

# Complete state encoding based on the theory of regions

Jordi Cortadella\*  
Universitat Politècnica de  
Catalunya, 08071  
Barcelona, Spain

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

Alex Kondratyev  
The University of Aizu  
Aizu-Wakamatsu,  
965-80 Japan

Luciano Lavagno‡  
Politecnico di Torino  
10129 Torino, Italy

Alex Yakovlev§  
University of Newcastle upon Tyne  
NE1 7RU England

## Abstract

*Synthesis of asynchronous circuits from Signal Transition Graphs (STGs) and/or State Graphs (SGs) involves solving state coding problems. A well-known example of such problems is that of Complete State Coding (CSC), which happens when a pair of different states in an SG has the same binary encoding. A standard way to approach state coding conflicts is to add new state signals into the original specification in such a way that the original behaviour remains intact. Existing methods have not yet been able to provide such theoretical foundation for event insertion, that could yield efficient practical results when applied to large models.*

*This paper aims at presenting such a general framework, which is based on two fundamental concepts. One is a region of states in an abstract labelled SG (called a Transition System). Regions correspond to places in the associated STG. The second concept is a speed-independence preserving set, which is strongly related to the implementability of the model in logic. Regions and their intersections offer "nice" structural properties that make them efficient "construction blocks" for event insertion. The application of our theory, through the software tool `petrify`, to state graphs of large size has proved to be successful.*

## 1 Introduction

The problem of *Complete State Coding (CSC)* is a fundamental problem in synthesis of asynchronous control circuits from Signal Transition Graphs (STGs) and State Graphs (SGs) [2]. This problem arises when a pair of semantically different states in an SG has the same binary encoding. Such states are said to be in *CSC conflict*. To resolve CSC conflicts, the synthesis

procedure must insert one or more new signals into the STG (or SG) specification. The value of these new signals have to be different in all pairs of states involved in a CSC conflict. State signal insertion must usually satisfy a set of important requirements: preserving equivalence of the specifications and implementability of the new and the original non-input signals without hazards (speed-independence preservation). The former requirement refers to the language generated by the STG. The latter implies that the implementability conditions (determinism, commutativity, persistency, deadlock-freedom and consistency) must be preserved in the transformed specification.

**Related Work.** A number of methods for solving the CSC problem are known to date [6, 8, 9, 10, 13, 14, 15, 18, 22, 23, 24, 26, 27, 30]. Reported experimental results and our own experience with the available tools for solving CSC ensured the authors of this paper that none of the published methods appears to be successful when applied to general SGs or STGs with more than a few thousand states. Methods from [13, 15, 18, 22, 24, 27] work at the STG level without doing state traversal. They allow to avoid state explosion and therefore can process large specifications if some additional constraints on an STG are given. Such constraints (no choice is allowed, or exactly one rising and falling transition for each signal is allowed, etc.) severely limit the design space and do not allow to get a solution for many practical specifications. [14] solves CSC problem by mapping an initial SG into a flow table synthesis problem and then using classical flow table minimization and state assignment methods. This method is restricted with live and safe free-choice STGs and cannot process large SGs due to limitations of classical state assignment methods.

In [23, 26] a very general framework for state assignment is presented. The CSC problem is formulated as state variable assignment on the state graph. The correctness conditions for such assignment are formulated as a set of Boolean constraints. The solution can be found using a Boolean satisfiability solver. Unfortunately, this approach allows to handle only relatively small specifications (hundreds of states) because

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

†This work has been partly supported by EPSRC grant 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.

the computational complexity of this method is double exponential from the number of signals in the SG. Although [6] presented a method to improve the resolution of the method based on a preliminary decomposition of the satisfiability problem, decomposition may produce sub-optimal solutions due to the loss of information incurred during the partitioning process. Moreover, the net contraction procedure used to decompose the problem has never been formally defined for non-free-choice STGs.

