs. Property 2.2 was given in [42]. Property 2.3 is a stronger refinement of the corresponding property from [6], which shows that any region can be viewed as a linear combination of minimal regions.

2.5 Elementary Transition Systems

Some of the results given in this section were formerly presented for *elementary nets* [42]. Their extension to Petri Nets is straightforward and discussed in [17]. In the sequel, we will reformulate for Petri Nets the major previous results on elementary nets.

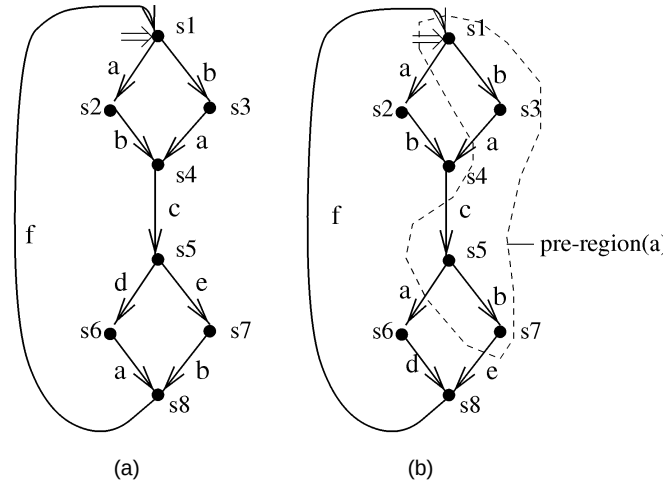


Fig. 3. Examples of elementary (a) and nonelementary (b) TSs.

2.5.1 Axioms for Elementary Transition Systems

A transition system $TS = (S, E, T, s_{in})$ is called *elementary* [42] (ETS) if it satisfies, in addition to (A1)-(A3), the following two axioms about regions:

A.4) State separation property: $\forall s, s' \in S : [R_s = R_{s'} \Rightarrow s = s']$;
A.5) Forward closure property:

$$\forall s \in S \forall e \in E : \left[{}^\circ e \subseteq R_s \Rightarrow s \xrightarrow{e} \right]$$

(A4) implies that two different states must belong to different sets of regions. (A5) implies that if state s is included in all preregions of event e, then e must be enabled in s. It is easy to see that the TS shown in Fig. 2 is elementary, since all axioms (A1)-(A5) are satisfied. For example, state s_1 is separated from any other state (axiom (A4)). This state is included into regions r_1, r_2, r_4, r_5 and there is no other state which is covered by the same set of regions (see Section 2.3 for the list of all regions). To illustrate axiom (A5), let us consider event a. For two states, s_1 and s_5, condition $^\circ a \subseteq R_{s_1}$ and $^\circ a \subseteq R_{s_5}$ holds. Both states can have an exit arc labeled by event a. Hence, Axiom (A5) is satisfied.

The TS shown in Fig. 3a is a cyclic elementary TS, while Fig. 3b shows a nonelementary TS. The forward closure property is violated for events a and b. Let us consider event a. The only minimal preregion of a is region $\{s_1, s_3, s_5, s_7\}$. Therefore $^\circ a \subseteq R_{s_7}$, but there is no transition labeled with a from s_7.

DEFINITION 2.3 (Reachability Graph). *Let $N = (P, E, F, m_0)$ be a* PN. *The* reachability graph *of N is the* TS $RG(N) = (S_N, E_N, T_N, m_0)$, *where $S_N \subseteq 2^P$, $T_N \subseteq 2^P \times E \times 2^P$ such that:*

1) *S_N is the Reachability Set of N, with each state represented as the set of places marked in the corresponding marking,*[2]

2. Since we are only considering safe and pure Petri Nets, places can have at most one token.

2) $T_N = \{(m, e, m') \mid m, m' \in S_N \wedge m \xrightarrow{e} m'\}$,
3) $E_N = \{e \mid e \in E \wedge \exists (m, e, m') \in T_N\}$.

It was shown in [42] that the RG of a PN is always an elementary TS and vice versa, i.e, if a TS is elementary, then a PN with a reachability graph isomorphic to the TS can be constructed. The procedure given by [42] to synthesize a PN, $N_{TS} = (R_{TS}, E, F_{TS}, R_{s_{in}})$, from an ETS, $TS = (S, E, T, s_{in})$, is as follows:

Algorithm: saturated PN synthesis

- For each event $e \in E$, generate a transition labeled with e in the PN;
- For each region $r_i \in R_{TS}$, generate a place r_i;
- Place r_i contains a token in the initial marking iff the corresponding region r_i contains the initial state of the ETS s_{in};
- The flow relation is as follows: $e \in r_i\bullet$ iff r_i is a preregion of e and $e \in \bullet r_i$ iff r_i is a postregion of e, i.e.,

$$F_{TS} \stackrel{\text{def}}{=} \left\{(r, e) \middle| r \in R_{TS} \wedge e \in E \wedge r \in {}^\circ e\right\} \cup \left\{(e, r) \middle| r \in R_{TS} \wedge e \in E \wedge r \in e^\circ\right\}.$$

THEOREM 2.1. *Let $TS = (S, E, T, s_{in})$ be an* ETS. *The reachability graph of $N_{TS} = (R_{TS}, E, F_{TS}, R_{s_{in}})$ obtained by the algorithm of saturated* PN *synthesis is isomorphic to TS.*

Intuitively, the net has as many places as nontrivial regions. Each preregion (postregion) of an event is a predecessor (successor) place of the corresponding transition. All places that correspond to regions that cover the initial state must be marked in the initial marking.

A PN which is synthesized following this procedure is called a *saturated* net, since all regions are mapped into the corresponding places. Any ETS has a unique saturated net; however, it has a lot of redundancy. As shown in [6], it is enough to consider only minimal regions. The net constructed from all minimal regions is also unique and is called a *minimal saturated net*. We will go two steps further and will first synthesize a *place-irredundant* PN and, then, a *place-minimal* PN, still preserving bisimilarity between its RG and the ETS.

Another important drawback of the procedure described in [42] is that axioms (A4) and (A5) do not provide an efficient algorithm for checking elementarity, since they require the derivation of *all regions* of a TS and checking elementarity conditions *for each individual state*. Our procedure is specifically aimed at deriving *minimal regions* by using simplified elementarity checks, that admit an efficient implementation.

Finally, the synthesis method presented in [42] produces a PN with an RG isomorphic to the TS. Our method extends this result to *excitation-closed* TSs (ECTS), which cover not only ETSs, but also all those TSs that have some bisimilar ETS.

2.6 Split-Morphism, Isomorphism, and Bisimilarity

The following notions will be used for comparing behavior of TSs and PNs.

DEFINITION 2.4 (Split-morphism). *Let $TS_1 = (S_1, E_1, T_1, s_{in_1})$ and $TS_2 = (S_2, E_2, T_2, s_{in_2})$ be two* TSs. *A* split-morphism *h from TS_1 to TS_2 is a pair (h_S, h_E) of total mappings, h_S being bijective and h_E being surjective,*

$$h_S : S_1 \to S_2$$
$$h_E : E_1 \to E_2$$

which satisfies

$$(s, e, s') \in T_1 \Leftrightarrow (h_S(s), h_E(e), h_S(s')) \in T_2.$$

DEFINITION 2.5 (Isomorphism). *A split-morphism $h = (h_S, h_E)$ from TS_1 to TS_2 is an* isomorphism *if h_E is bijective.*

Two TSs are said to be *isomorphic* (*split-morphic*) if there exists an isomorphism (split-morphism) between them (from one to the other).

The concept of split-morphism will be used when an event is represented by different instances in a TS. Later on, when deriving Petri Nets, this splitting will result in different transitions with the same label.

DEFINITION 2.6 (Bisimulation [2]). *Let $TS_1 = (S_1, E, T_1, s_{in_1})$ and $TS_2 = (S_2, E, T_2, s_{in_2})$ be two* TSs *with the same set of events. A* bisimulation *between TS_1 and TS_2 is a binary relation R between S_1 and S_2 such that*

1.a) *for every $s_1 \in S_1$, there exists $s_2 \in S_2$ such that $s_1 R s_2$.*
1.a) *for every $s_2 \in S_2$, there exists $s_1 \in S_1$ such that $s_1 R s_2$.*
2.a) *for every $(s_1, e, s_1') \in T_1$ and for every $s_2 \in S_2$ such that $s_1 R s_2$, there exists $(s_2, e, s_2') \in T_2$ such that $s_1' R s_2'$.*
2.b) *for every $(s_2, e, s_2') \in T_2$ and for every $s_1 \in S_1$ such that $s_1 R s_2$, there exists $(s_1, e, s_1') \in T_1$ such that $s_1' R s_2'$.*

Intuitively, Conditions 1a) and 2a) define a simulation of TS_1 by TS_2. Two TSs are said to be *bisimilar* if they can simulate each other, i.e., there exists a bisimulation between them.

DEFINITION 2.7 (Auto-bisimulation and bisimilar states [2]). *Let $TS = (S, E, T, s_{in})$ be a* TS. *An* auto-bisimulation *of TS is a bisimulation between TS and itself. Two states $s_1, s_2 \in S$ are* bisimilar *if $s_1 R s_2$ for some auto-bisimulation of TS.*

The relation "*is bisimilar to*" is an equivalence relation and partitions all TSs into equivalence classes. A TS is said to be *minimal* if no other element in its equivalence class has a set of states with smaller cardinality, in other words:

DEFINITION 2.8 (Minimal TS). *A* TS *is said to be minimal if it contains no different states, s_1 and s_2, which are bisimilar.*

Fig. 4 illustrates the notions of isomorphism, split-morphism, and bisimulation between TSs. The split-morphism of Fig. 4b maps two different instances of an event (a_1 and a_2) onto the same event (a).

The work presented in this paper is based on transformations between split-morphic or bisimilar TSs. These notions of equivalence provide stronger conditions than language equivalence in general ([37]). This implies, for example, that deadlock and liveness properties will be preserved for a PN generated from the TS.

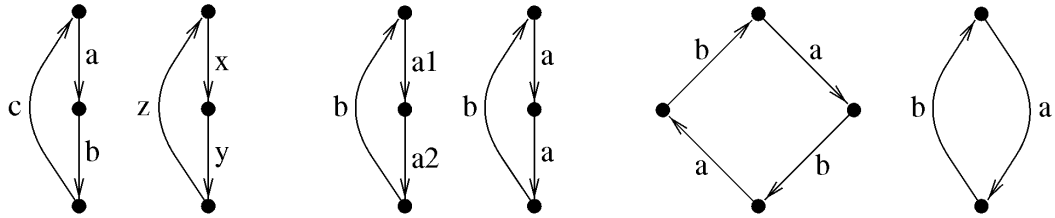


Fig. 4. (a) Isomorphism, (b) Split-morphism, (c) Bisimulation.

3 Excitation Closed Transition Systems

In this section, we define the concept of ECTS based on the notion of excitation regions, which are the sets of states corresponding to the transitions of a PN.

3.1 Excitation Regions and Switching Regions

While regions in a TS are related to places in the corresponding PN, an excitation region [32] (ER) for event a is a maximal set of states in which transition a is enabled. Therefore, excitation regions are related to transitions of the PN. Similarly to ERs, we define switching regions as sets of states reached *immediately after* the occurrence of an event.

DEFINITION 3.1 (Excitation and switching regions). *A set of states S' is called an* excitation region *for event a, denoted by $ER(a)$, if it is a* maximal *set of states such that for every state $s \in S'$ there is a transition $s \xrightarrow{a}$.*

A set of states S' is called a switching region *for event a, $SR(a)$, if it is a* maximal *set of states such that for every state $s \in S'$ there is a transition $\xrightarrow{a} s$.*

In the TS from Fig. 2a, $ER(a) = \{s_1, s_5\}$ and $SR(a) = \{s_2, s_6\}$.

3.2 Excitation Closure

In this section, we introduce a new class of TSs, excitation-closed transition systems, and show their relationship with ETSs. We prove two important theorems which create a formal foundation for our synthesis framework. These theorems taken together state that for any excitation-closed TS there is an ETS that is bisimilar to the original TS.[3] The practical importance of this result will be illustrated further, by showing an efficient way of checking the excitation closure condition.

DEFINITION 3.2 (Excitation-closed TS (ECTS)). *A transition system $TS = (S, E, T, s_{in})$ is called* excitation-closed *if it satisfies the following two axioms:*

A.4) *Excitation closure: For each event a:* $\bigcap_{r \in {}^\circ a} r = ER(a)$

A.5) *Event effectiveness: For each event a:* ${}^\circ a \neq \emptyset$.

We now establish a correspondence between ETSs and ECTSs. For this we introduce the notion of region-based state equivalence, and define a region-separated TS as a TS minimized with respect to that equivalence.

DEFINITION 3.3 (Region-based state equivalence). *Let $TS = (S, E, T, s_{in})$ be an* ECTS *and let $\mathcal{R}$ be the binary relation on S defined as follows:*

$$s_1 \mathcal{R} s_2 \overset{\text{def}}{\Leftrightarrow} R_{s_1} = R_{s_2}.$$

Clearly, $\mathcal{R}$ is an equivalence relation. For each $s \in S$, we denote by $[s]$ the equivalence class of s. For each $S' \subseteq S$, we denote by $[S']$ the set of equivalence classes of the states in S'.

Given an ECTS and the set of all equivalence classes, $[S] = \{\pi_1, \ldots, \pi_k\}$, with respect to $\mathcal{R}$, it is possible to construct a minimized version of the ECTS such that, for each equivalence class of states $\pi_j \in [S]$, there will be exactly one corresponding state in the new TS. We will denote the new state corresponding to π_j as $s[\pi_j]$. Slightly abusing the notation, the set of states corresponding to all equivalence classes of $S' \subseteq S$ is denoted as $[S']$ and the whole set of the new states as $[S]$.

DEFINITION 3.4 (Region-separated TS). *Let $TS = (S, E, T, s_{in})$ be an* ECTS. *A region-separated* TS *is defined as follows:*

$TS_{\mathcal{R}} = ([S], E, T_{\mathcal{R}}, s[\pi_{in}])$, *where $[S]$ is defined above,*

$(s[\pi_1], e, s[\pi_2]) \in T_{\mathcal{R}} \Leftrightarrow \exists s_1 \in \pi_1, s_2 \in \pi_2 : (s_1, e, s_2) \in T$ *and* $s_{in} \in \pi_{in}$.

