Synthesizing Petri Nets from State-Based Models

Jordi Cortadella[*]
Univ. Politècnica de Catalunya, 08071 Barcelona, Spain

Michael Kishinevsky[†]
The University of Aizu Aizu-Wakamatsu, 965-80 Japan

Luciano Lavagno[‡]
Politecnico di Torino 10129 Torino, Italy

Alex Yakovlev[§]
Univ. of Newcastle upon Tyne, NE1 7RU England

Abstract

This paper presents a method to synthesize labeled Petri nets from state-based models. Although state-based models (such as Finite State Machines) are a powerful formalism to describe the behavior of sequential systems, they cannot explicitly express the notions of concurrency, causality *and* conflict*. Petri nets can naturally capture these notions. The proposed method in based on deriving an* Elementary Transition System *(*ETS*) from a specification model. 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 ETS.*

This paper presents the first known approach to obtain an ETS *from a non-elementary* TS *and derive a place-irredundant Petri net. Furthermore, by imposing constraints on the synthesis method, different classes of Petri nets can be derived from the same reachability graph (pure, free choice, unique choice). This method has been implemented and efficiently applied in different frameworks: Petri net composition, synthesis of Petri nets from asynchronous circuits, and resynthesis of Petri nets.*

1 Introduction

In this paper we present a method which given a finite state model, called Transition System (TS), synthesizes a safe Petri Net with a reachability graph that is either isomorphic to the original TS or isomorphic to a minimized version of the original TS. The synthesized PN is always place-irredundant, i.e., it is not possible to remove any place from the net without violating its behavior.

The synthesis method provides us with a technique for transforming specifications. Given a model which can be mapped into a TS, we can derive a PN which is equivalent 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. Also, we can use this tool for transformation of Petri Nets aimed 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 (safe, Free-Choice, Unique-Choice, etc.) This opens up an avenue for building interactive tools where a designer has 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.

A basic intermediate object between a TS and a PN is a *region* [11, 1, 3, 10]. "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. A region in a Transition System is a set of states, such that transitions in and out of it "mimic" the PN firing behavior which un-marks predecessor places and marks successor places of a transition. In this way, it is possible to identify regions with places, and construct a PN which has exactly the same set of labeled firing sequences as the TS.

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

- Their contribution was mainly theoretical, aimed at obtaining a *canonical* representation of the PN, with 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 more understandable by the designer.

- They did not address the problem of *merging* and *splitting* "equivalent" labels, i.e., those labels which model the same event, but must be split in order to yield a valid Petri Net.

- They were limited to *elementary* TSs, which are quite restricted, while we can handle the full class of TSs by means of label splitting.

In this paper, we present an algorithm for generating a complete set of *minimal* regions (which are analogues 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 among them (an exact minimization of places in PNs), or heuristically choose a minimal place-irredundant net, if searching for a minimum one is too time consuming.

The paper is organized as follows. Sections 2 and 3 formally introduce Transition Systems, Petri nets and regions. Section 4 describes the synthesis algorithms in detail, and briefly outlines extensions of the basic method to cope with a broader class of specifications. Section 5 shows the experimental results obtained by a practical application of the proposed methodology. Section 6 concludes the paper.

[*] This work has been partly supported by the Ministry of Education of Spain (CICYT TIC 95-0419).

[†] This work has been partly supported by the U.K. SERC GR/J78334.

[‡] This work has been partly supported by the U.K. SERC GR/J72486 and by MURST research project "VLSI architectures".

[§] This work has been partly supported by the U.K. SERC GR/J52327.

2 Models

2.1 Transition systems

A *transition system* (TS) is a quadruple [11] $TS = (S, E, T, s_{in})$, where S is a *finite non-empty* set of *states*, E is a set of *events*, $T \subset S \times E \times S$ 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').

The *reachability relation* between states is the transitive closure of the transition relation T. If there is a (possibly empty) sequence of transitions σ between states s and s', then this is denoted by $s \xrightarrow{\sigma} s'$ or simply by $s \xrightarrow{*} s'$. We also write $s \xrightarrow{e}$, $\xrightarrow{e} s'$, and $s \xrightarrow{\sigma}$ if $s \xrightarrow{e} s'$ or $s \xrightarrow{\sigma} s'$, correspondingly. Note that each state is reachable from itself. Furthermore, a TS must satisfy the following four basic axioms:

A1. No self-loops: $\forall (s, e, s') \in T : s \neq s'$;

A2. No multiple arcs between a pair of states: $\forall (s, e_1, s_1), (s, e_2, s_2) \in T : [s_1 = s_2 \Rightarrow e_1 = e_2]$

A3. Every event has an occurrence: $\forall e \in E : \exists (s, e, s') \in T$;

A4. Every state is reachable from the initial state: $\forall s \in S : s_{in} \xrightarrow{*} s$.