In [8, 9, 10] another method based on state signals insertion at the SG level was given. At first the excitation regions are distinguished in the SG. These are sets of states, which correspond to transitions of STG. Then the graph of CSC-conflicts between excitation regions is constructed and colored with binary encoded colors. Each bit of this code corresponds to a new state signal. After that new state signals are inserted into the SG using excitation regions of the original or previously inserted signals. The main drawback of this approach was its limitation to STGs without choices.

The method described in detail in [29, 30] is probably the most efficient and general published so far. It is based on partitioning of the state space into blocks which contain no internal CSC-conflicts. Similar to [9] a coloring procedure is used to find the optimal number of state signals to resolve all the CSC-conflicts between blocks of partitioning. Each of these state signals can be inserted using as excitation region the sets of states that immediately follow excitation regions (switching regions).

**Contribution of this paper.** This paper provides a general theoretical framework for insertion-based resolution of coding conflicts. The transformations described here are applied to abstract SGs, called Transition Systems (TSs) and to binary encoded SGs. This framework is aimed at being independent of the sort of conflicts between states to be resolved, therefore its application to CSC conflicts is only a special case. Another application of the method may be, e.g., solving Monotonous Cover conflicts [12], for technology mapping of asynchronous circuits in the basis of simple gates (AND, OR, NAND, NOR).

It is essential that the theory presented in this paper is based on the concept of *regions* in a TS. It renders an efficient framework for such transformations due to the two following major reasons. Firstly, regions are subsets of states which have a uniform “crossing” (exit-entry) relationship with events in a TS (see Section 3). They can be easily manipulated in intersections and unions, thus providing a good level of granularity in sectioning the TS (for example the excitation and switching regions are obtained as intersection of pre- and post-regions for the same transition). Secondly, regions in a TS directly correspond to places in an STG with a reachability graph isomorphic to the TS. This allows reconstructing an STG for the TS with all CSC conflicts resolved – an option much more suitable for the designer than viewing the TS.

The concept of regions was firstly presented in [17] and further applied to efficient generation of Petri Nets

and Signal Transition Graphs from state-based models [4]. The practical implementation of our method, which is only briefly outlined in this paper (those details require a separate presentation), uses symbolic BDD representation of the main objects in the insertion procedure. It has enabled us to solve CSC problem for state graphs with hundreds of thousands of states while the quality of the solutions obtained for smaller state graphs has been quite comparable with other known methods.

Our method differs from previous work as follows:

- Our technique for state signal insertion is more general and allows to explore more solutions than that of [29] and [30], since our method uses regions, their intersections and unions of intersections for insertion. On the other hand, [29] and [30] used excitation regions and switching regions, which are just particular cases of region intersections. Even though the authors of those papers admitted that intersections and unions of intersections of excitation and switching regions “could” be used for insertion, they claimed that “this does not seem necessary in practice”, and did not provide any method for reducing the complexity of the resulting huge search space.
- The notion of speed-independence preserving set (SIP-set) by which the insertion of state signals can be done without violation of speed-independence properties is generalized in comparison to [26], as will be shown in detail when discussing Theorem 4.1.
- Our method is proven to be complete for a fairly general class of SGs.
- An additional advantage of the theory presented in this paper is back-annotation at the STG level. The result of CSC resolution is shown to the user as a modified STG, so that the impact of state signal insertion on, e.g., the concurrency of the specification, can be more easily analyzed.

From the practical side we observed that although the tool *assassin* [28] which implements methods from [24, 26, 30] often allows better solutions than other previously known tools, it has difficulties in handling large specifications. For example, a *master-read* STG with 8932 states ran for more than 24 hours of CPU time at SPARC-10 machine without having solved CSC. Our tool *petrify* solved this example in 15 min of CPU time. We also solved examples with  $10^{11}$  states using a few hours of CPU time. It is worth to mention that on the basis of the region approach *petrify* succeeded in handling examples that were traditionally difficult for CSC solution by any other tool (see Section 8 for more details).