It is easy to show that $T_{\mathcal{R}}$ is well-defined, i.e., if $(s[\pi_1], e, s[\pi_2])$, then $\forall s_1 \in \pi_1 \exists s_2 \in \pi_2 : (s_1, e, s_2) \in T$ and vice versa.

Fig. 5a depicts a nonelementary TS in which two pairs of states, $\{s_0, s_4\}$ and $\{s_1, s_5\}$ are region-based equivalent. The set of all equivalence classes of states is $[S] = \{\pi_1, \pi_2, \pi_3, \pi_4\} = \{\{s_0, s_4\}, \{s_1, s_5\}, \{s_2\}, \{s_3\}\}$ and $\pi_{in} = \pi_1$. The corresponding region-separated TS is shown in Fig. 5b, where the equivalent states have been merged. This minimized TS is now an ETS with a behavior bisimilar to the reachability graph of the Petri Net shown in Fig. 5c.

The next theorem shows that the region-separated TS is a (partly) minimized version of a ECTS with respect to bisimulation.

THEOREM 3.1. *Let $TS = (S, E, T, s_{in})$ be an* ECTS. *Then,* TS *and the corresponding region-separated* TS, *$TS_{\mathcal{R}}$, are bisimilar.*

The proof outline is given in the Appendix.

The following theorem establishes a connection between ETS and ECTS in both directions.

THEOREM 3.2.

1) *If a TS is elementary, then it is excitation-closed.*
2) *Let $TS = (S, E, T, s_{in})$ be an* ECTS. *Then, $TS_{\mathcal{R}}$ is elementary.*

3. Due to lack of space, the proofs have been omitted. They can be found in [17].

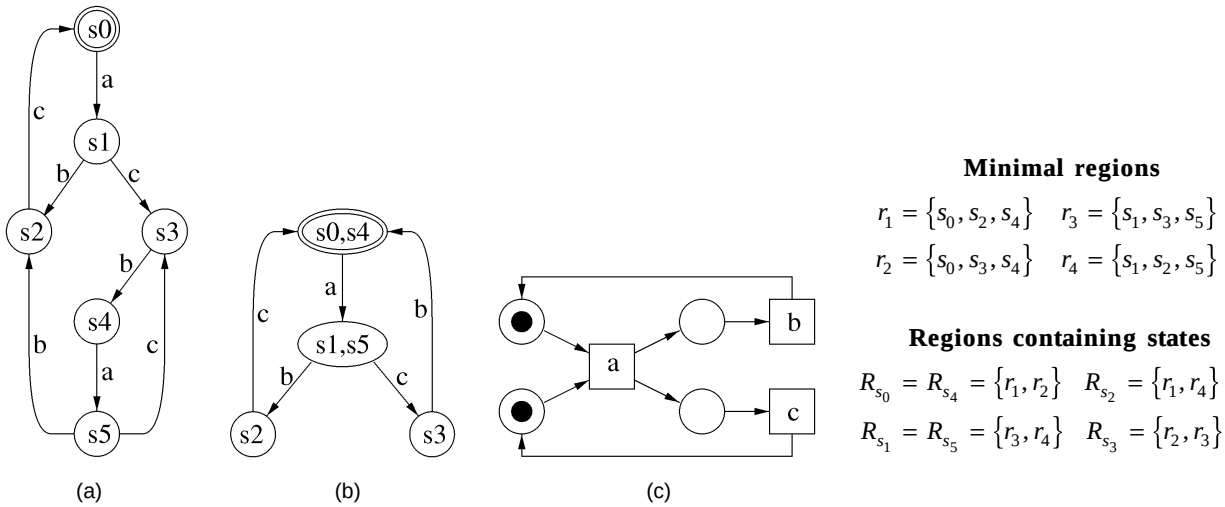


Fig. 5. (a) Nonelementary ECTS, (b) Bisimilar ETS after merging region-based equivalent states, (c) Petri Net after synthesis of the ETS.

The proof outline is given in the Appendix. As follows from Theorem 3.2.1, each ETS is an ECTS. On the other hand, Theorems 3.1 and 3.2.2, show that, for each ECTS, an elementary bisimilar TS can be constructed by mapping all region-equivalent states into one state. This procedure corresponds to classical state minimization of finite automata. However, this relation with classical state minimization is not straightforward and will be analyzed more in detail in Section 3.5. For example, Fig. 6a shows a nondeterministic TS which is excitation closed, but not elementary, since, e.g., output states of transitions labeled with *a* are not separated by regions. The corresponding TSR is obtained by merging two pairs of region-equivalent states and is elementary (Fig. 6b). An important observation can be made. Any ETS is deterministic [42]. However, a nondeterministic TS can be an ECTS when it has a bisimilar ETS. Fig. 6 depicts an example illustrating this fact.

3.3 Minimality

This subsection focuses on minimality of TSs. Recall (Section 2.6) that a TS is minimal if all its states are unique in terms of their posthistory, i.e., no two distinct states of the TS are bisimilar to each other. Our main goal here is to show the conditions under which region-based equivalence R of states coincides with bisimulation-based equivalence R. These conditions are based on the notion of confluence. The following important properties are related to minimality.

THEOREM 3.3. *Let $TS = (S, E, T, s_{in})$ be an* ETS. *If TS is not minimal, then for any pair of bisimilar states, $s_1, s_2 \in S$, s_1 and s_2 are not confluent.*

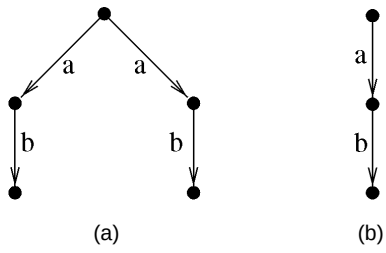


Fig. 6. (a) A nondeterministic ECTS. (b) A bisimilar ETS.

The proof of this theorem is given in the Appendix.

Let $TS = (S, E, T, s_{in})$ be an ECTS, then the following properties (proven in [17]) hold:

- If *TS* is minimal, then *TS* is elementary.
- If *TS* is minimal, then it is isomorphic to the region-separated TS, $TS_{\mathcal{R}}$.
- If every pair of bisimilar states of *S* is confluent, then $TS_{\mathcal{R}}$ is minimal.

The last property implies that by constructing a region-separated TS one implicitly minimizes the TS for all confluent states. For example, the region-separated TS in Fig. 4c on the right is a minimized version of the ECTS from Fig. 4c on the left. The relationship with state minimization for nonconfluent states will be discussed in Section 3.5. Fig. 7d shows an example of an excitation closed TS, which is not minimal, since states marked with 2 and with ∅ are bisimilar. Note that these states are not confluent. After minimizing the TS, Fig. 7e is obtained, which is not excitation closed. The excitation closure condition is violated for event *f*.

3.4 Place-Irredundant Petri Nets

The above theory provides an efficient framework to derive Petri Nets from transition systems. The main differences from the work presented in [42] are the following:

- The check for elementarity (that must be done once for each state) is reduced to a check for excitation closure (that must be done once for each event; events in a concurrent model are in general much fewer than states).
- Excitation closure guarantees the existence of a bisimilar ETS with a one-to-one correspondence between regions. The ETS will be minimal for all those pairs of bisimilar states that are confluent.

Therefore, given an ECTS, one obtains a PN whose RG is bisimilar to the ECTS by only calculating the minimal regions of the ECTS. Furthermore, the method can not only derive a Petri Net, but also minimize its reachability graph if all the bisimilar states are confluent.

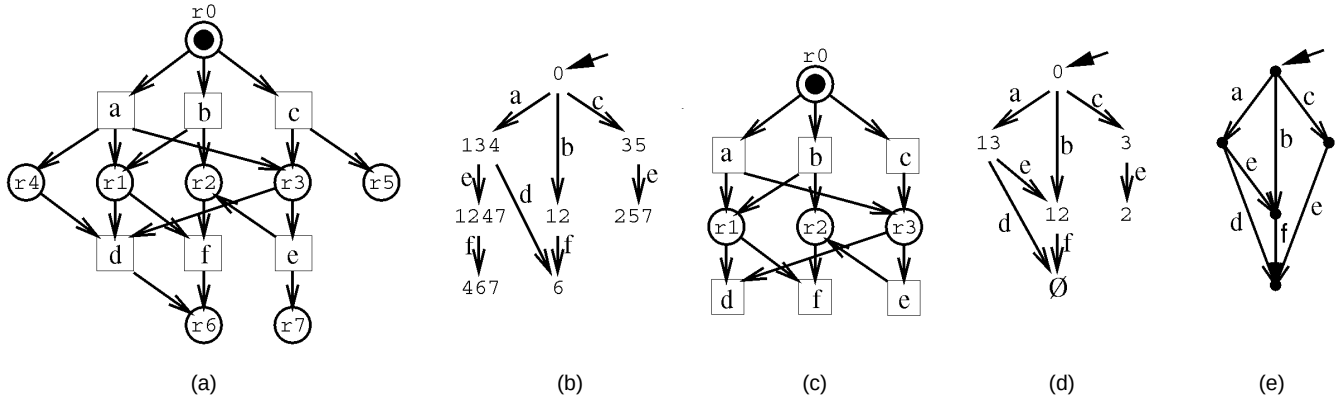


Fig. 7. (a) Minimal saturated PN, (b) RG with each state labeled with the indices of the marked places, (c) place-redundant PN, (d) bisimilar RG, (e) minimal nonelementary and not excitation closed TS.

A minimal saturated net can be redundant. Many places can still be removed from it while still preserving the bisimilarity between its RG and the ETS. By analogy with logic minimization, a *saturated net* is like the set of *all implicants* for a Boolean function, while a *minimal saturated net* is like the set of *all prime implicants*. Our goal is to provide a method for constructing an *irredundant net with minimal regions*, which is similar to an *irredundant cover of prime implicants* [9]. Unfortunately, the analogue of Quine and McCluskey's result does not hold in this case, i.e., there exist ECTSs for which *any minimum* corresponding PN requires using at least one nonminimal region. An example of such a case is given in Section 3.5.

It was proven in [6] that the RG of a minimal saturated net is isomorphic to the RG of the saturated net. Here, we provide a method to build a place-irredundant net from an ECTS and prove that the RG of the place-irredundant net is bisimilar to the ECTS. Similar related work for ETSs was presented in [21].

THEOREM 3.4. *Let* $TS = (S, E, T, s_{in})$ *be an* ETS *and let* $N_{TS} = (R_{TS}, E, F_{TS}, R_{s_{in}})$ *be the saturated net obtained by the algorithm "saturated* PN *synthesis" from Section 2.5.1. Let* $I \subseteq R_{TS}$ *be a subset of regions of TS and let* $I' = R_{TS} - I$ *be such that the excitation closure condition is satisfied without* I':

$$\forall e \in E. \bigcap_{r \in {}^\circ e - I'} r = ER(e).$$

Let $N_I = (I, E, F_I, s_I)$ *be a net where* $s_I = R_{s_{in}} - I'$ *and*

$$F_I = F_{TS} - \{(r, e), (e, r) \mid r \in I'\}.$$

Then, RG(N_I) *is bisimilar to TS.*

The proof outline of this theorem is given in the Appendix.

The converse of Theorem 3.4 is trivially true, i.e., if $I \subseteq R_{TS}$ does not satisfy the excitation closure condition for event e, then $RG(N_I)$ is not bisimilar to the TS. Discrepancy in behavior will occur in any state from $\bigcap_{r \in {}^\circ e} r - ER(e)$.

COROLLARY 3.1. *There exists a subset* $I \subseteq R_{TS}$ *such that I contains only minimal preregions and* RG(N_I) *is bisimilar to TS.*

Next, we state a relationship between irredundant sets of regions and place-irredundant nets.

DEFINITION 3.5. (Place-irredundant net). *A labeled Petri Net is place-irredundant if no place can be removed from it without losing the bisimilarity of the* RG.

DEFINITION 3.6 (Irredundant set of regions). *Let TS be a Transition System. A set of regions R is called* redundant *if there is a region* $r \in R$ *such that* $R - \{r\}$ *still satisfies the excitation closure condition. Otherwise, R is called* irredundant.

THEOREM 3.5. *Let* $N_I = (I, E, F_I, s_I)$ *be a safe pure* PN *obtained from an* ETS *TS, as described by Theorem 3.4. If I is an irredundant set of regions, then* N_I *is place-irredundant.*

The proof is trivial.

Fig. 7 shows a minimal saturated PN (Fig. 7a) and its RG (Fig. 7b). Regions $r0$, $r1$, $r2$, and $r3$ are enough to guarantee the excitation closure of the RG and a place-irredundant PN can be obtained (Fig. 7c) with a bisimilar RG (Fig. 7d). Note that, in the initial PN region, $r4$ is a preregion for event d. The RG in Fig. 7d is a partly minimized version of RG (Fig. 7b), however, it is not minimal. A minimal RG can be neither elementary nor excitation closed (Fig. 7e). By removing redundant regions one can perform an implicit partial state minimization for nonconfluent states. However, complete minimization of such states can destroy elementarity and excitation closure.

3.5 Place-Minimal Petri Nets

In this section, we define a link between place-minimal nets and minimal sets of regions and further establish a relation between irredundant sets of minimal regions and minimal sets of regions.

DEFINITION 3.7 (Place-minimal net). *A labeled Petri Net is place-minimal if any other bisimilar* PN *contains a greater or equal number of places.*

DEFINITION 3.8 (Minimal set of regions). *Let TS be a Transition System. A set of regions R is called* minimal *if it is irredundant and any other irredundant set of regions contains a greater or equal number of regions.*

The relationship between minimal nets and minimal sets of regions is similar to that between irredundant nets and irredundant sets of regions. It can be stated as follows:

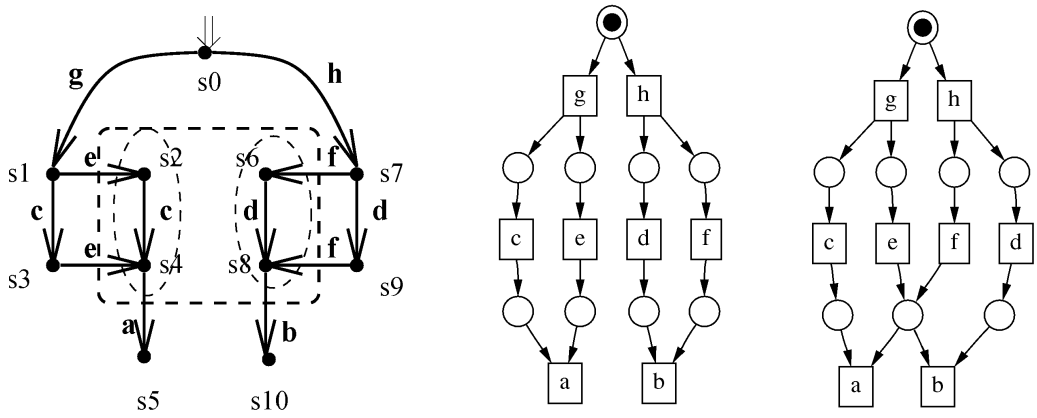


Fig. 8. ECTS, its place-irredundant and place-minimal net.