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 *non-deterministic*.

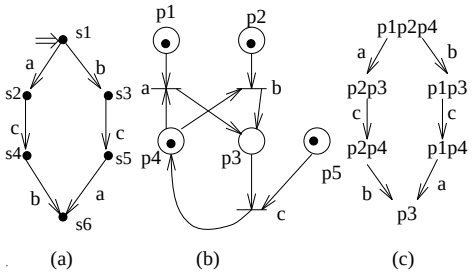


Figure 1: An example of Transition System (a), the corresponding PN (b), its labeled RG (c)

A TS can be represented as an arc-labeled directed graph. A simple example of a TS without cycles is shown in Figure 1,a.

2.2 Petri Nets

A Petri Net [12] 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 marking is a function that assigns every place a non-negative number of tokens. 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 ($m_1 \xrightarrow{t} m_2$). A PN expressing the same behavior as the TS from Figure 1,a is shown in Figure 1,b.

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 reachable in N from the initial marking m_0 is called its Reachability Set. The graph with vertices corresponding to markings of a PN and with an arc (m_1, m_2) in the graph if and only if $m_1 \rightarrow m_2$ is called its Reachability Graph (RG).

A *labeled* PN is a PN with a labeling function $\lambda : T \longrightarrow A$ 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 case we can use the label as the name of the transition. In the RG of a labeled PN, an arc between markings m_1 and m_2 is labeled by the label $\lambda(t)$ of the transition t, which fires between markings m_1 and m_2. Such RG is called a *labeled* RG.

One can easily check that the labeled RG Figure 1,c derived for the PN from Figure 1,b is isomorphic to the TS (Figure 1,a).

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 [7] 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., $\forall t_1, t_2$ $\bullet t_1 \neq \bullet t_1$ or $t_1\bullet \neq t_2\bullet$).

2.3 Equivalence

This paper presents an algorithm for transforming TSs into PNs. The notion of equivalence in such transformation is based on isomorphism of labeled graphs and variations of such isomorphism.

Two TSs $TS_1 = (S_1, E_1, T_1, s_{in_1})$ and $TS_2 = (S_2, E_2, T_2, s_{in_2})$ are *isomorphic* if there exist two bijections $f_S : S_1 \longrightarrow S_2$ and $f_E : E_1 \longrightarrow E_2$ such that $s_{in_2} = f_S(s_{in_1})$ and $(s, e, s') \in T_1$ if and only if $(f_S(s), f_E(e), f_S(s')) \in T_2$. A RG of a labeled PN can be always interpreted as a TS and therefore the equivalence of a TS and the corresponding PN can be viewed as an isomorphism of a TS and a RG of the corresponding PN.

Often we will consider isomorphism of a TS and a minimized version of another TS. For comparing two TSs with different event counts a split-isomorphism is used. Two TSs TS_1 and TS_2 are *split-isomorphic* if there is another TS TS such that:

1. the underlying graphs of these three TSs are isomorphic, and
2. labels on arcs in TS_1 and TS_2 can be viewed as two different enumerations of events from TS.

This enumeration corresponds to assigning instance numbers to the events. For example, if in TS there are three arcs labeled with the same event a, then in TS_1 one of these arcs can be labeled with a_1, and two others with a_2, while in TS_2 all three arcs can be assigned different instance numbers: $a_1, a_2, a_3, \ldots$. This procedure of assigning instance numbers to different occurrences of labels in TS is called *splitting*.

All three notions of equivalence (isomorphism of TSs, isomorphism with a minimized TS, and split-isomorphism) guarantee that two equivalent TSs are *bi-similar*, which is a stronger condition than language equivalence in general ([9]). This implies, for example, that deadlock and liveness properties are preserved for a PN generated from the TS.

3 Regions

Let S_1 be a subset of the states of a TS, $S_1 \subseteq S$. If $s \notin S_1$ and $s' \in S_1$, then we say that transition $s \xrightarrow{a} s'$ *enters* S_1. If $s \in S_1$ and $s' \notin S_1$, then transition $s \xrightarrow{a} s'$ *exits* S_1. Otherwise, transition $s \xrightarrow{a} s'$ *does not cross* S_1. In particular, if $s \in S_1$ and $s' \in S_1$, then the transition is said to be *internal* to S_1, and if $s \notin S_1$ and $s' \notin S_1$, then the transition is *external* to S_1.

A subset of states, r, is a *region* if for each event e one of the following conditions hold: all transitions labelled with e (1) exit r, (2) enter r, or (3) do not cross r.

Let us consider the TS shown in Figure 1. 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 non-trivial regions.

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 iff r' is not a subregion of any other region of the TS.

3.1 Properties of regions

The following propositions state a few important properties of regions.