The paper is further organized as follows. Section 2 introduces both state-based and event-based models. Section 3 presents the basics of the theory of regions. Section 4 is dedicated to property-preserving event insertion, which uses the notion of speed-independence-preserving sets (SIP-sets) of states. Section 5 discusses

the issue of selection of SIP-sets, based on regions. Section 6 applies the event insertion technique to binary encoded TSs. Section 7 characterizes the set of STGs for which CSC can be solved using the proposed method. Section 8 outlines some experimental results. Finally, Section 9 draws conclusions.

## 2 State and Event models

**Transition systems.** A *transition system* (TS) might be viewed as an abstract state graph and is formally defined as a quadruple [17]  $A = (S, E, T, s_{in})$ , where  $S$  is a finite non-empty 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  $(s, e, s')$ .

The *reachability relation* between states is the transitive closure of the transition relation  $T$ . A *feasible sequence* is a (possibly empty) sequence of transitions  $\sigma$  between states  $s$  and  $s'$  (denoted by  $s \xrightarrow{\sigma} s'$  or simply by  $s \xrightarrow{*} s'$ ). A *feasible trace* is obtained from a feasible sequence by removing states. If  $s_1 \xrightarrow{e_1} s_2, s_2 \xrightarrow{e_2} s_3, s_3 \xrightarrow{e_3} s_4$  is a feasible sequence, then  $e_1, e_2, e_3$  is the corresponding feasible trace. We also write  $s \xrightarrow{e} s'$ , and  $s \xrightarrow{\sigma} s'$  if  $s \xrightarrow{e} s'$  or  $s \xrightarrow{\sigma} s'$ , correspondingly. Note that each state is reachable from itself. A state of a TS is called a *deadlock* if there is no event  $e \in E$  such that  $s \xrightarrow{e}$ .

Furthermore, a TS must satisfy the following four basic axioms:

- (A1) No self-loops,
- (A2) No multiple arcs between a pair of states,
- (A3) Every event has an occurrence,
- (A4) Every state is reachable from the initial state.

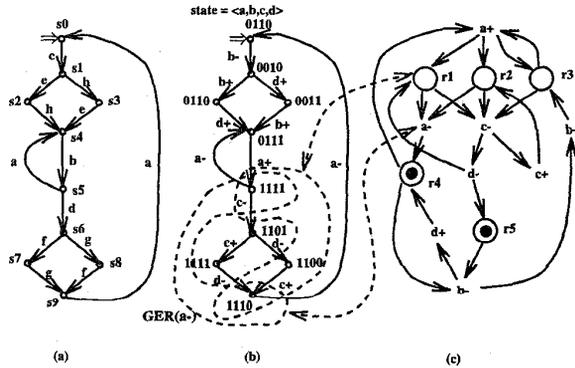


Figure 1: An example of Transition System (a), the corresponding SG (b), and STG (c)

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*. In the following we are interested only in deterministic TSs. An example of a deterministic TS is shown in Figure 1,a.

**State Graph.** For the purpose of logic synthesis TSs must be binary encoded. A *state graph*, SG, is a binary encoded TS. A SG is given by  $(A, X, \lambda_S, \lambda_E)$ , where  $A = (S, E, T, s_{in})$  is a transition system,  $X = X_I \cup X_O$  is the set of binary signals,  $X_I$  is the set of input signals, and  $X_O$  is the set of output signals<sup>1</sup>, such that  $X_I \cap X_O = \emptyset$ .

Each state  $s \in S$  in the SG is labelled with a *binary vector*  $(s(1), s(2), \dots, s(n))$  according to the signals  $X = \{x_1, x_2, \dots, x_n\}$  of the system. The labeling is given by a *state assignment function*  $\lambda_S : S \times X \mapsto \{0, 1\}$ . For a given state  $s \in S$ ,  $s(i)$  denotes the  $i$ -th component of  $s$  corresponding to the value of signal  $x_i \in X$ .