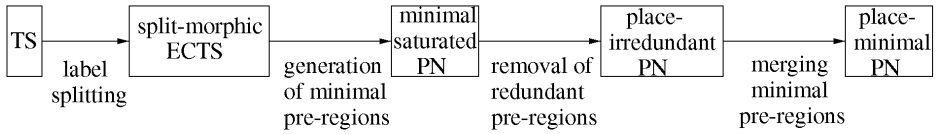


Fig. 9. Framework for the synthesis of PNs from TSs.

THEOREM 3.6. *Let $N_I = (I, E, F_I, s_I)$ be a safe pure* PN *obtained from an* ETS *TS, as described by Theorem 3.4. If I is a minimal set of regions, then N_I is place-minimal.*

The proof is trivial.

As discussed in Section 3.4, a place-irredundant PN can be always obtained using an irredundant set of *minimal* regions. In that respect, minimal regions resemble prime implicants in Boolean minimization. However, a minimal set of regions does not necessarily contain minimal regions only. Fig. 8 shows a transition system (on the left) and a place-irredundant net, corresponding to the (unique) irredundant set of minimal regions (in the middle). However, a place-minimal safe pure net (on the right) can be obtained from the place-irredundant net by merging two minimal regions, which are denoted in the TS with dotted lines, into one nonminimal region.

The relationship between irredundant sets of minimal regions and minimal sets of regions is given by the following theorem.

DEFINITION 3.9 (Min-expansion of regions). *Let R be a set of regions of a* TS. *A set of regions R′ is called a* min-expansion *of R if*

1) *for any region r, $r \in R$, the following conditions hold:*
   a) *if r is a minimal region, then $r \in R'$.*
   b) *if r is a nonminimal region and $r = r^1 \cup r^2 \ldots \cup r^k$, where $r^1, \ldots, r^k$ are disjoint minimal regions, then $r \notin R'$ and $r^1 \in R', \ldots, r^k \in R'$.*
2) *for any region r′, $r' \in R'$, there exists a region $r \in R$ such that*
   a) *if r is a minimal region, then $r' = r$*
   b) *if r is a nonminimal region and $r = r^1 \cup r^2 \ldots \cup r^k$, where $r^1, \ldots, r^k$ are disjoint minimal regions, then there exists i such that $r' = r^i$.*

THEOREM 3.7 (Minimal set of regions). *Let R be a minimal set of regions of an* ECTS. *Then, any min-expansion of R is an irredundant set of regions.*

This theorem shows that a place-minimal net can always be obtained from one of the irredundant sets of minimal regions by merging some of the disjoint minimal preregions into nonminimal preregions.

Based on the equivalences for TSs and PNs presented in this section, the framework depicted in Fig. 9 and described in the next section can be devised for the synthesis of PNs.

- Initially, the labels of a TS are split to obtain a split-morphic ECTS. Next, all minimal preregions that are predecessors of some event are generated to derive a minimal saturated PN. This restriction to event predecessors only is due to our region generation mechanism and has been shown to be sufficient in practice to obtain good results using a reasonable amount of computation time.
- Then, an irredundant subset of preregions is calculated and a place-irredundant PN obtained.
- Finally, by merging minimal preregions, further minimization of regions can be obtained. Exploring all place-irredundant nets can be computationally very expensive. Hence, we use only a greedy place merging starting from a place-irredundant net, thus yielding a quasi-place-minimal PN.

4 PETRI NET SYNTHESIS

The skeleton of the algorithm for synthesis of a PN is given by the pseudocode of function `Petri Net synthesis` shown in Fig. 10.

The input of this algorithm is a TS. The output is a PN whose RG is bisimilar to the TS. Here, we only give a rough sketch of the procedures. Further details on the methods used in the current implementation to represent and manipulate TSs efficiently are given in Section 4.5.

```
Petri net synthesis
repeat /* Generation of pre-regions
    and label splitting */
    split := false;
    for each e ∈ E do
        °e = generate_min_pre-regions (e);
        if ¬ excitation_closure (°e) then
            split_labels (e);
            split := true;
        end if
    end for
until ¬ split;
find_irredundant_cover;
map_to_PN;

generate_min_pre-regions (e)
begin
    °e = ∅;
    explored = ∅;
    expand_states (ER(e), °e,explored);
end;

expand_states (r,R,explored)
begin
    /* r is the set of states to be expanded; R collects all regions */
    /* explored is the set of expansions already generated */

    /* This check avoids to repeat previous expansions */
    if r ∈ explored then return;

    if r is a region then
        R = R - {r_i | r ⊂ r_i}; /* remove supersets of r */
                /* since they are not minimal regions */
        if ¬∃r_j ∈ R : r_j ⊆ r then R = R ∪{r};
        /* r is now minimal among the ones already generated */
        return; /* any region expanded from r is not minimal */
    end if;

    find e ∈ E violating some region condition in r;
    r' = r ∪ {set of states to legalize e}; /* (lemma 4.2) */
    expand_states (r',R,explored);
    explored = explored ∪ {r'};

    /* For some conditions the set of states must be expanded in
    two directions (see lemma 4.2) */
    if another expansion is needed then
        r' = r ∪ {another set of states to legalize e};
        expand_states (r',R,explored);
        explored = explored ∪ {r'};
    end if;
end
```

Fig. 10. Pseudocode for the algorithm to synthesize Petri Nets.

The function **generate_min_preregions** generates all minimal preregions for one event. The function **find_irredundant_cover** produces an irredundant set of regions. From this set, a place-irredundant net is generated. This function is discussed in Section 4.2. The function **split_labels** performs the splitting of labels if the initial TS is not an ECTS. This function is discussed in Section 4.3. The function **map_to_PN** is the final step for constructing a PN from the set of regions, which has been described in Section 2.5.

4.1 Generation of Minimal Preregions

The generation of preregions of an event e is based on the fact that any preregion must cover the $ER(e)$. Starting from $ER(e)$, any event with an illegal crossing relation is legalized by adding new states to a set of states containing $ER(e)$ until it becomes a region. An exhaustive search through all the possible legalizations guarantees that all minimal preregions that are predecessors of some event will eventually be found.

The following lemmas, proven in [17], are the basis for the generation of preregions.

LEMMA 4.1 (Violation of region conditions). *Let $TS = (S, E, T, s_{in})$ be a* TS. *Let $r \subset S$ be a subset of states such that r is not a region. Then, there exists an $e \in E$ such that at least one of the following predicates holds:*

1) $\mathbf{in}(e, r) \wedge [\mathbf{enter}(e, r) \vee \mathbf{exit}(e, r)]$
2) $\mathbf{enter}(e, r) \wedge \mathbf{exit}(e, r)$
3) $\mathbf{out}(e, r) \wedge \mathbf{enter}(e, r)$
4) $\mathbf{out}(e, r) \wedge \mathbf{exit}(e, r)$

LEMMA 4.2 (Essential states to become a region). *Let $TS = (S, E, T, s_{in})$ be a* TS. *Let $r \subset S$ be a set of states such that r is not a region. Let $r' \subseteq S$ be a region such that $r \subset r'$. Let $e \in E$ be an event that violates some of the conditions for r to be a region. The following predicates hold:*

1) $\mathbf{in}(e, r) \wedge [\mathbf{enter}(e, r) \vee \mathbf{exit}(e, r)] \Rightarrow \{s \mid \exists s' \in r : (s, e, s') \in T \vee (s', e, s) \in T\} \subseteq r'$
2) $\mathbf{enter}(e, r) \wedge \mathbf{exit}(e, r) \Rightarrow \{s \mid \exists s' \in r : (s, e, s') \in T \vee (s', e, s) \in T\} \subseteq r'$
3) $\mathbf{out}(e, r) \wedge \mathbf{enter}(e, r) \Rightarrow [\{s \mid \exists s' \in r : (s, e, s') \in T\} \subseteq r'] \vee [\{s \mid \exists s' \notin r : (s', e, s) \in T\} \subseteq r']$
4) $\mathbf{out}(e, r) \wedge \mathbf{exit}(e, r) \Rightarrow [\{s \mid \exists s' \in r : (s', e, s) \in T\} \subseteq r'] \vee [\{s \mid \exists s' \notin r : (s, e, s') \in T\} \subseteq r']$.

Note that the asymmetry between **in** and **out** is due to the fact that we always *expand* a set of states to become a region, starting from "minimal" seeds that are excitation regions.

The pseudocode for the function **generate_min_preregions** is shown in Fig. 10.

PROPOSITION 4.1. *The function* **generate_min_preregions** *generates all minimal preregions of the* ECTS *that are predecessors of some* TS *event.*

The restriction to predecessor regions is not so severe because our excitation closure condition is based only on predecessors of a given event. This means that we can find a PN equivalent to any ECTS by looking only at predecessor regions. We can lose with respect to minimality, but efficient implementation is currently more of concern than exact minimization.

Note also that, for Predicates 3 and 43 of Lemma 4.2, two expansions are possible. The algorithm **generate_min_preregions** expands the set of states in both directions, thus implicitly generating a binary exploration tree as shown in Fig. 12. The search tree is, however, reduced in a few ways:

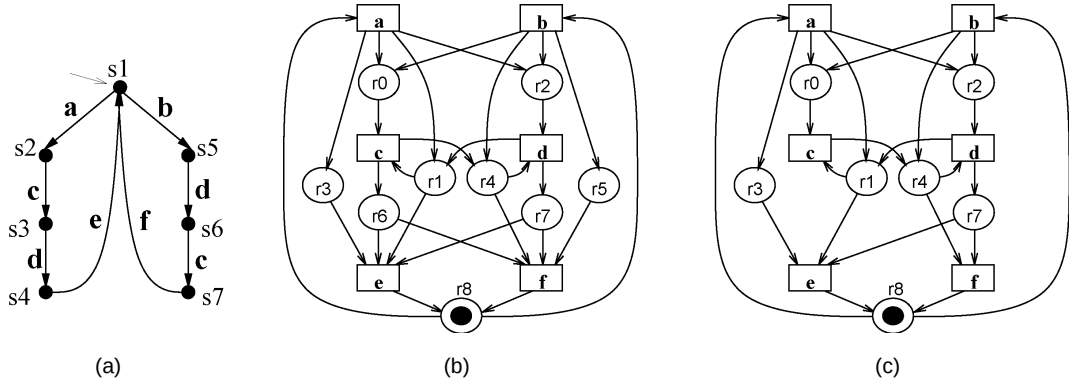


Fig. 11. (a) Transition system. (b) Minimal saturated net. (c) Place-irredundant net.

TABLE 1
ALL MINIMAL PREREGIONS OF THE TRANSITION SYSTEM DEPICTED IN FIG. 11A

preregion	events	preregion	events	preregion	events
$r_0 = \{s_2, s_5, s_6\}$	c	$r_1 = \{s_2, s_4, s_6\}$	c,e	$r_2 = \{s_2, s_3, s_5\}$	d
$r_3 = \{s_2, s_3, s_4\}$	e	$r_4 = \{s_3, s_5, s_7\}$	d,f	$r_5 = \{s_5, s_6, s_7\}$	f
$r_6 = \{s_3, s_4, s_7\}$	e,f	$r_7 = \{s_4, s_6, s_7\}$	e,f	$r_8 = \{s_1\}$	a,b

- If the same set of states is generated more than once, only one branch of the tree is explored.
- If a region is generated during the exploration, then the search along this branch is immediately bounded. This is sound, since the search tree is monotonic in the following sense: each parent vertex of the tree is a subset of the child vertex. Hence, it is not possible to generate minimal regions along the branches starting from another minimal region.
- If the target of the procedure is just to produce an irredundant set of regions, then the excitation closure for a given event is checked on-the-fly and the search is stopped as soon as it is satisfied. This mechanism can be used to produce a locally optimal solution even for very large TSs.

The complexity of region generation is known to be polynomial in the size of the TS [3], which gives an upper bound on the size of the search tree.

4.2 Irredundant Sets of Regions

Let R be a set of regions such that both the excitation closure condition and the event effectiveness condition hold (the set of all minimal preregions of an ECTS satisfies these two conditions). Note that if R satisfies the event effectiveness condition, and some regions are removed from R so that the excitation closure condition still holds, then the event effectiveness condition remains satisfied. Therefore, we need to monitor only the excitation closure condition, while removing redundant regions.

We will illustrate how an irredundant set of places can be calculated by means of the example of Fig. 11. Table 1 presents all minimal preregions of the TS.

As a preliminary step, *essential regions* are calculated. A region r is *essential* if there exists a state s and an event e such that $r \in {}^\circ e$, $s \notin r$ and, for all $r' \in {}^\circ e$, $r' \neq r$, we have $s \in r'$ (i.e., r is the only region that removes from the intersection of preregions a state in which e is not enabled). For example, for event c, we have

$${}^\circ c = \{r_0, r_1\}; \quad ER(c) = \{s_2, s_6\} = r_0 \cap r_1.$$

In this case, both r_0 and r_1 are essential, since none of them can be removed from ${}^\circ c$ without violating its excitation closure. Similarly, we can deduce that r_2, r_4, and r_8 are also essential (r_2 and r_4 are essential for d and r_8 for a, b). Thus, we have four nonessential regions: r_3, r_5, r_6, and r_7.