Property 3.1 *If r and r' are two different regions such that r' is a subregion of r, then $r - r'$ is a region.*

Property 3.2 *A set of states r is a region, if and only if its coset $\overline{r} = S - r$ is a region, where S is a set of all states of the* TS.

Property 3.3 *Every region can be represented as a union of disjoint minimal regions.*

Property 3.1 has been mentioned in [3]. Property 3.2 was given in [11]. Property 3.3 is a stronger refinement of the corresponding property from [3], which shows that any region can be viewed as a linear combination of the minimal regions. The proofs are given in [4].

For each state $s \in S$ we define the set of non-trivial regions containing s, denoted by R_s.

A region r is a *pre-region* of event e if there is a transition labeled with e which exits r. A region r is a *post-region* of event e if there is a transition labeled with e which enters r. The set of all pre-regions and post-regions 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 enter r. Similarly, if $r \in e^\circ$, then all transitions labeled with e exit r.

There are eight non-trivial regions in the TS from Figure 1: $r_1 = \{s_1, s_3, s_5\}$; $r_2 = \{s_1, s_2, s_4\}$; $r_3 = \{s_2, s_3, s_6\}$; $r_4 = \{s_1, s_4, s_5\}$; $r_5 = \{s_1, s_2, s_3\}$; $r_6 = \{s_4, s_5, s_6\}$; $r_7 = \{s_2, s_4, s_6\}$; $r_8 = \{s_3, s_5, s_6\}$. All of these regions are minimal. Pre-regions and post-regions are defined as follows: ${}^\circ a = \{r_1, r_4\}$; ${}^\circ b = \{r_2, r_4\}$; ${}^\circ c = \{r_3, r_5\}$; $a^\circ = \{r_3, r_7\}$; $b^\circ = \{r_3, r_8\}$; $c^\circ = \{r_4, r_6\}$.

3.2 Excitation and switching regions

While regions in a TS are related to places in the corresponding PN, an excitation region [8] 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.

A set of states S_1 is called a *generalized excitation region* (an *excitation region*) for event a, denoted by $GER(a)$ (by $ER_j(a)$), if it is a *maximal* (a *maximal connected*) set of states such that for every state $s \in S_1$ there is a transition $s \xrightarrow{a}$. The GER for a is the union of all ERs for a. In the TS from Figure 1,a there are two excitation regions for event a: $ER_1(a) = \{s_1\}$ and $ER_2(a) = \{s_5\}$. The corresponding GER for event a is $GER(a) = \{s_1, s_5\}$.

Similarly to excitation regions, we define generalized switching regions, $GSR(a)$, and switching regions, $SR_j(a)$, as sets of states reached *immediately after* the occurrence of an event. In the TS from Figure 1,a there are two switching regions for event a: $SR_1(a) = \{s_2\}$ and $SR_2(a) = \{s_6\}$. The corresponding GSR for event a is $GSR(a) = \{s_2, s_6\}$.

3.3 Elementary Transition Systems

3.3.1 Axioms for ETS

A TS $TS = (S, E, T, s_{in})$ is called *elementary* [11] (ETS) if it satisfies, in addition to (A1) – (A4), the following two axioms about regions:

A5. State separation property:
$\forall s, s' \in S : [R_s = R_{s'} \Rightarrow s = s']$;

A6. Forward closure property:
$\forall s \in S \forall e \in E : [{}^\circ e \subseteq R_s \Rightarrow s \xrightarrow{e}]$

Axiom 5 implies that two different states must belong to different sets of regions. Axiom 6 implies that if state s is included in all pre-regions of event e, then e must be enabled in s. It is easy to see that the TS shown in Figure 1 is elementary. The TS shown in Figure 2,a is a cyclic elementary TS, while Figure 2,b shows a non-elementary TS. The forward closure property is violated for events a and b. Let us consider event a. The only pre-region of a is region $\{s_1, s_3, s_5, s_7\}$. Therefore ${}^\circ e \subseteq R_{s_7}$, but there is no transition labeled with a from s_7.

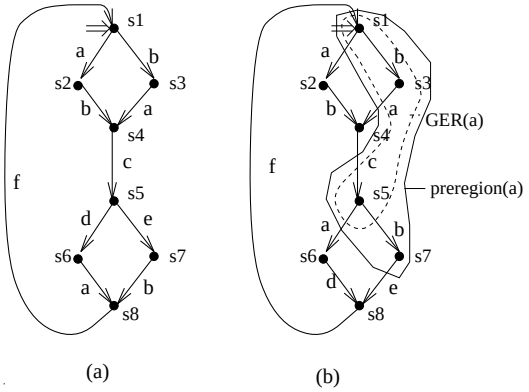


Figure 2: Examples of elementary (a) and non-elementary (b) TSs

It has been shown in [11] that if a TS is elementary, then a PN with a reachability graph isomorphic to the TS can be constructed.