Each event  $e \in E$  in the SG is labelled with a *signal transition*. The labeling is given by an *event assignment function*  $\lambda_E : E \mapsto X \times \{+, -\}$ . Each signal transition can be represented as  $x_i+$  or  $x_i-$  for the rising ( $0 \rightarrow 1$ ) or falling ( $1 \rightarrow 0$ ) transition of signal  $x_i$ .  $x_i^*$  is used to depict either a " $x_i+$ " transition or a " $x_i-$ " transition. Further, if no confusion arises, we will denote different signal names by different letters  $a, b, \dots$  instead of  $x_1, x_2, \dots$ . Also,  $(s, x_i^*, s') \in T$  stands for  $(s, e, s') \in T \wedge \lambda_E(e) = x_i^*$ .

An SG has a *consistent state assignment* (we call such an SG consistent) if the following conditions for assignment functions are met: let  $(s, e, s') \in T$  then

- (1) if  $\lambda_E(e) = x_i+$ , then  $s(i) = 0$  and  $s'(i) = 1$ ;
- (2) if  $\lambda_E(e) = x_i-$ , then  $s(i) = 1$  and  $s'(i) = 0$ ;
- (3) in all other cases  $s(i) = s'(i)$ .

Consistent state assignment is a necessary condition for deriving logic functions for signals encoding a SG [2]. Figure 1,b shows a consistent SG which is obtained by binary encoding of the TS from Figure 1,a. After binary encoding, for example, event  $c$  is mapped into signal transition  $b^-$  and state  $s_1$  is mapped into binary code  $\langle a, b, c, d \rangle = 0010$ .

**Complete State Coding.** An unambiguous state assignment is required for deriving logic for encoding binary signals. Logic must be derived only for output signals and therefore the unambiguous state assignment must concern only output signals. This requirement is called *Complete State Coding* (CSC, [2]):

A SG is said to satisfy the Complete State Coding requirement if for any two states  $s_1$  and  $s_2$  which are assigned the same binary vectors the sets of enabled output signals are identical.

Let  $a$  and  $b$  be output and  $c$  and  $d$  be input signals for the SG in Figure 1,b. States  $s_0$  and  $s_2$  have the same binary code 0110. Output signal  $b$  is enabled in

<sup>1</sup>The output signals include both external output and internal signals of the modeled circuit.

$s_1$  and is not enabled in  $s_2$ , therefore CSC is violated and we say that states  $s_0$  and  $s_2$  are in CSC *conflict*. Although states  $s_5$  and  $s_7$  are also assigned the same binary code 1111, they are not in CSC conflict, since no output signals are enabled in  $s_5$  and  $s_7$ .

**Petri Nets and Signal Transition Graphs.** A Petri Net is often a more compact model to represent systems with concurrency than a TS.

A Petri Net [20] 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 ( $m_1 \xrightarrow{t} m_2$ ). The sets of input and output places of transition  $t$  are denoted by  $\bullet t$  and  $t\bullet$ .

The Reachability Graph (RG) of a PN is a graph with:

- a vertex for each reachable marking of the PN and
- an arc  $(m_1, m_2)$  if and only if  $m_1 \rightarrow m_2$  in some firing sequence of the PN.

A net is called *safe* if no more than one token can appear in a place. Safe nets are used in many applications, since they have simple verification algorithms [5] and simple semantics. A *labeled* PN is a PN with a labeling function  $\lambda : T \rightarrow \mathcal{A}$  which puts into correspondence every transition of the net with a symbol (called label) from the alphabet  $\mathcal{A}$ . A *Signal Transition Graph* (STG) is a PN whose transitions are labelled with signal transitions ( $a+$ ,  $a-$ , ...). Places with one input and one output transition are called implicit places and are depicted as an arc connecting these two transitions. An STG expressing the same behavior as the SG from Figure 1,b is shown in Figure 1,c.

### 3 Basics of the theory of regions

In this section we will briefly review the theory of regions and will show how this theory allows to perform transformations between TSs and PNs (hence, between SGs and STGs).