Next, for each event with nonessential preregions (e and f in the example), all minimal covers are implicitly generated. To reduce the complexity of the problem, essential regions are assumed to be implicitly included in each cover. For event e, we have two minimal covers: $\{r_6\}$ and $\{r_3, r_7\}$. For event f, we also have two minimal covers: $\{r_7\}$ and $\{r_5, r_6\}$. Finding a minimum cost cover can be posed as finding a minimum cost solution of a Boolean equation describing the covering conditions [49], [36]. The equation corresponding to the example is as follows:

$$(r_6 + r_3 \cdot r_7) \cdot (r_7 + r_5 \cdot r_6) = 1.$$

A cost must be assigned to each region, according to the objective function to be minimized, which depends on the application. For example, if we want to minimize the total number of places and arcs (a heuristic measure of the "simplicity" of the PN), then we can assign to each place p a cost of

$$|\,{}^\bullet p\,| + |\,p^\bullet\,| + 1$$

If we want to minimize only the number of places and obtain a place-minimal PN, then the cost of each place is 1.

In our case,

$$\text{cost}(r_3) = \text{cost}(r_5) = 3; \text{cost}(r_6) = \text{cost}(r_7) = 4$$

and two minimum-cost covers exist: $\{r_3, r_7\}$ and $\{r_5, r_6\}$ (the former is shown in Fig. 11c). There is another possible solution ($\{r_6, r_7\}$), but it has nonminimum cost. The existence of two place-minimal nets for this example gives a negative answer to a question posed in [21]: whether there always exists at most one optimal net which could be considered canonical.

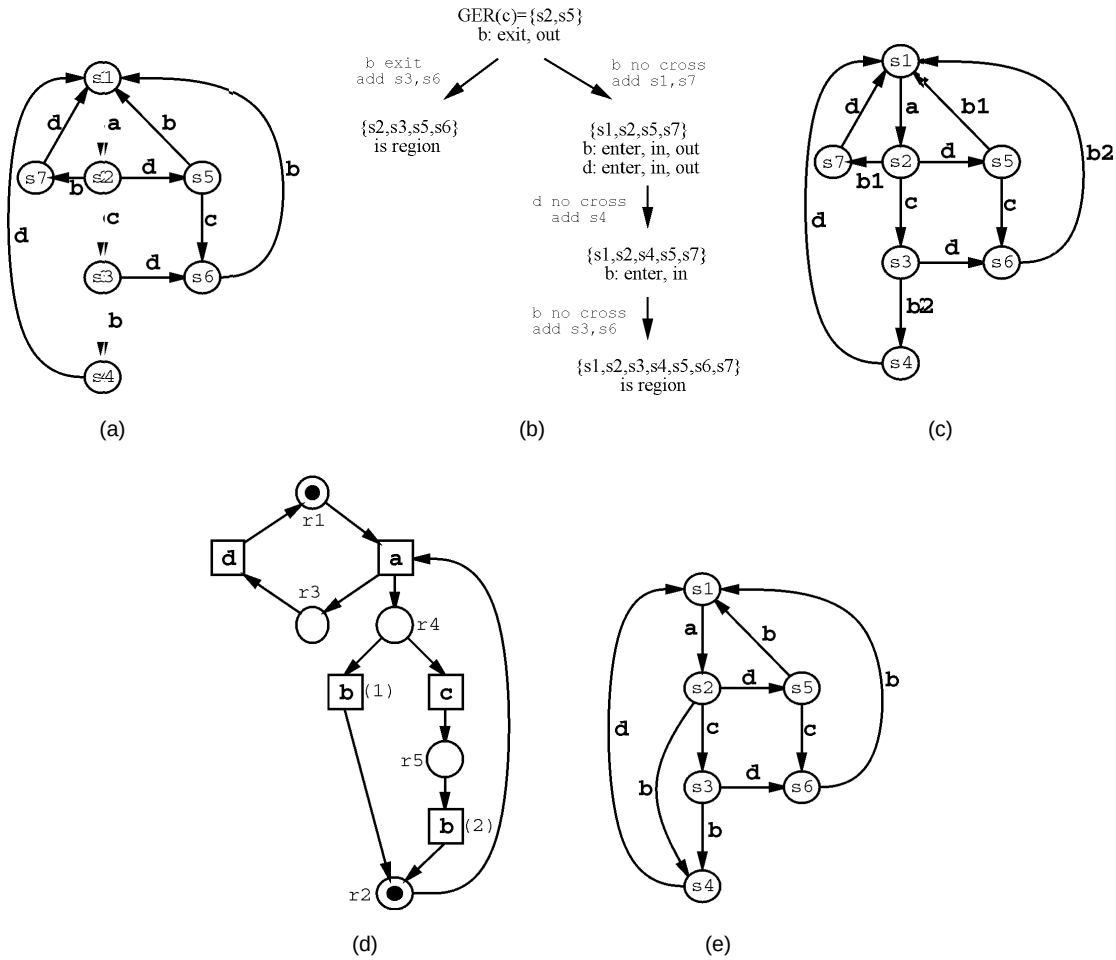


Fig. 12. (a) TS, (b) expansion tree for preregions of event *c*, (c) split-morphic ECTS, (d) PN, (e) RG of the PN.

4.3 Label Splitting

The set of minimal preregions of an event a is calculated by gradually expanding $ER(a)$ to obtain sets of states that do not violate the "entry-exit" relationship. When the excitation closure is not fulfilled (see Definition 3.2), i.e.,

$$\bigcap_{r \in {}^\circ a} r \neq ER(a),$$

some events must be split to make the TS elementary.

The strategy to split events is as follows. During the expansion of $ER(a)$ toward the preregions of a, several sets of states are explored. We focus our attention on sets of states S' such that

$$ER(a) \subseteq S' \subset \bigcap_{r \in {}^\circ a} r.$$

For each of these sets of states, the number of events that violate the region conditions are calculated. Finally, the set that has the least number of "bad" events is selected. If several sets have the same number of "bad" events, the smallest one is selected.

The selected set of states is then forced to be a region. Informally, this is done by splitting the labels of those events that do not fulfill the region conditions. This strategy guarantees that the new intersection of preregions is closer to $ER(a)$.

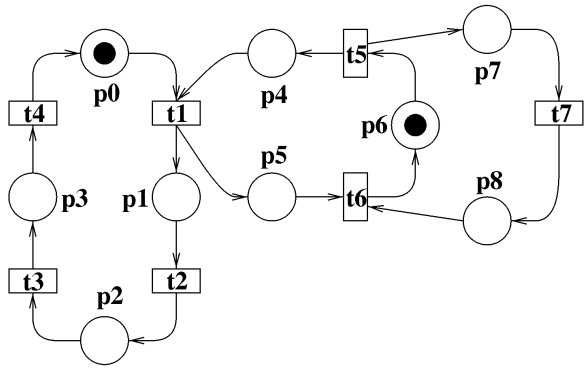
An example is depicted in Figs. 12a and 12b for the preregions of event c. Initially, $ER(c) = \{s_2, s_5\}$ is taken for expansion. Next, two possible legalizations for event b are considered. Further expansions are applied until all branches of the search tree find a region. In this case, regions covering states in $SR(c)$ have been also explored (this type of regions are also valid in case a nonpure PN is sought, as explained in Section 4.4). The example also illustrates how all branches will eventually be pruned, in the worst case, when covering the whole set of states. Let us call r' the intersection of the regions found in the expansion. We have

$$r' = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\} \cap \{s_2, s_3, s_5, s_6\} = \{s_2, s_3, s_5, s_6\}.$$

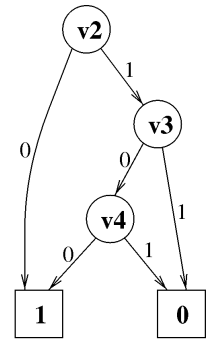
The strategy for label splitting will take all those explored sets r such that

$$\{s_2, s_5\} \subseteq r \subset r'.$$

All three states explored before finding regions are good candidates. However, the set $\{s_2, s_5\}$ is the best one by the fact that only one event violates the crossing conditions and it makes the intersection of pre-regions smaller (closer to ER). Thus, event b is split into two new events (b_1 and b_2) for $\{s_2, s_5\}$ to become a region. The new TS is split-morphic with respect to the original one and is now an ECTS. The corresponding PN is shown in Fig. 12d and its RG in



(a)



(b)

place	v_0	v_1	v_2	v_3	v_4
p_0	0	0	-	-	-
p_1	0	1	-	-	-
p_2	1	0	-	-	-
p_3	1	1	-	-	-
p_4	-	-	0	0	-
p_5	-	-	0	1	-
p_6	-	-	1	0	-
p_6, p_7	-	-	-	-	0
p_8	-	-	-	-	1

(c)

Fig. 13. (a) Petri Net, (b) BDD representing the set of reachable markings, and (c) place encoding.

Fig. 12e. Note that it contains one state less than the original TS, due to the implicit minimization for bisimilar confluent states $s4$ and $s7$.

4.4 Modifications of the Basic Synthesis Method

The above synthesis method can be easily adapted to synthesize different classes of PNs, as sketched below.

nonpure nets: For any event e, the algorithm also explores minimal regions r such that $ER(e) \cup SR(e) \subseteq r$. These regions correspond to self-loop places in the PN and can contribute to fulfill the excitation closure condition.

free-choice nets: A net is said to be *free choice* if for any place p such that $|p\bullet| > 1$, $t \in p\bullet$ implies that $|\bullet t| = 1$. Informally, when a place is a choice, it must be the only predecessor of its successor transitions [27]. This property can be enforced by splitting labels until all choice regions become the only preregions of their postevents.

unique-choice nets: A net is said to be *unique choice* when all the choice places are free (as in free-choice nets) or when no pair of successor transitions can be enabled simultaneously. As an example, all PNs shown in Fig. 14 are unique-choice nets. The method to synthesize unique-choice nets is similar to the one for free-choice nets.

SM-decomposable nets: A state machine (SM) is a subnet composed of a subset of places of a net and all their predecessor and successor transitions, with the condition that any transition of the subnet has only one predecessor place and one successor place [27]. A net is SM-decomposable if each place of the net belongs to some state machine. The synthesis of SM-decomposable PNs is based on the observation that any set of disjoint regions covering all states of a TS, i.e., a partition of the set of states into regions, corresponds to an SM in the PN [5].

unsafe nets: Based on the theory presented in [3] and incorporating label splitting to extend the method to any class of TS [31].

Further details on the synthesis of different classes of PNs can be found in [17].

4.5 Symbolic Representation with Binary Decision Diagrams

In this section, we briefly explain how sets of states can be represented by means of Boolean functions and efficiently manipulated by using Binary Decision Diagrams (BDDs) [35], [10]. As an example, it will be shown how traversal of the reachability set of markings can be done symbolically using BDDs.

Given the set P of places of a PN, a set M of (safe) markings over P can be represented by its *characteristic function*, denoted χ_M, that is a Boolean function that evaluates to 1 for each marking in M. In particular, the set of reachable markings of a PN in which a place p_i is marked will be denoted by χ_i.

A BDD is a directed acyclic graph with one root and two leaf nodes (0 and 1). Each nonleaf node is labeled with a Boolean variable and has two outgoing arcs with labels 0 and 1. A BDD represents a Boolean function as follows: Each variable assignment has a corresponding path that goes from the root node to one of the leaf nodes. The label of the leaf node is the value of the function for that assignment. As an example, the BDD depicted in Fig. 13b represents the function $f(v_2, v_3, v_4) = \overline{v_2} + \overline{v_3}\,\overline{v_4}$ corresponding to

the reachability set of the PN in Fig. 13a. The value of the function for the assignment $v_2 = v_3 = 1$ and $v_4 = 0$ is 0. This assignment corresponds to the path $v_2 \xrightarrow{1} v_3 \xrightarrow{1} 0$. Note that the function has 20 minterms and that each of them corresponds to a reachable marking of the function. Here are some examples of how the markings (represented as sets of marked places) are encoded:

$$(0,\ 1,\ 0,\ 1,\ 1) \rightarrow \{p_1, p_5, p_8\}$$
$$(1,\ 1,\ 1,\ 0,\ 0) \rightarrow \{p_3, p_6\}$$
$$(0,\ 0,\ 1,\ 0,\ 1) \rightarrow \text{unreachable marking.}$$

We refer the reader to [10] for further details on how to manipulate Boolean functions efficiently by means of BDDs. With such a representation, the basic operations on sets of states (union, intersection, complement) can be mimicked as Boolean operations on Boolean functions (or, and, not). Moreover, such functions can have a compact representation. In [46], some examples are shown in which graphs with 10^{18} states can be represented with BDDs having 10^3 nodes by using very naive encodings (one Boolean variable per place).

Starting from a simple example of a PN specification shown in Fig. 13a we will explain the following steps.

- Selection of encoding variables for representing markings of individual places. Instead of the simplest naive encoding, one variable per place, we use a more dense encoding based on state machine decomposition of the original PN.
- Traverse the reachability space of the net starting from the initial marking until the fixed point is reached. The traversal is done symbolically, using the BDD representation of Boolean characteristic functions for sets of markings and the transition relation of the net.
- As a result of the computation we will get a BDD representation of the characteristic function of the reachability set. This function has the encoding variables as its arguments.
- Given this function we show how different operations with sets of markings can be performed.

4.5.1 Encoding

The dense encoding used for the markings of the PN of Fig. 13 is based on the observation that the sets of places $P_1 = \{p_0, p_1, p_2, p_3\}$, $P_2 = \{p_4, p_5, p_6\}$, and $P_3 = \{p_6, p_7, p_8\}$ define three state machines, *SM*1, *SM*2, and *SM*3, of the PN [27], [20] with the following sets of transitions $T_1 = \{t_1, t_2, t_3, t_4\}$, $T_2 = \{t_1, t_5, t_6\}$, and $T_3 = \{t_5, t_6, t_7\}$, respectively. This information can be structurally obtained by using algebraic methods [20]. State machines correspond to place-invariants of the PN and preserve their token count in all reachable markings. Given the initial marking of the net, at most one of the places of each state machine will be marked at each marking. Thus, the following encoding can be proposed: two Boolean variables (v_0 and v_1) can be used to encode the token in M_1, two Boolean variables (v_2 and v_3) for M_2. Only one Boolean variable (v_4) is sufficient for M_3, since M_2 already uniquely encodes p_6 ($v_2 = 1 \Leftrightarrow p_6$ has a token) and only places p_7 and p_8 must be distinguished. The table in Fig. 13c proposes an encoding for the places that leads to the following characteristics functions for places:

place, p	χ_p	place, p	χ_p
p_0	$\overline{v_0}\,\overline{v_1}$	p_1	$\overline{v_0}\,v_1$
p_2	$v_0\,\overline{v_1}$	p_3	$v_0 v_1$
p_4	$\overline{v_2}\,\overline{v_3}$	p_5	$\overline{v_2}\,v_3$
p_6	$v_2\,\overline{v_3}$	p_7	$\overline{v_4}\left(\overline{v_2} + v_3\right)$
p_8	v_4	–	–

Note that, since variable v_4 has value 0 both for places p_6 and p_7 and its characteristic function χ_{p_6} depends only on variables corresponding to the second state machine, *SM*2, the characteristic function for p_7 is

$$\chi_{p_7} = \overline{v_4}\,\overline{\chi_{p_6}} = \overline{v_4}\left(\overline{v_2} + v_3\right).$$

4.5.2 Transition Function and Reachable Markings

The methods used for deriving the transition function and calculating the reachable markings of a PN are similar to those used for reachability analysis and equivalence checking of finite state machines [26].