Proposition 3.1 *For each elementary* TS *there exists a safe, pure, and simple* PN *such that:*

1. *each* PN *transition is labeled with an event of the* TS*;*
2. *no two transitions are labeled with the same event;*
3. *the* RG *of the* PN *is isomorphic to the* TS*.*

The procedure given by [11, 3] to synthesize a Petri net from an ETS is as follows:

- For each event a a transition labeled with a is generated in the PN;
- For each (minimal) region r_i a place p_i is generated;
- Place p_i contains a token in the initial marking m_0 iff the corresponding region r_i contains the initial state of the ETS s_{in};
- The flow relation is as follows: $a \in p_i\bullet$ iff r_i is a pre-region of a and $a \in \bullet p_i$ iff r_i is a post-region of a.

A PN which is synthesized from all regions is called a *saturated* net. The net constructed from all minimal regions is called a *minimal saturated net*. These nets are canonical, however, even a minimal saturated net can be redundant. Places can still be removed from it while still preserving the required isomorphism 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.*

Another important drawback of the described procedure is that axioms 5 and 6 do not provide an efficient algorithm for checking elementarity, since they require to derive *all the regions* and also to check 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.

3.4 Elementary and minimal Transition Systems

In this section we show the relationship between the elementarity of a TS and its minimality. Let us first recall a few useful definitions.

Two states s and s' of a TS are *equivalent*, $s \cong s'$, if for every sequence of transitions σ: $s \xrightarrow{\sigma}$ if and only if $s' \xrightarrow{\sigma}$. A TS is called *minimal* if it contains no equivalent states. We say that the *confluence condition* holds for states s and s' if there is a state s'' which is reachable both from s and s'. Note that according to the definition of reachability s'' can coincide with s or s'.

The relation between minimality and elementarity of TS is two-fold:

- If an elementary TS is not minimal, then for each pair of equivalent states the confluence condition does not hold.
- If a TS is not minimal and there is a pair of equivalent states for which the confluence condition is satisfied, then the TS is not elementary.

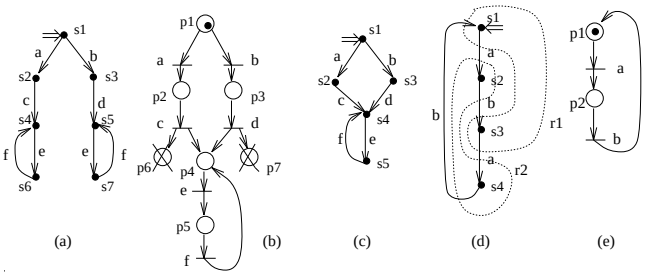


Figure 3: Minimality and elementarity

This relationship is illustrated by Figure 3. The TS in Figure 3,a is elementary. However, it is not minimal since $s_4 \cong s_5$ and $s_6 \cong s_7$. Obviously, the confluence condition for these two pairs of states is not satisfied. A safe PN with an RG isomorphic to this TS is shown in Figure 3,b. Places p_6 and p_7 corresponds to the minimal regions $\{s_4, s_6\}$ and $\{s_5, s_7\}$. These places have no output transitions (the corresponding regions do not serve as pre-regions to any events). Therefore, these places can be removed from the PN without changing its behavior.

The RG of the PN *without* p_6 *and* p_7 is isomorphic to the TS shown in Figure 3,c, which is a minimized version of the initial TS (Figure 3,a).

Another TS example is given by Figure 3,d. This TS is not minimal and also not elementary, since the state separation property A6 is not satisfied for equivalent states $s_1 \cong s_3$ and $s_2 \cong s_4$: $R_{s_1} = R_{s_3} = r_1 = \{s_1, s_3\}$ and $R_{s_2} = R_{s_4} = r_2 = \{s_2, s_4\}$. Note that the confluence conditions for equivalent states is satisfied. The PN from Figure 3,e contains two places which corresponds to regions r_1 and r_2 and two transitions labeled with a and b. Its RG is isomorphic to a minimized version of the initial non-elementary TS. After minimization, the TS from Figure 3 becomes elementary.

As indicated by the two examples in Figure 3 an implicit minimization occurs (1) when regions which do not serve as pre-regions are removed (Figure 3,a-c) or (2) when regions including all equivalent states are generated (Figure 3,d,e).

3.5 Elementarity conditions

In this section we present conditions for elementarity which allow for efficient checking. We first connect the notions of pre-regions and post-regions with those of excitation and switching regions.

Property 3.4

1. *A region r is a pre-region of event a iff $GER(a) \subseteq r$ and $GSR(a) \cap r = \emptyset$.*
2. *A region r is a post-region of event a iff $GSR(a) \subseteq r$ and $GER(a) \cap r = \emptyset$.*

This property allows to construct regions which serve as pre-regions starting with excitation regions. This allows for efficient BDD-based implementation. The following property helps in performing efficient PN synthesis.

Proposition 3.2

1. *If a* TS *is elementary, then the following three conditions are satisfied:*
   (a) *Excitation closure. For each event* a*:* $\bigcap_{r \in {}^\circ a} r = GER(a)$*;*
   (b) *Event effectiveness. For each event* a*:* ${}^\circ a \neq \emptyset$*;*
   (c) *If the* TS *is not minimal, then for each pair of equivalent states the confluence condition does not hold.*
2. *If a* TS *is minimal and the excitation closure and the event effectiveness conditions are satisfied, then the* TS *is elementary.*

The proof is given in [4].

Let us consider again the non-elementary TS from Figure 2,b. This TS is non-elementary, because the excitation closure condition is violated for a and b. For example, $ER(a) = \{s_1, s_3, s_5\}$, while the only pre-region of a, $\{s_1, s_3, s_5, s_7\}$, is a proper superset of $ER(a)$. According to Proposition 3.2, if the generated PN has an RG which is minimal, then only the excitation closure and the event effectiveness conditions must be checked to verify the elementarity of a TS. The event effectiveness condition is trivial to check. The major new result, yielding our new synthesis procedure, is the excitation closure condition.

4 Petri net synthesis

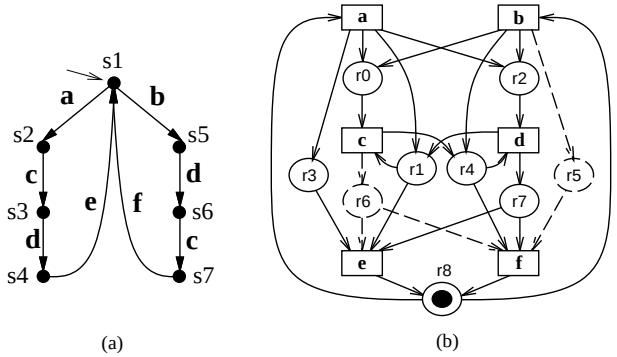
This section describes the algorithms for synthesis. The proof of their correctness is given in [4].

4.1 Synthesis algorithms

The skeleton of the algorithm for synthesis of a PN is given by the following pseudo-code.

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

The input of this algorithm is a TS. The output is a PN which is equivalent to the initial TS. The function `expand_states` recursively generates all minimal regions which serve as pre-regions for one event. These minimal regions are called *minimal pre-regions.* Minimal regions which are not pre-regions are not needed, since a PN with a minimized RG is generated (see section 3.4). 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 minimized version of the initial TS has been found to be non-elementary. This function is discussed in section 4.3. Other means of handling non-elementary TSs, based on using self-loops and dummy transitions, are discussed in section 4.4. The function `map_to_PN` is the final step for constructing a PN from the set of regions, which has been described in section 3.3.



(a) (b)

Figure 4: (a) Transition system. (b) Minimal saturated and place-irredundant nets.

pre-region	events	pre-region	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		

Table 1: All minimal pre-regions of the transition system depicted in Figure 4,a

```
expand_states (r,R)
begin
    /* r is the set of states to be expanded */
    /* R collects all regions generated */
    if r is a region then
        R = R ∪{r};
        return;
        /* since any region expanded from r
        would not be minimal */
    end if;
    find e ∈ E violating some region condition in r;
    r' = r ∪ {1st set of states to legalize e};
    expand_states (r',R);
    /* For some conditions the set of states must be
    expanded in two directions (see [4]) */
    r' = r ∪ {2nd set of states to legalize e};
    expand_states (r',R);
end
```

4.2 Irredundant regions

A set of regions R is called *redundant* if there is a region $r \in R$ such that set of regions $R - r$ still satisfies the excitation closure (and therefore also the event effectiveness) condition. Otherwise R is called *irredundant*. A region is said to be *essential* when it cannot be removed from any set of regions without violating the excitation closure of some event.

We will illustrate how an irredundant set of places can be calculated by means of the example of Figure 4. Table 1 presents all minimal pre-regions of the TS.

As a preliminary step, *essential regions* are calculated. A region r is *essential* if there exists a state $s \in r$ and an event e such that $r \in {}^\circ e$, $s \notin GER(e)$ 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

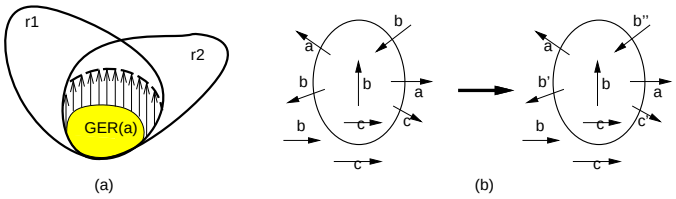


Figure 5: (a) Selection of a set of states to force the excitation closure. (b) Forcing a set of states to be a region by means of label splitting.

intersection of pre-regions a state in which e is not enabled). For example, for event c we have

$$^\circ c = \{r_0, r_1\}; \qquad GER(c) = \{s_2, s_6\} = r_0 \cap r_1$$

In this case, both r_0 and r_1 are essential, since none of them can 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 non-essential regions: r_3, r_5, r_6 and r_7.