For calculating the transition function, let us first introduce the characteristic functions for two important sets related to a transition $t \in T$:

$$\mathrm{E}(t) = \bigwedge_{p_i \in {}^\bullet t} \chi_{p_i}(V) \quad (t \text{ enabled}),$$
$$\mathrm{ASM}(t) = \bigwedge_{p_i \in t^\bullet} \chi_{p_i}(V) \quad (\text{all successors marked}).$$

Function $E(t)$ ($ASM(t)$) states that all input (output) places of transition t contain a token. For example, for transition t_1 in Fig. 13a

$$E(t_1) = \chi_{p_0}(V)\chi_{p_4}(V) = \overline{v_0}\,\overline{v_1}\,\overline{v_2}\,\overline{v_3}$$

and

$$ASM(t) = \chi_{p_1}(V)\chi_{p_5}(V) = \overline{v_0}\,v_1\,\overline{v_2}\,v_3.$$

Let M_N be the set of all possible markings of a PN N. The *transition function* of a Petri Net is a function

$$\delta_N : 2^{M_N} \times T \rightarrow 2^{M_N}$$

that transforms, for each transition, t, a set of markings M into a new set of markings M' one-step reachable from M by firing transition t:

$$M' = \delta_N(M, t) = \left\{ m_2 \in M_N : \exists m_1 \in M, m_1 \xrightarrow{t} m_2 \right\}.$$

We illustrate the calculation of the transition function with an example (for a more detailed explanation of the algorithms, see [46]). Assume that, in the example of Fig. 13a, we calculate $M' = \delta_N(M, t_1)$ given the set of markings: $M = \{\{p_0, p_4, p_7\}, \{p_0, p_4, p_8\}, \{p_3, p_6\}\}$ represented by the characteristic Boolean function:

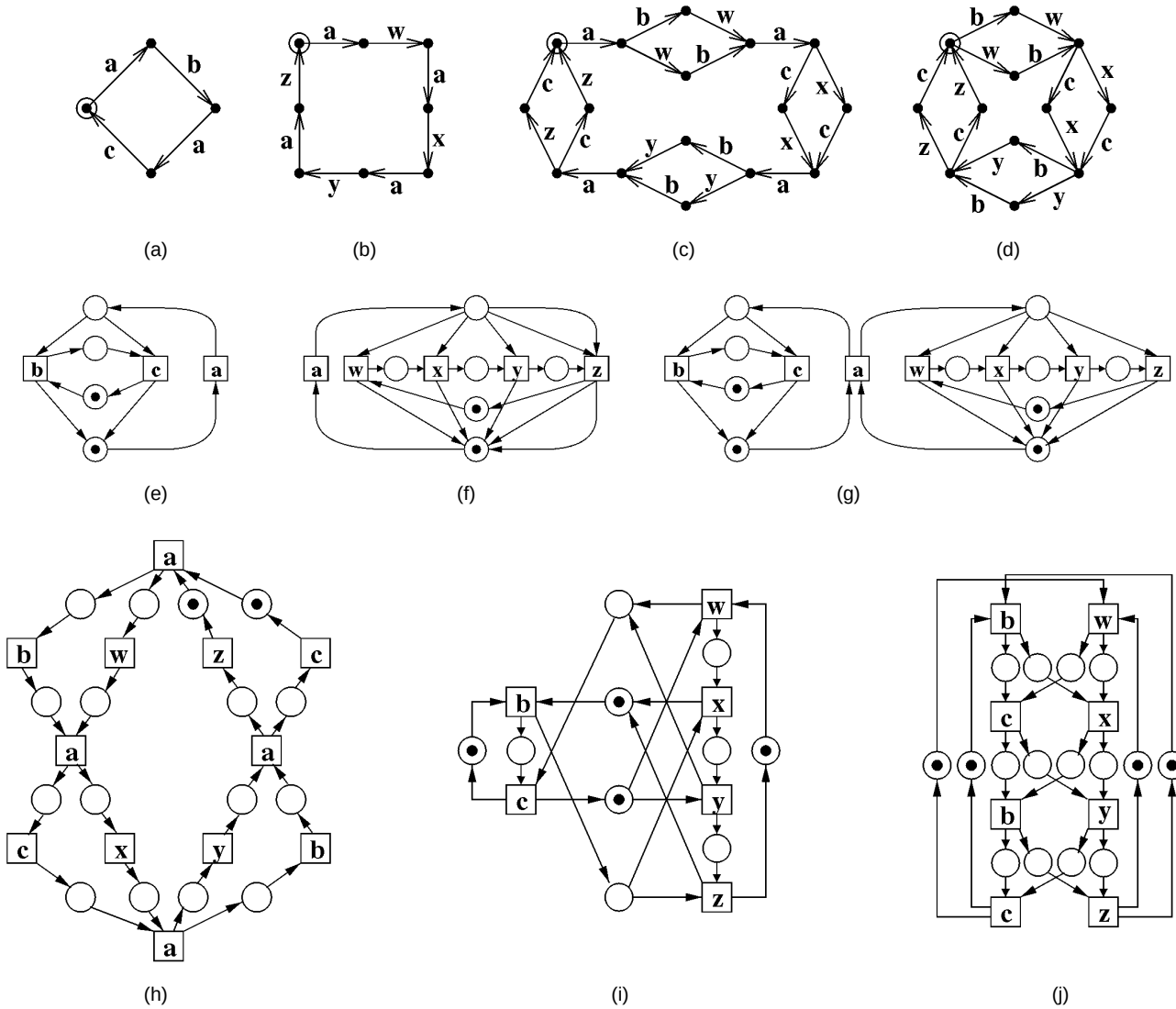


Fig. 14. Synchronized composition of transition systems and synthesis of minimized and free-choice Petri Nets (for simplicity, 1-input/1-output places are represented by a transition-transition arc). (a) P_1, (b) P_2, (c) $P_1 \| P_2$, (d) $(P_1 \| P_2) \backslash \{a\}$, (e) P_1, (f) P_2, (f) $P_1 \| P_2$ (minimized), (g) $P_1 \| P_2$ (free-choice), (h) $(P_1 \| P_2) \backslash \{a\}$ (minmized), (i) $(P_1 \| P_2) \backslash \{a\}$ (free choice).

$$M = \chi_{p_0}\chi_{p_4}\chi_{p_7} + \chi_{p_0}\chi_{p_4}\chi_{p_8} + \chi_{p_3}\chi_{p_5}\chi_{p_7} = \overline{v_0}\,\overline{v_1}\,\overline{v_2}\,\overline{v_3} + v_0 v_1 \overline{v_2}\, v_3 \overline{v_4}.$$

First, by calculating $M \cdot E(t_1) = \overline{v_0}\,\overline{v_1}\,\overline{v_2}\,\overline{v_3}$, one selects those markings from M in which t_1 is enabled. After that, we determine all literals that are logically implied by the characteristic functions of input places of transition t_1, i.e., places p_0 and p_4. The following implications hold: $\chi_{p_0} \Rightarrow \overline{v_0}$, $\chi_{p_0} \Rightarrow \overline{v_1}$, $\chi_{p_4} \Rightarrow \overline{v_2}$, and $\chi_{p_4} \Rightarrow \overline{v_3}$. All implied literals should be cofactored from function $M \cdot E(t)$.[4] The result is $(M \cdot E(t))_{\overline{v_0}\,\overline{v_1}\,\overline{v_2}\,\overline{v_3}} \equiv 1$. Informally, this corresponds to removing predecessor places of t_1 from the characteristic function. The final step is adding successor places of t_1 into the characteristic function, which is done by calculating conjunction of the previous result with $ASM(t_1)$:

$$\left(M \cdot E(t)\right)_{\overline{v_0}\,\overline{v_1}\,\overline{v_2}\,\overline{v_3}} \cdot ASM\left(t_1\right) = \overline{v_0}\, v_1\, \overline{v_2}\, v_3.$$

This result is the characteristic function of M', which can be considered as a Boolean version of the marking equation of the PN. The existential quantification of t from the above formula gives the set of markings $\delta_N(M)$ reachable by firing any *one* enabled transition from a marking in M.[5]

In such a way, starting from the initial marking, by iterative application of the transition function we calculate the characteristic function of the reachability set until the fixed

4. The cofactor of $f(v_1, \ldots, v_i, \ldots, v_n)$ with respect to literal v_i, denoted by f_{v_i}, is $f(v_1, \ldots, 1, \ldots, v_n)$ and with respect to literal $\overline{v_i}$, $f_{\overline{v_i}}$, is $f(v_1, \ldots, 0, \ldots, v_n)$. The notion of cofactor can be generalized to a product of literals, e.g., $f_{v_i, v_2} = (f_{v_1})_{v_2}$ [10].

5. The existential abstraction of $f(v_1, \ldots, v_i, \ldots, v_n)$ with respect to v_i is $\exists_{v_i}(f) = f_{v_i} + f_{\overline{v_i}}$ [10].

TABLE 2
RESULTS ON SYNTHESIS MINIMIZED PNS AND FREE-CHOICE PNS

example	initial PN				minimized PN				minimized FC PN			CPU
	P	T	F	M	P	T	F	M	P	T	F	(secs)
alloc-outbound	17	18	36	17	13	14	37	16	16	17	34	0.1
clock	10	10	20	10	8	5	26	10	10	10	20	0.1
dff (*)	20	20	44	20	8	8	29	10	10	14	28	0.7
espinalt	27	25	57	27	19	20	52	26	26	24	54	1.1
fair_arb	13	20	40	13	11	10	31	13	13	20	40	0.2
future	30	28	60	36	18	16	38	36	30	28	60	1.1
gcd-ra (*)	66	58	136	3,240	43	40	110	3,090	64	56	136	27.3
intel_div3	8	8	16	8	8	6	23	8	8	8	16	0.1
intel_edge	28	36	72	28	17	25	111	25	24	32	64	6.5
isend (*)	56	44	116	53	20	19	89	36	39	36	100	8.1
lin_edac93	14	12	28	20	10	8	22	20	14	12	28	0.3
master-read	36	26	72	8,932	33	26	66	8,932	33	26	66	5.7
pe-rcv-ifc	43	38	96	46	20	20	105	36	37	32	87	5.6
pulse	12	12	24	12	7	6	20	12	12	12	24	0.1
rcv-setup	14	15	32	14	10	10	34	11	11	12	26	0.2
vme_read (*)	41	32	84	255	32	27	114	251	38	30	92	9.7
vme_write (*)	49	36	100	821	38	31	139	817	46	34	112	20.8
Total	484	438	1,033	13,552	315	291	1,046	13,349	431	403	987	
Reduction	1.00	1.00	1.00	1.00	0.65	0.66	1.01	0.99	0.89	0.92	0.96	
arcs/node	1.12				1.73				1.18			

((*) silent events hidden before synthesis)

point in calculation is reached. The resulting function for the reachability set for the example considered above is $f(v_2, v_3, v_4) = \overline{v_2} + \overline{v_3}\,\overline{v_4}$. All calculations are done using a BDD representation of the corresponding characteristic Boolean functions.

4.5.3 Minimal Preregions and Excitation Closure

As an example of the type of operations that can be performed with BDDs, we give some further details on the calculation of minimal preregions and excitation closure.

Given a set of states and a transition function, successor and predecessor states reachable in one step from the given set can be obtained by applying the direct and inverse transition function, respectively. This is the main operation performed in the function `expand_states` (see Fig. 10) to legalize events according to the conditions of Lemma 4.2 and obtain minimal preregions.

Checking the excitation closure for an event after the set of minimal preregions has been obtained is now reduced to calculating their intersection (Boolean AND operation) and checking its equivalence to the ER of the event.[6]

5 APPLICATIONS

The methodology presented in this paper has its main application in the area of analysis and synthesis of concurrent systems. Representing a concurrent system with an event-based model (Petri Net) instead of a state-based model (transition system) has some clear advantages:

- The relations between events are explicit in the model.
- The representation is usually much more succinct and does not suffer from the state explosion problem.
- Some properties can be verified at the structural level, without requiring the enumeration of the states of the system.

PN synthesis, since it starts from a state-based representation and reconstructs the relations, obviously applies mostly to the first aspect. Several applications of this reconstruction have been outlined in Section 1. In this section, we illustrate the usefulness of the method in two synthesis approaches:

top-down approach: A system is synthesized by composing specifications of communicating subsystems.

bottom-up approach: A system is analyzed by composing fragments corresponding to components.

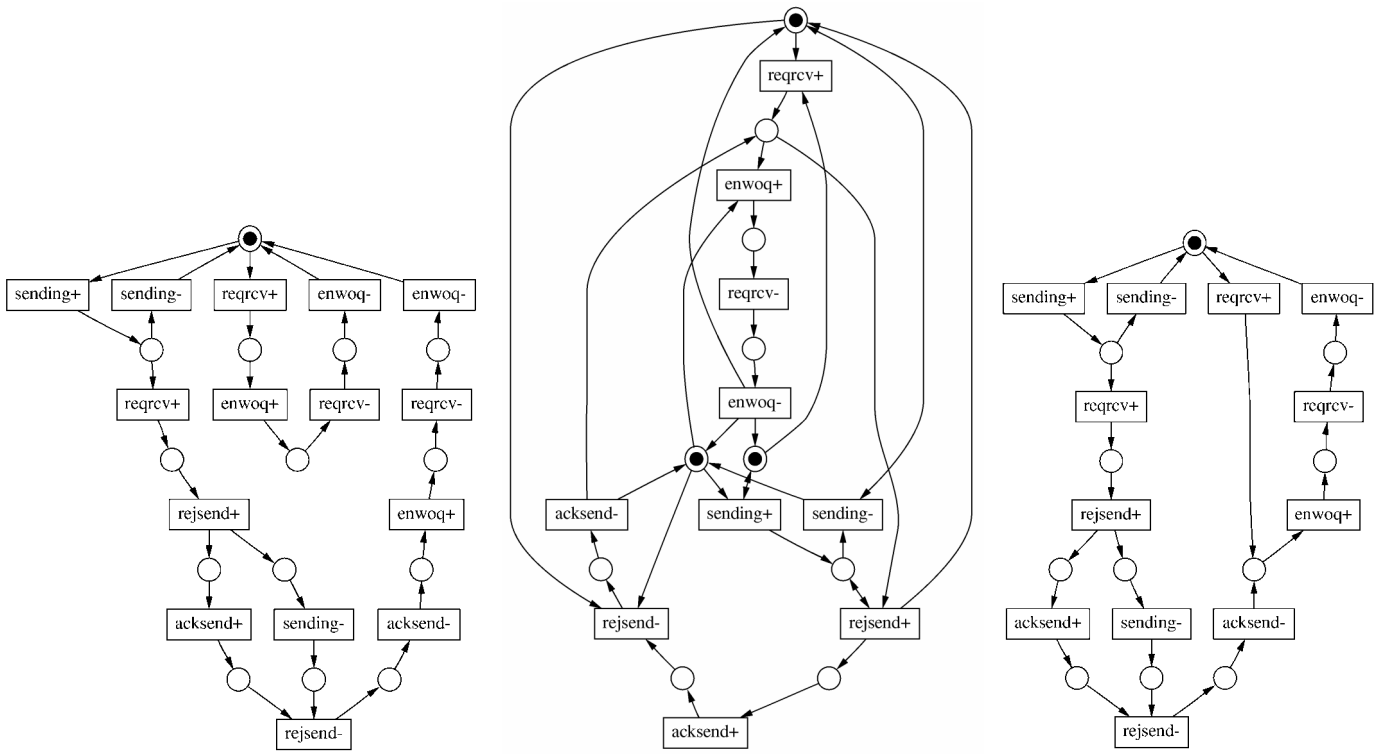
5.1 Top-Down Approach: Petri Net Composition

PN synthesis can be applied to the derivation of specifications obtained by the composition of processes. The semantics of composition is defined as follows: Let us have two processes, each with an alphabet that labels its events (the alphabets may not be disjoint). The composition of the processes is another process that models their concurrent behavior synchronizing on pairs of events with the same label. We refer the reader to [37], [57], [51] for a more formal definition of composition.

The methods known so far to compose PNs are structural [51], [57], [19], i.e., they derive a new PN by combining nodes of the original PNs. These methods may, however, introduce redundancy in the resulting PN because they are conservative when calculating the pairs of synchronizing transitions. Moreover, they do not allow one to obtain, e.g., a Free-Choice composed PN, even when this is possible.

Fig. 14 depicts an example of how PN synthesis can be used for the synthesis of concurrent systems by composition of subsystems. The example obtains a system from two concurrent processes, P_1 and P_2, that synchronize through a

6. Checking equivalence is a constant time operation in most BDD packages.



(a) (b) (c)

Fig. 15. **rcv-setup**: (a) Initial Petri Net, (b) Petri Net with minimum transition count, and (c) minimized free-choice PN.

TABLE 3
SYNTHESIS OF PETRI NETS FROM SPEED-INDEPENDENT CIRCUITS

circuit	signals	states	places	transitions	arcs	markings	CPU (secs)
unsafe	5	22	17	12	46	22	0.7
a4_tflo1	8	20	17	16	40	20	0.2
a_10_dr2	50	9,408	93	100	336	9,408	1,582.4
a_11_sen	19	85	38	38	93	85	4.6
dags55	19	130	33	38	187	130	54.9

common event *a*. By composing TSs and eventually hiding the nonobservable events of the communication, different PNs can be derived. Fig. 14 shows specifications for $P_1 \| P_2$ and $(P_1 \| P_2) \setminus \{a\}$ (after hiding event *a*).

For each case, two different PNs are obtained: one by minimizing the number of transitions of the PN and the other by forcing the net to be free choice. Each case pursues different goals. The former attempts to find the most succinct representation for the system by minimizing the transition and place count of the PN. The latter attempts to minimize the flow relation of the PN, i.e., the number of arcs, in order to find a more readable representation from the point of view of the designer, at the expense of losing optimality in the transition and place counts. Note that the free-choice nets[7] of the example have multiple transitions with the same label, produced by the extra label splitting required to force the free-choice conditions.

Minimal place and transition counts can be better for manipulating nets by automatic synthesis tools, whereas more readable representations, such as free-choice nets, may be better for design frameworks with a high interaction with the designer. All the PNs shown in the figure have been obtained automatically by the synthesis tool **petrify** [16].

This approach allows one to create an efficient link between high-level languages, such as CSP or CCS, and Petri Nets. Through the modeling of the language constructs with basic primitives and the composition of such primitives, a netlist of communicating subsystems can be obtained [56]. A Petri Net for the whole system can be derived by composing the Petri Nets that model the behavior of the basic primitives. Such an approach has been used in [47] to synthesize asynchronous circuits from CSP-like descriptions.

5.1.1 Efficiency of PN Synthesis

Table 2 describes the results of the application of our algorithms to the minimization of labeled Petri Nets. The examples (taken from the set of standard benchmarks for asynchronous control circuits [34]) correspond to specifications of asynchronous circuits that have been produced manually by system designers. *P*, *T*, *F*, and *M* are the numbers of places, transitions, arcs, and markings, respectively.

7. In fact, the resulting nets are Marked Graphs, since they contain no choice places.

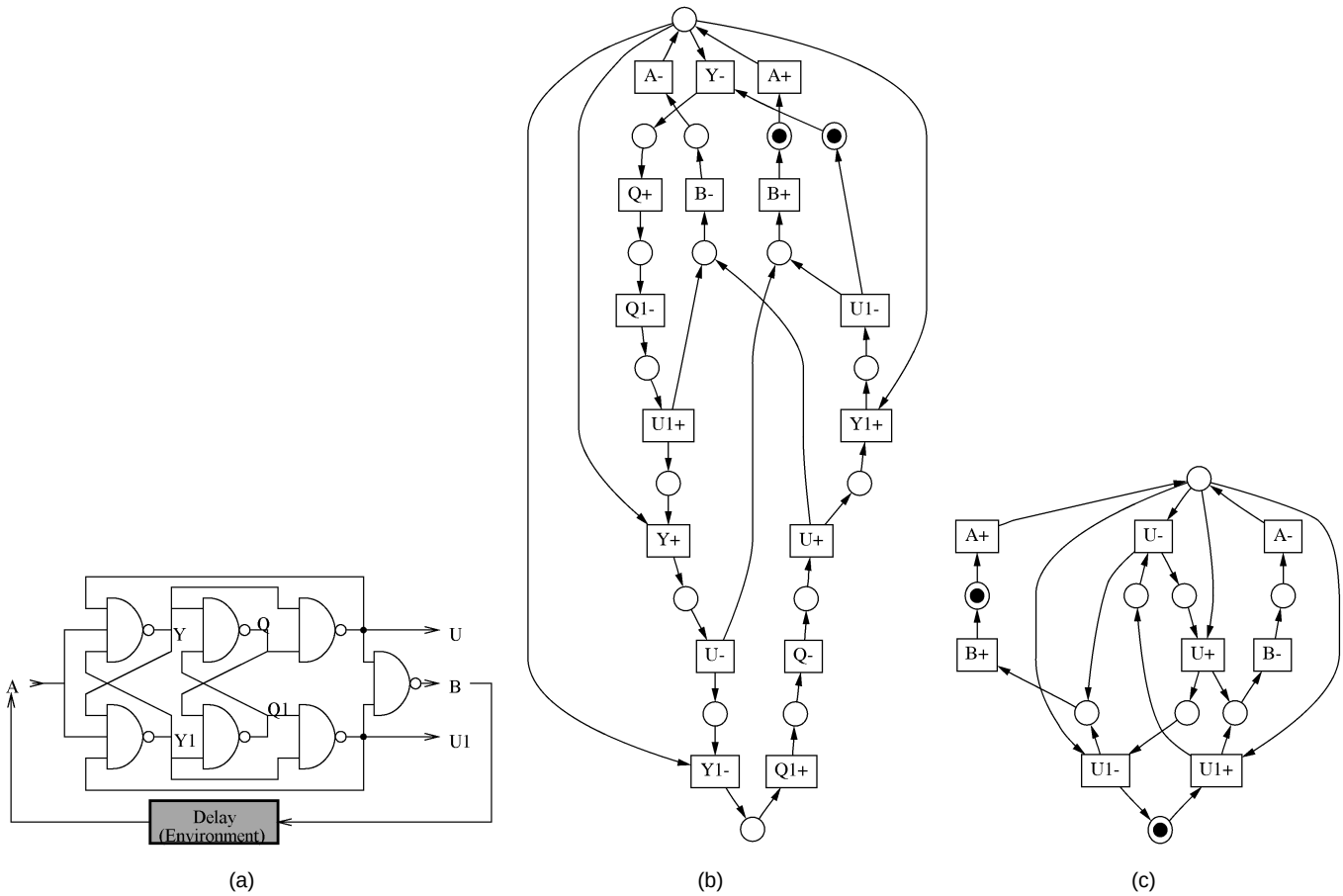


Fig. 16. (a) Speed-independent circuit (initially, A = Q = U1 = 0, B = Q1 = U = Y = Y1 = 1), (b) Petri Net with bisimilar behavior, (c) Petri Net describing the observable behavior (+ and – indicate rising and falling transitions of the signals).

The table illustrates the trade-off between succinctness and readability of the PNs. We have used the ratio *arcs/node* as a measure of (un)readability. The results show that significant reductions in the number of nodes can be obtained (35 percent less nodes, 1.73 arcs / node on average). If priority is given to readability (free-choice nets), satisfactory reductions are still achieved (10 percent less nodes, 1.18 arcs / node on average). Note also that the number of markings is sometimes reduced, since equivalent states can be merged. Fig. 15 depicts the synthesized PNs for one of the examples of Table 2.

5.2 Bottom-Up Approach: Analysis of Concurrent Systems

PN synthesis can also be used for the analysis of systems. Table 3 describes the results of the application of our algorithms to the synthesis of Petri Nets from TSs obtained from speed-independent circuits (all examples are described in [32]). This can be used to produce a user-readable description of the functionality of a circuit in the form of a timing diagram-like labeled Petri Net (a Signal Transition Graph, STG). Another potential application is to optimize the input to direct synthesis methods that have been devised for Petri Nets using both synchronous and asynchronous circuit design techniques ([39], [8], [44]).

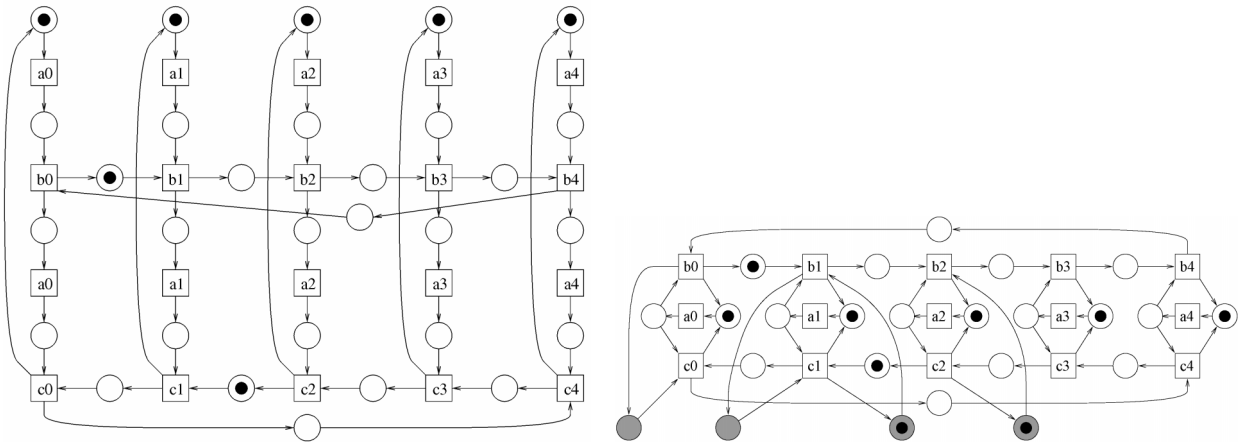
Fig. 16 illustrates one of the examples shown in Table 3 (a4_tflo1). Initially, a TS is derived by calculating all reachable states of the circuit (symbolic techniques can be used here). A Petri Net capturing the behavior of all gate outputs can be obtained by synthesis (Fig. 16b). Finally, since the user is probably only interested in the observable behavior of the circuit, a projection of the TS onto the observable signals can be done and a simplified Petri Net can be obtained (Fig. 16c). The simplified Petri Net can now be used to resynthesize the circuit and generate different implementations.

5.3 Application to Large Transition Systems

The manipulation of the state space by means of BDDs enables synthesis of Petri Nets with large reachability graphs. We have chosen one scalable example to illustrate this fact (see Fig. 17). It is an n stage pipeline with forward synchronization through events $b_i \rightarrow b_{i+1}$ and backward synchronization through events $c_i \leftarrow c_{i+1}$. Fig. 17a depicts a five-stage pipeline. In general, the Petri Net has $6n$ places, $4n$ transitions, and $12n$ arcs, n being the number of stages.

The minimization of the Petri Net is not trivial (see Fig. 17b) since, besides the regular structure derived from the pipeline, it requires some extra places (shadowed in the figure) for the proper initialization of the firing sequences. The final Petri Net has $4n + 4$ places, $3n$ transitions, and $10n + 8$ arcs.