Next, for each event with non-essential pre-regions (e and f in the example), all minimal covers are implicitly generated. For event e, we have two minimal covers: $\{r_1, r_6\}$ and $\{r_1, r_3, r_7\}$. For event f we also have two minimal covers: $\{r_4, r_7\}$ and $\{r_4, r_5, r_6\}$. Finding a minimum cost cover can be posed as finding a minimum cost solution of a Boolean equation describing the covering conditions (unate covering problem). To reduce the complexity of the problem, essential regions are not included. 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, 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 Figure 4,b, where the redundant places are depicted by dotted lines). There is another possible solution ($\{r_6, r_7\}$), but it has non-minimum cost.

4.3 Label splitting

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

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

some events must be split to make the TS elementary. This situation is illustrated in Figure 5,a. Event a has two minimal pre-regions (r_1 and r_2) and their intersection is larger than $GER(a)$.

The strategy to split events is as follows. During the expansion of $GER(a)$ towards the pre-regions of a, several sets of states are explored. We focus our attention on those sets S such that

$$GER(a) \subseteq S \subset \bigcap_{r \in {}^\circ a} r$$

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. This is done by splitting the labels of those events that do not fulfill the region conditions. An example is depicted in Figure 5,b.

The strategy for splitting ensures that, with the new labeling, there will be one pre-region *smaller* than the intersection of the pre-regions with the former labeling. Next, pre-regions are recomputed, and excitation closure is verified. If the closure test fails, the procedure is executed again. This strategy converges monotonically, and in the worse case splits all the labels, so that each state of the TS trivially becomes a region. For practical examples, only one or two iterations are usually required to converge. Obviously, the PN obtained after any splitting is split-isomorphic to the initial TS.

4.4 Modifications of the basic synthesis method

The class of TSs and PNs that can be handled by our synthesis procedure can be extended to include some non-elementary TSs without using the label splitting technique. One powerful extension, to *non-pure nets*, is concerned with allowing *self-loop regions* to be involved in the excitation closure condition. A region r is a self-loop region for an event e if it is not a pre-region and the $GER(a)$ is contained in r. Including a place corresponding to a self-loop region into the set of input and output places does not restrict the enabling conditions for an event unnecessarily. Yet it allows to "trim" the intersection of pre-regions so that the given event is not enabled in the states not included in its excitation region. Non-pure nets appear to be very useful in practice, e.g. modelling arbitration circuits in which one event asymmetrically disables another event. More details on this extension and others, such as inserting *dummy events*[1] and relaxing axiom A2 (PN simplicity) can be found in [4].

Our basic synthesis method (namely, excitation closure) can be customized to satisfy the requirements of some classes of Petri nets which can be useful in practice. In particular, it can produce *free-choice and unique-choice* PNs, a class of nets allowing efficient algorithms for their verification and circuit implementation. More on that is also in [4].

5 Applications

Table 2 describes the results of the application of our algorithms to the synthesis of Petri nets from transition systems obtained from speed-independent circuits. 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). All CPU times are in seconds on a SUN SPARC 10 workstation. Z, S, P, T, F and M are the numbers of signals, states, places, transitions, arcs and markings, respectively.

[1] I.e., events whose firing produces no visible effect on the trace and bi-simulation semantics [9].

circuit	Z	S	P	T	F	M	CPU
unsafe	5	22	15	12	34	22	2.8
a4_tflo1	8	20	18	16	40	20	3.3
a_10_dr2	50	9408	128	100	336	9408	2923.6
a_11_sen	19	85	44	38	93	85	31.2
dags55	19	130	43	38	342	130	121.9

Table 2: Synthesis of Petri nets from speed-independent circuits.

example	initial PN	final PN	CPU
	P / T / F / M	P / T / F / M	
alloc-outbound	17 / 18 / 36 / 17	15 / 14 / 37 / 16	1.5
clock	10 / 10 / 20 / 10	9 / 7 / 27 / 10	0.7
dff	20 / 20 / 44 / 20	13 / 11 / 44 / 20	3.7
espinalt	27 / 25 / 57 / 27	23 / 20 / 52 / 27	5.2
fair_arb	13 / 20 / 40 / 13	11 / 10 / 33 / 13	2.0
future	31 / 28 / 62 / 36	18 / 16 / 38 / 36	6.9
gcd-ra	66 / 58 / 136 / 3240	48 / 42 / 114 / 3190	109.0
intel_div3	8 / 8 / 16 / 8	7 / 5 / 20 / 8	0.3
intel_edge	28 / 36 / 72 / 28	15 / 16 / 132 / 26	16.2
isend	56 / 44 / 116 / 53	24 / 19 / 105 / 36	41.4
lin_edac93	14 / 12 / 28 / 20	10 / 8 / 22 / 20	1.1
master-read	36 / 26 / 72 / 8932	33 / 26 / 66 / 8932	29.6
pe-rcv-ifc	43 / 38 / 96 / 46	26 / 21 / 118 / 37	33.9
pulse	12 / 12 / 24 / 12	7 / 6 / 20 / 12	0.6
rcv-setup	14 / 15 / 32 / 14	11 / 12 / 26 / 11	1.2
vme_read	41 / 32 / 84 / 255	38 / 27 / 114 / 251	62.1
vme_write	49 / 36 / 100 / 821	44 / 32 / 139 / 817	135.8

Table 3: Petri net minimization.

Table 3 describes the results of the application of our algorithms to the minimization of a given labeled Petri net. The same notation is used for places, transitions, arcs and markings[2].

The implemented tool, called `petrify`, is available at http://www.ac.upc.es/~vlsi/petrify/petrify.html.

5.1 Counterflow pipeline processor: stage control circuit

As an application example, we used our method to derive a PN specification and its circuit implementation for a stage control circuit in a counterflow pipeline processor (for a complete description of the architecture refer to [13]). Although some of the steps were assisted by hand, our net synthesis approach played a key methodological role. These steps are shown in Figure 6. The original behavioral description of the stage control was taken from C.E. Molnar in the form of a TS.

The two transformations performed at this level were (1) reducing the model to asymmetric form, which would then allow to (2) insert a (single) dummy transition (denoted by ϵ). The legitimacy of this reduction was verified by means of TS composition. The latter proved that the reduced model preserved all the main functionalities: the counterflow pipe remained deadlock-free and guaranteed the propagation of instructions and results without missing their synchronisation in those stages where they may meet. The resulting TS is *quasi-elementary* in the sense that it satisfies our extended conditions of elementarity. The subsequent synthesis of the PN was performed using the techniques described in this paper, including operating with self-loop regions. Region $r2$ is a minimal region, which is a self-loop region for event PR. On the other hand, the other self-loop region $r6$ (for event AR) is not minimal; it is a union of minimal regions $r2$ and $r4$.

[2] the number of markings may differ when different markings of the original PN correspond to equivalent states in the RG.

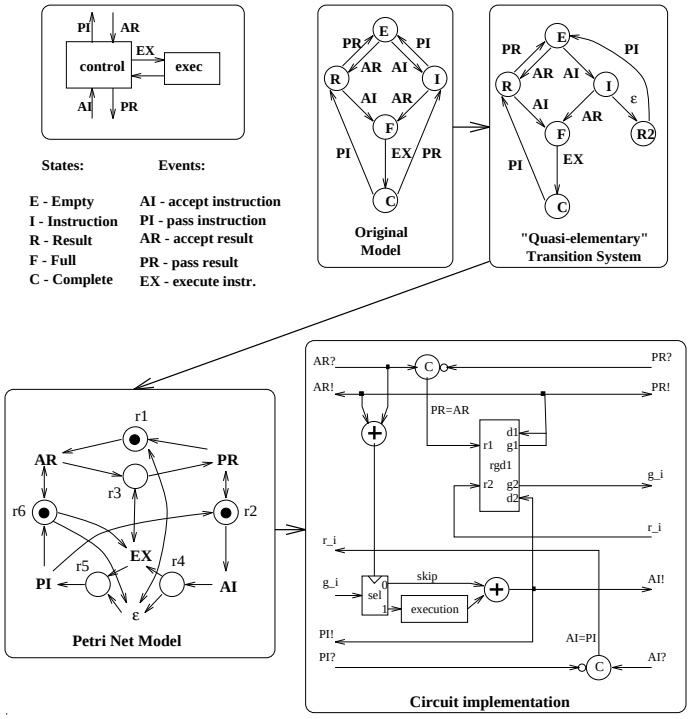


Figure 6: Counterflow pipeline example: from transition system to circuit

This example illustrates that the amount of search for a self-loop region can be significant, hence the complexity of synthesizing non-pure PNs can be much greater than for pure nets. In the net model of the stage control circuit all events are uniquely represented. It was then relatively straightforward to apply the technique of [5], which systematically adds semaphores for conflict places at the PN level. Finally, implementing semaphores by mutual exclusion elements we can obtain a circuit implementation (somewhat similar to a solution recently presented by Ebergen [6]) shown in Figure 6.