It is worth noting that small BDDs (e.g., 1,117 nodes) can represent large state spaces (e.g., 2.2×10^{17} states). From the total CPU time reported in the table of results, only a small fraction (about 10 percent) is used to synthesize the final



(a) (b)

Example	States	BDD size	CPU (secs)
pipe-5	576	106	1
pipe-10	3.9×10^4	220	5
pipe-20	8.2×10^7	427	46
pipe-30	1.3×10^{11}	659	184
pipe-40	1.7×10^{14}	893	552
pipe-50	2.2×10^{17}	1,117	1,425

(c)

Fig. 17. (a) Five-stage pipeline, (b) minimized Petri Net, (c) experimental results.

Petri Net. The rest of the running time is used to calculate the reachability graph and find a good encoding for the final synthesis step.[8] The reported BDD sizes correspond to the characteristic function of the reachable states after having found an efficient encoding.

6 Conclusions

Petri Nets are an appropriate formalism to describe the behavior of systems with concurrency, causality, and conflicts between events. For this type of systems, the method presented in this paper allows one to transform different models (CSP, CCS, FSMs, PNs) into a unique formalism for which synthesis, analysis, composition, and verification tools can be built.

Synthesizing Petri Nets from state-based models is a task of reverse engineering that abstracts the temporal dimension from a flat description of the sequences of events produced by the system. The synthesis method discovers the actual temporal relations among the events. The cooperation among the notions of ETS, *region*, and *excitation region* in the same method has been crucial to derive efficient algorithms.

One can ask about the real need to generate a flat description (a TS) from compact models (CSP, CCS, or even PNs) that can describe temporal relations in a natural way. An alternative way of doing so would be to obtain PNs by means of syntax-directed translation from those models. We have shown some experiments that illustrate the interest in going through transition systems. The results on synthesis from an STG into an STG showed how the behavioral descriptions proposed by the designers can be usually made more compact (some temporal relations are not easy to describe and designers are often tempted to derive an FSM-like description). The discussion about PN composition also showed that previous composition methods can produce redundant specifications. Much simpler descriptions can be obtained by first generating a TS, removing internal events (not relevant to the external behavior of the system), and deriving a PN.

Generating a TS from a high-level description (such as CSP) may suffer from the state explosion problem, thus making manipulations at the TS level tedious or even impractical. For this reason, we have chosen to use a symbolic (BDD-based) representation of the TS. Even though BDDs do not always guarantee compactness, we have observed that the regular interleaving of events manifested by highly concurrent systems is well-captured by symbolic representations.

This work has been mainly motivated by the activities carried out by the authors in the area of asynchronous circuits. However, the method for PN synthesis presented here can be equally applied for optimization of control structures of parallel programs [50] or manufacturing systems [58]. The wide applicability of the method opens new possibilities to create a framework with tools for synthesis, analysis, and verification in which the designer can freely choose and mix different specification formalisms.

8. The tool **`petrify`** attempts to improve the encoding of the reachability space after the latter has been calculated. Although this helps making the forthcoming steps more efficient, it becomes the dominant part of the running time in some cases.

APPENDIX: PROOFS OF THE MAIN STATEMENTS

Proof of Theorem 3.1

The following two lemmas proven in [17] are required for the proof.

LEMMA 6.1. *Let* $TS = (S, E, T, s_{in})$ *be an* ECTS *and let* $s_1, s_2 \in S$ *be two states such that* $R_{s_1} = R_{s_2}$. *Then, for each event* $e \in E$,

$$\left[\exists s_1' \in S : (s_1, e, s_1') \in T\right] \Leftrightarrow \left[\exists s_2' \in S : (s_2, e, s_2') \in T\right].$$

LEMMA 6.2. *Let* $TS = (S, E, T, s_{in})$ *be an* ECTS *and let* $s_1, s_2 \in S$ *be two states such that* $R_{s_1} = R_{s_2}$. *Then, for each event* $e \in E$,

$$(s_1, e, s_1') \in T \wedge (s_2, e, s_2') \in T \Rightarrow R_{s_1'} = R_{s_2'}.$$

PROOF (sketch). Let R be the following binary relation between S and $[S]$:

$$s_i R s[\pi_j] \Leftrightarrow s_i \in \pi_j.$$

From Definition 2.6 and the construction of $TS_{\mathcal{R}}$, it follows that R is a bisimulation between TS and $TS_{\mathcal{R}}$:

- (i a, i b) trivially hold, since R is a surjective mapping from S onto $[S]$.
- (ii a): Assume that $(s_1, e, s_2) \in T$. Let $s_1 \in \pi_1$ and $s_2 \in \pi_2$. Since TS is an ECTS, π_1 and π_2 are different classes. $s[\pi_1]$ and $s[\pi_2]$ are the only states related with s_1 and s_2, respectively and, by the construction of $TS_{\mathcal{R}}$, $(s[\pi_1], e, s[\pi_2]) \in T_{\mathcal{R}}$.
- (ii b): Assume that $(s[\pi_1], e, s[\pi_2]) \in T_{\mathcal{R}}$. Then, for each $s_1 \in \pi_1$, there exists $(s_1, e, s_2) \in T$ such that $s_2 \in \pi_2$ (this is ensured by Lemmas 6.1 and 6.2). □

Proof of Theorem 3.2

PROOF.

1) It was proven in [17] (Lemma 3.1) that if the state separation property (A4) is satisfied for a TS, then the event effectiveness property (A5′) holds. Hence, we need to prove excitation closure (A4′).

   Due to (A5′), $\forall e \in E$ intersection $\bigcap_{r \in {}^\circ e} r$ is defined since ${}^\circ e \neq \emptyset$. By definition of preregion, $GER(e) \subseteq \bigcap_{r \in {}^\circ e} r$, and, with the help of axiom A2, $\forall e \in E : \bigcap_{r \in {}^\circ e} r \neq \emptyset$. It remains to show that $\bigcap_{r \in {}^\circ e} r \subseteq GER(e)$. Let e be an arbitrary event and let state s belong to $\bigcap_{r \in {}^\circ e} r$. Then, for any region $r \in {}^\circ e : s \in r$. Hence, the left part of the implication of the axiom A5, ${}^\circ e \subseteq R_s$, holds. By assumption, the TS is elementary, hence, the right part of the implication must hold and, therefore, $s \xrightarrow{e}$. Hence, $s \in GER(e)$. Therefore, $s \in \bigcap_{r \in {}^\circ e} r \Rightarrow s$ and, consequently, $\bigcap_{r \in {}^\circ e} r \subseteq GER(e)$. Thus, the TS is excitation-closed.

2) From the definition of $TS_{\mathcal{R}}$, it follows that, for an ECTS,

   r is a region of $TS \Leftrightarrow [r]$ is a region of $TS_{\mathcal{R}}$.

   Since there is a one-to-one correspondence between regions of TS and $TS_{\mathcal{R}}$, it immediately follows that excitation closure and event effectiveness are preserved in $TS_{\mathcal{R}}$. Hence, (A5) also holds. Really, the following conditions are true: ${}^\circ e \neq \emptyset$ and $\forall e \in E \forall s \in S : s \in \bigcap_{r \in {}^\circ e} r \Rightarrow s \in GER(e)$. Let us choose an arbitrary event $e \in E$ and a state $s \in S$ and let us assume that ${}^\circ e \subseteq R_s$. Then, $\forall r \in {}^\circ e \Rightarrow r \in R_s$ and, hence, $\forall r \in {}^\circ e \Rightarrow s \in r$. From the latter, we deduce that $s \in \bigcap_{r \in {}^\circ e}$. Then, by the excitation closure condition, we have $s \in GER(e)$; hence, $s \xrightarrow{e}$.

   (A4) is directly enforced by the definition of the set of states of $TS_{\mathcal{R}}$. Therefore, $TS_{\mathcal{R}}$ is elementary. □

Proof of Theorem 3.3

PROOF. Let us actually prove a stronger statement, assuming only that the TS satisfies state separation (A4). Assume the opposite, i.e., axiom A4 is satisfied and there is a pair of bisimilar states s and s', $s \neq s'$, for which the confluence condition is satisfied.

It follows from axiom A4 that there is a region r such that $r \in R_s$ and $r \notin R_{s'}$. Due to the confluence condition, the following condition holds: There is a state $s'' \in S$ and two sequences of transitions $\sigma, \sigma' \in T^*$ such that $s \xrightarrow{\sigma} s'' \wedge s \xrightarrow{\sigma'} s''$. Let s'' be the first such confluence state, i.e., there is no state s''' such that $s''' \neq s''$ and $s''' \in s \xrightarrow{\sigma} s'' \wedge s''' \in s \xrightarrow{\sigma'} s''$.

Two cases are possible:

1) $s'' \notin r$, i.e., r intersects σ since one of the transitions from σ before s'' exits r. Let e_1 be the label of the first transition $s_1 \xrightarrow{e_1} s_2$ that exits r along σ (see Fig. 18), i.e., $s_1 \in r$ and $s_2 \notin r$. Since s and s' are bisimilar, there must be $s' \xrightarrow{\sigma}$. Then, by the definition of a region (r), $\exists s_1' \xrightarrow{e_1} s_2' \in s' \xrightarrow{\sigma}$ such that $s_1' \in r$ and $s_2' \notin r$. Since s'' is chosen to be the first confluence state, $s_1' \neq s_1$.

   Since $s' \notin r$, there must be another transition $s_3' \xrightarrow{e_2} s_4'$, on the σ leading from s' such that $s_3' \notin r$ and $s_4' \in r$. Again, by the definition of a region, for a similar transition, $s_3 \xrightarrow{e_2} s_4$, labeled with the same event e_2 on the σ leading from s the following condition holds: $s_3 \notin r$ and $s_4 \in r$. However, then, since $s \in r$, there should be another transition on the σ

leading from s, labeled, e.g., with e_3, which exits region r. This transition must occur before $s_3 \xrightarrow{e_2} s_4$, and, therefore, before $s_1 \xrightarrow{e_1} s_2$, which contradicts our assumption that $s_1 \xrightarrow{e_1} s_2$ was the first exit transition for r on σ.

2) $s'' \in r$, i.e., r does not cross σ before s''.

This case is easily reduced to the previous one. Indeed, since $r \notin R_{s'}$ then the boundaries of r must cross σ', a sequence leading from s' to s''. We can then use coregion $\bar{r} = S - r$ (a region complementary to r). Region $\bar{r}$ crosses σ' and must include state s' by the definition of complementary region.

Thus, in both cases, we have reached contradiction, and, therefore, $s = s'$ must be true. □

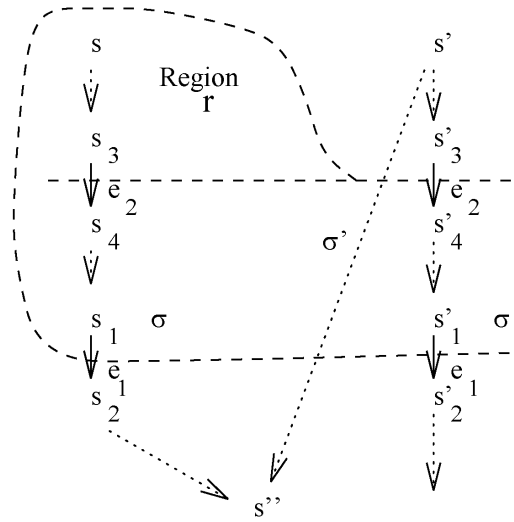


Fig. 18. Illustration to the proof of Property 3.3.

Proof of Theorem 3.4

PROOF. Let $RG(N_I) = (S_I, E_I, T_I, s_I)$. By Theorem 2.1, RG($N_{TS}$) is isomorphic to TS. Therefore, and for the sake of simplicity in the notation, we will assume TS to be the RG of N_{TS} with $S \subseteq 2^{R_{TS}}, T \subseteq 2^{R_{TS}} \times E \times 2^{R_{TS}}$, and $s_{in} \subseteq R_{TS}$. Since N_{TS} is a saturated net, if $r \in e^\circ$, then $\bar{r} \in {}^\circ e$. Hence, for any transition, if some successor place is marked, there is always some predecessor place that is not marked. Therefore, the net is contact-free [42] and, thus, behaves as a PN.

We will also denote by $(\bullet e)_I$ and $(\bullet e)_{TS}$ the set of input places of event e in N_I and N_{TS}, respectively.

We will first prove that:

$$S_I = \{s' \mid \exists s \in S \wedge s' = s - I'\}$$

$$E_I = E$$

$$T_I = \{(s_1', e, s_2') \mid \exists (s_1, e, s_2) \in T \wedge s_1' = s_1 - I' \wedge s_2' = s_2 - I'\}.$$

The proof for the definition of T_I immediately implies those for S_I and E_I. Thus, we will prove that

$$(s_1, e, s_2) \in T \Leftrightarrow (s_1', e, s_2') \in T_I \wedge s_1' = s_1 - I' \wedge s_2' = s_2 - I'.$$

$\Rightarrow$ By induction on the length, n, of the shortest sequence that leads to s_1 from s_{in}. Initially, we have that $s_{in} \in S$ and $s_I \in S_I$. Therefore, it holds for $n = 0$. Assume that there is a sequence of events $s_{in} \xrightarrow{\sigma} s_1$ of length n. By the induction hypothesis $s_I \xrightarrow{\sigma} s_1'$. Since $(\bullet e)_I = (\bullet e)_{TS} - I'$ and $s_1' = s_1 - I'$, we have $(\bullet e)_{TS} \subseteq s_1 \Rightarrow (\bullet e)_I \subseteq s_1'$ and, therefore, $s_1 \xrightarrow{e} s_2 \Rightarrow s_1' \xrightarrow{e}$. According to the firing rules of a PN and $s_2' = s_2 - I'$, it follows that $s_1 \xrightarrow{e} s_2 \Rightarrow s_1' \xrightarrow{e} s_2'$. Therefore, the hypothesis also holds for $n + 1$.