5.2 Translation of high-level specifications

This section presents an example that illustrates how this method can be useful for PN composition and translation from high-level languages into Petri nets. The example is aimed at modeling circuits obtained from TANGRAM descriptions (a CSP-like language [2] for asynchronous circuit compilation).

Figure 7 depicts how the STG of the composition of two handshake circuits (a sequencer and a parallelizer) is obtained. From the STG of each handshake circuit, synchronization places between events with the same labels are inserted (dashed places and arcs). Each synchronization place receives arcs from the STG modeling the circuit that "produces" the event (output channel) and generates arcs towards the corresponding events at the input channel. After composing several STGs, the internal events can be removed (signals `req_2` and `ack_2`). The elimination of internal events is done at TS level, thus obtaining a TS with less states. Finally, the STG for the whole circuit is derived by resynthesis.

We also performed an experiment by modeling a 3-token FIFO

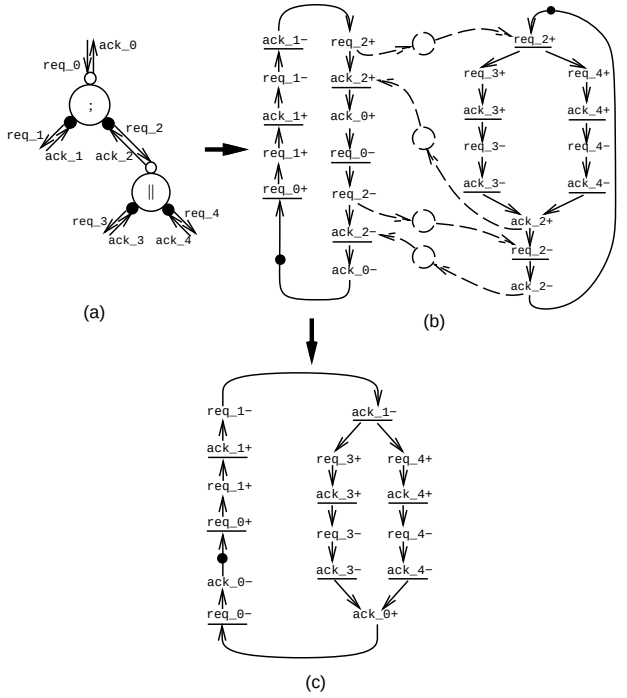


Figure 7: (a) Handshake circuits, (b) PN composition, (c) PN after elimination of internal events and synthesis.

in TANGRAM that resulted in a system with 20 handshake circuits. By iteratively composing the STGs of neighbouring handshake circuits and eliminating internal events (corresponding to internal communication channels), we obtained an STG with 6 signals (3 channels), 18 places, 23 transitions, 71 arcs and 50 markings (states).

6 Conclusions

Petri nets have been shown to be 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 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 symbiosis among the notions of ETS, *region* and *excitation region* in the same method has been crucial to derive efficient algorithms.

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. 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.

Acknowledgements

We are grateful to Marta and Maciej Koutny, who directed us towards the existing literature about regions, and to Marco A. Peña for performing the experiments with TANGRAM.

References

[1] E. Badouel, L. Bernardinello, and Ph. Darondeau. Polynomial algorithms for the synthesis of bounded nets. Technical Report 2316, INRIA, RENNES Cedex, France, 1994.

[2] K. van Berkel. *Handshake circuits: an intermediary between communicating processes and VLSI.* PhD thesis, Technical University of Eindhoven, 1992.

[3] L. Bernardinello, G. De Michelis, K. Petruni, and S. Vigna. On synchronic structure of transition systems. Technical report, Universita di Milano, Milano, 1994.

[4] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri nets from state-based models. Technical Report RR 95/09 UPC/DAC, Universitat Politecnica de Catalunya, April 1995.

[5] J. Cortadella, L. Lavagno, P. Vanbekbergen, and A. Yakovlev. Designing asynchronous circuits from behavioural specifications with internal conflicts. In *Proceedings of Int. Conf. on Adv. Res. in Asynch. Circ. and Syst.*, pages 106–115, November 1994.

[6] J. Ebergen. Personal communication. March 1995.

[7] J. Esparza and M. Nielsen. Decidability issues for Petri nets. *Petri Nets Newsletter*, 94:5–23, 1994.

[8] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. John Wiley and Sons, London, 1993.

[9] Robin Milner. A calculus of communication systems. In *Lecture Notes in Computer Science*, volume 92, 1980.

[10] M. Mukund. Petri nets and step transition systems. *Int. Journal of Foundations of Computer Science*, 3(4):443–478, 1992.

[11] M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.

[12] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962. (technical report Schriften des IIM Nr. 3).

[13] R.F. Sproull, I.E. Sutherland, and C.E. Molnar. The counterflow pipeline processor architecture. *IEEE Design and Test of Computers*, pages 48–59, Fall 1994.