$\Leftarrow$ By induction on the length n of the shortest sequence that leads to s_1'. As above, it holds for $n = 0$. Assume that there is a sequence $s_I \xrightarrow{\sigma} s_1'$ and $s_{in} \xrightarrow{\sigma} s_1$ of length n. If $s_1' \xrightarrow{e}$, then we have that $s_1' = s_1 - I'$ and $({}^\circ e)_I \subseteq s_1'$. But, s_1' is also a set of preregions of e in TS such that $({}^\circ e)_{TS} - I' \subseteq s_1'$. Since

$$\bigcap_{r \in {}^\circ e - I'} = GER(e)$$

$s_1 \xrightarrow{e}$ in TS and, therefore, it follows that e is enabled in s_1 in $N(TS)$. Again, according to the firing rules of a PN and the fact that $s_2' = s_2 - I'$, $s_1' \xrightarrow{e} s_2' \Rightarrow s_1 \xrightarrow{e} s_2$ and the hypothesis also holds for $n + 1$.

Now, the bisimilarity between TS and RG(N_I) can be proved by defining a bisimulation R between S and S_I as follows:

$$sRs' \overset{def}{\Leftrightarrow} s' = s - I'.$$

The conditions for R to be a bisimulation can be trivially proved and are left for the reader. □

ACKNOWLEDGMENTS AND SYNTHESIS TOOL

We are grateful to Marta and Maciej Koutny, who directed us towards the existing literature about regions. The theory in this paper has been implemented in **petrify**, a tool for the synthesis of Petri Nets and asynchronous circuits (available at http://www.lsi.upc.es/~jordic/petrify). This work has been supported in part by Acid-WG (ESPRIT 21949) and CICYT (grant TIC95-0419), by EPSRC (visiting fellowship grants GR/J72486 and GR/J78334, and research grant GR/J52327), and MURST (project "VLSI Architectures").

REFERENCES

[1] R. Alur and D. Dill, "Automata for Modeling Real-Time Systems," *Automata, Languages, and Programming: 17th Ann. Colloquium, Lecture Notes in Computer Science*, vol. 443, pp. 322-335, Warwick Univ., 16-20 July 1990.

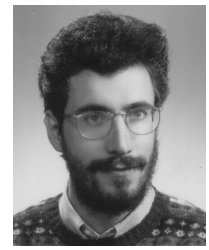
[2] A. Arnold, *Finite Transition Systems*. Prentice Hall, 1994.

[3] E. Badouel, L. Bernardinello, and P. Darondeau, "Polynomial Algorithms for the Synthesis of Bounded Nets," *Lecture Notes in Computer Science*, vol. 915, pp. 364-383, 1995.
[4] E. Badouel and P. Darondeau, "Theory of Regions," *Third Advance Course on Petri Nets*. Springer-Verlag, 1998.
[5] L. Bernardinello, "Synthesis of Net Systems," *Application and Theory of Petri Nets, Lecture Notes in Computer Science*, vol. 691, pp. 89-105. Springer-Verlag, 1993
[6] L. Bernardinello, G. De Michelis, K. Petruni, and S. Bigna, "On Synchronic Structure of Transition Systems," *Proc. Int'l Workshop Structures in Concurrency Theory (STRICT)*, pp. 69-84, May 1995.
[7] G. Berthelot, "Transformations and Decompositions of nets," *Advances in Petri Nets '86*, W. Reisig, W. Brauer, and G. Rozenberg, eds., *Lecture Notes in Computer Science*, vol. 254, pp. 359-376. Springer-Verlag, Feb. 1987.
[8] K. Bilinski and E. Dagless, "High Level Synthesis of Synchronous Parallel Controllers," *Proc. 17th Int'l Conf. Applications and Theory of Petri Nets, Lecture Notes in Computer Science*, vol. 1,091, pp. 346-365, Osaka, Japan, June 1996.
[9] R. Bryaton et al., *Logic Minimisation Algorithms for VLSI Synthesis*. Hingham, Mass: Kluwer Academic, 1984.
[10] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293-318, Sept. 1992.
[11] F. Di Cesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat, *Practice of Petri Nets in Manufacturing*. Chapman & Hall, 1993.
[12] T.-A. Chu, "Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications," PhD thesis, Massachusetts Inst. of Technology, June 1987.
[13] E.M. Clarke, D.E. Long, and K.L. McMillan, "A Language for Compositional Specification and Verification of Finite State Hardware Controllers," *Proc. IEEE*, vol. 79, no. 9, Sept. 1991.
[14] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Complete State Encoding Based on the Theory of Regions," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems*, pp. 36-47, Mar. 1996.
[15] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Methodology and Tools for State Encoding in Asynchronous Circuit Synthesis," *Proc. Design Automation Conf.*, pp. 63-66, June 1996.
[16] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE Trans. Information and Systems*, vol. E80-D, no. 3, pp. 315-325, Mar. 1997.
[17] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri Nets from Finite Transition Systems," Technical Report UPC-DAC-1996-19, Dept. of Computer Architecture, Universitat Politècnica de Catalunya, June 1996. ftp://ftp.ac.upc.es/pub/reports/DAC/1996/UPC-DAC-1996-19.ps.Z.
[18] O. Coudert, C. Berthet, and J.C. Madre, "Verification of Sequential Machines Using Boolean Functional Vectors," *Proc. IFIP Int'l Workshop Applied Formal Methods for Correct VSLI Design*, L. Claesen, ed., pp. 111-128, Leuven, Belgium, Nov. 1989.
[19] G. de Jong and B. Lin, "A Communicating Petri Net Model for the Design of Concurrent Asynchronous Modules," *Proc. Design Automation Conf.*, pp. 49-55, Apr. 1994.
[20] J. Desel and J. Esparza, *Free-Choice Petri Nets, Cambridge Tracts in Theoretical Computer Science*, vol. 40. Cambridge Univ. Press, 1995.
[21] J. Desel and W. Reisig, "The Synthesis Problem of Petri Nets," *Acta Informatica*, vol. 33, no. 4, pp. 297-315, 1996.
[22] D.L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. Cambridge, Mass.: MIT Press, 1988.
[23] D. Drusinsky, "Extended State Diagrams and Reactive Systems," *Dr. Dobb's J.*, pp. 72-80, 106-107, Oct. 1994.
[24] A. Ehrenfeucht and G. Rozenberg, "Partial (Set) 2-Structures, Parts I II," *Acta Informatica*, vol. 27, pp. 315-368, 1990.
[25] J. Esparza and M. Nielsen, "Decidability Issues for Petri Nets," *Petri Nets Newsletter*, vol. 94, pp. 5-23, 1994.
[26] G.D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic, 1996.
[27] M. Hack, "Analysis of Production Schemata by Petri Nets," Technical Report TR 94, Project MAC, Massachusetts Inst. of Technology, 1972.
[28] C.A.R. Hoare, "Communicating Sequential Processes," *Comm. ACM*, pp. 666-677, Aug. 1978.
[29] H. Hulgaard and S.M. Burns, "Bounded Delay Timing Analysis of a Class of CSP Programs with Choice," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems*, pp. 2-11, Nov. 1994.
[30] R.M. Keller, "A Fundamental Theorem of Asynchronous Parallel Computation," *Lecture Notes in Computer Science*, vol. 24, pp. 103-112, 1975.
[31] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Synthesis of General Petri Nets," Technical Report 57, IEICE, Japan, May 1996.
[32] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. London: John Wiley and Sons, 1993.
[33] R.P. Kurshan, "Analysis of Discrete Event Coordination," *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
[34] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Kluwer Academic, 1993.
[35] C.Y. Lee, "Representation of Switching Functions by Vinary Decision Programs," *Bell System Technical J.*, vol. 38, pp. 985-999, 1959.
[36] B. Lin and F. Somenzi, "Minimization of Symbolic Relations," *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp. 88-91, Santa Clara, Calif., Nov. 1990.
[37] R. Milner, "A Calculus of Communication Systems," *Lecture Notes in Computer Science*, vol. 92. Springer-Verlag, 1980.
[38] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
[39] D. Misunas, "Petri Nets and Speed-Independent Design," *Comm. ACM*, pp. 474-481, Aug. 1973.
[40] M. Mukund, "Petri Nets and Step Transition Systems," *Int'l J. Foundations of Computer Science*, vol. 3, no. 4, pp. 443-478, 1992.
[41] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541-580, Apr. 1989.
[42] M. Nielsen, G. Rozenberg, and P.S. Thiagarajan, "Elementary Transition Systems," *Theoretical Computer Science*, vol. 96, pp. 3-33, 1992.
[43] S.M. Nowick and D.L. Dill, "Automatic Synthesis of Locally-Clocked Asynchronous State Machines," *Proc. Int'l Conf. Computer-Aided Design*, Nov. 1991.
[44] J. Oldfield and R. Dorf, *Field-Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*. John Wiley and Sons, 1995.
[45] E. Pastor and J. Cortadella, "Polynomial Algorithms for the Synthesis of Hazard-Free Circuits from Signal Transition Graphs," *Proc. Int'l Conf. Computer-Aided Design*, Nov. 1993.
[46] E. Pastor, O. Roig, J. Cortadella, and R. Badia, "Petri Net Analysis Using Boolean Manipulation," *Proc. 15th Int'l Conf. Application and Theory of Petri Nets*, Zaragoza, Spain, June 1994.
[47] M. Peña and J. Cortadella, "Combining Process Algebras and Petri Nets for the Specification and Synthesis of Asynchronous Circuits," *Proc. Int'l Symp. Advanced Research in Asynchronous Circuits and Systems*, pp. 222-232, Mar. 1996.
[48] C.A. Petri, "Kommunidation mit Automaten," PhD thesis, Technical Report Schriften des IIM Nr. 3, Institut für Instrumentalle Mathematik, Bonn, Germany, 1962.
[49] S.R. Petrick, "A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants," Technical Report AFCRC-TR-56-110, Air Force Cambridge Research Center, Cambridge, Mass., Apr. 1956.
[50] M. Pezzé, R.N. Taylor, and M. Young, "Graph Models for Reachability Analysis of Concurrent Programs," *ACM Trans. Software Eng. and Methodology*, vol. 4, no. 2, pp. 171-213, 1995.
[51] I. Reicher and M. Yoeli, "Net-Based Modeling of Communicating Parallel Processes with Applications to VLSI Design," Technical Report 532, Technion, Haifa, Israel, 1988.
[52] T.G. Rokicki, "Representing and Modeling Digital Circuits," PhD thesis, Stanford Univ., 1993.
[53] L.Y. Rosenblum and A.V. Yakovlev, "Signal Graphs: From Self-Timed to Timed Ones," *Proc. Int'l Workshop Timed Petri Nets*, Torino, Italy, 1985.
[54] M. Silva, *Las Redes de Petri en la Automática y la Informática*. Madrid, Spain: AC, 1985. (in Spanish)
[55] D.C. Tsichritzis and P.A. Bernstein, *Operating Systems*. London: Academic Press, 1974.
[56] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming, Int'l Series Parallel Computation*, vol. 5. Cambridge Univ. Press, 1993.
[57] G. Winskel, "Petri Nets, Algebras, Morphisms, and Compositionality," *Information and Computation*, vol. 7, pp. 197-238, 1987.

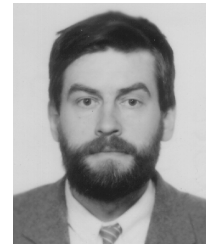
[58] M. Zhou, F. DiCesare, and A. Desrochers, "A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems," *IEEE Trans. Robotics and Automation*, vol. 8, no. 3, pp. 350-361, June 1992.

Jordi Cortadella received the MS and PhD degress in computer science from the Technical University of Catalonia, Barcelona, Spain, in 1985 and 1987, respectively. He is an associate professor in the Department of Software at the Technical University of Catalonia. In 1988, he was a visiting scholar at the University of California, Berkeley. His research interests include computer-aided design of VLSI systems, with special emphasis on synthesis and verification of asynchronous circuits, concurrent systems, computer arithmetic, and parallel architectures. He has coauthored more than 80 research papers in technical journals and conferences. He served on the technical committees of several international conferences in the field of design automation and concurrent systems.

Michael Kishinevsky received the MSc and PhD degrees in computer science from the Electrotechnical University of St. Petersburg, Russia. He was a researcher at the St. Petersburg Mathematical Economics Institute Computer Department, Russian Academy of Science in 1979-1982 and 1987-1989. From 1982 to 1987, he has been with a software company. From 1988 to 1992, he was a senior researcher at the R&D Coop TRASSA. In 1992, he joined the Department of Computer Science, Technical University of Denmark, as a visiting associate professor. From the end of 1994 through 1998, he is a professor at the University of Aizu, Japan. In 1998, he joined the Strategic CAD Lab Intel Corporation, Hillsboro, Oregon. His current research interests include design of asynchronous and reactive systems and theory of concurrency. He coauthored two books on asynchronous design and has published more than 50 journal and conference papers.



Luciano Lavagno graduated magna cum laude in electrical engineering from Politecnico di Torino (Italy) in 1983. From 1984 to 1988, he was with CSELT Laboratories (Torino, Italy), where he was involved in an ESPRIT project that developed a complete high-level synthesis system. In 1988, he joined the Department of Electrical Engineering and Computer Science of the University of California at Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. In 1992, he received his PhD in electrical engineering and computer science from the University of California at Berkeley. Dr. Lavagno is the author of a book on asynchronous circuit design, the coauthor of a book on hardware/software co-design of embedded systems, and has published more than 60 journal and conference papers. In 1991, he received the Best Paper award at the 28th Design Automation Conference in San Francisco. He served on the technical committees of several international conferences in his field (namely the Design Automation Conference, the International Conference on Computer Aided Design, the European Design Automation Conference). He has also been a consultant for various EDA companies, such as Synopsys and Cadence. He is currently an assistant professor at the Politecnico di Torino, Italy, and a research scientist at Cadence Berkeley Laboratories. His research interests include the synthesis of asynchronous and low-power circuits, the concurrent design of mixed hardware and software systems, and the formal verification of digital systems.



Alexandre Yakovlev holds his MSc and PhD degrees in computing science from Electrotechnical University of St. Petersburg, Russia, where he has worked in the area of asynchronous and concurrent systems since 1980, and, in the period between 1982 and 1990, held the positions of assistant and associate professor in the Computing Science department. He first visited Newcastle in 1984-1985 for research in VLSI and design automation. After returning to Britain in 1990, he worked for one year at the Politechnic of Wales (now University of Glamorgan). Since 1991, he has been a lecturer and, quite recently, a reader in computing systems design in the Department of Computing Science, University of Newcastle upon Tyne, where he is heading the VLSI design research group. His current research interests and publications are in the field of modeling and design of asynchronous, concurrent, real-time, and dependable systems.