**Regions.** Regions are sets of states which correspond to places in Petri Nets. 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  $e$  exactly one of the following conditions holds: 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,a. The set of states  $r_1 = \{s_5, s_8, s_9\}$  is a region, since all transitions labeled with  $a$  and with  $d$  exit  $r_1$ , and all transitions labeled with  $b$  and with  $g$  enter  $r_1$ . On the other hand,  $\{s_8, s_9\}$  is not a region since transition  $s_9 \xrightarrow{a} s_0$  exits this set, while another transition also labeled with  $a$ ,  $s_5 \xrightarrow{a} s_4$ , does not.

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. 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  $e^{\circ}$  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  $r$ . There are two pre-regions for event  $a$  in Figure 1:  $r_1 = \{s_5, s_8, s_9\}$  and  $r_2 = \{s_5, s_7, s_9\}$ . Both of them are minimal regions, since no subset of  $r_1$  or  $r_2$  is a region.

The following propositions state a few important properties of regions [1, 4, 17].

#### Property 3.1

1. If  $r$  and  $r'$  are two different regions such that  $r'$  is a subregion of  $r$ , then  $r - r'$  is a region.
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.
3. Every region can be represented as a union of disjoint minimal regions.

**Excitation regions.** While regions in a TS are related to places in the corresponding PN, an excitation region [9] 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  $a$ :  $ER_1(a) = \{s_5\}$  and  $ER_2(a) = \{s_9\}$ . The corresponding GER for event  $a$  is  $GER(a) = \{s_5, s_9\}$ .

**Deriving Petri Nets from Transition Systems.** The procedure to synthesize a PN from an elementary TS 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  $s_{in} \in r_i$ ;

- 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  $a$ .

This procedure allows to obtain a safe PN with a RG isomorphic to the initial TS or to its minimized version if the initial TS is *elementary* [17]. As shown in [4] elementarity for minimal TS can be defined by the following two conditions.

- *Excitation closure.*  
For each event  $a$ :  $\bigcap_{r \in \bullet_a} r = GER(a)$ ;
- *Event effectiveness.*  
For each event  $a$ :  $GER(a) \neq \emptyset$ ;

As shown in Figure 1, region  $r_1$  is mapped into place  $r_1$  of the STG.

If a TS is not elementary, then it is always possible to transform it to an elementary one by label splitting (one label  $a$  which causes violations of elementarity is substituted in the TS by a few independent labels  $a_1, a_2, \dots$ ) or by inserting dummy transitions. Therefore, for any TS an equivalent safe PN can be synthesized.

#### 4 Constrained transformations of TSs

In this section we describe constrained transformations of TSs which preserve equivalence and other important properties. In particular, we formalize the notion of behavioral equivalence for TSs, and we define speed-independence.

**Speed-Independent Transition Systems.** A design is speed-independent if its behavior does not depend on the speed of its components (gates). As shown in [7], two properties ensure that a deterministic TS allows for a speed-independent implementation: *persistence* and *commutativity*. The persistence property states that no event can be disabled by any other event. The commutativity property guarantees that the same state of the TS is reached under any order of enabled event firing.

##### Definition 4.1 (Event persistence)

Let  $A = (S, E, T)$  be a transition system. An event  $a \in E$  is said to be persistent in  $r \subseteq S$  iff:  $\forall s1 \in r : [s1 \xrightarrow{a} \wedge (s1 \xrightarrow{b} s2) \in T] \implies s2 \xrightarrow{a}$

An event  $a \in E$  is said to be persistent if  $a$  is persistent in  $S$ .

##### Definition 4.2 (Commutativity)

A transition system  $A$  is called commutative if for any traces  $ab$  and  $ba$  that are feasible from some state  $s1 \in S$  both traces lead to the same state, i.e., if  $s1 \xrightarrow{a} s2, s2 \xrightarrow{b} s4$  and  $s1 \xrightarrow{b} s3, s3 \xrightarrow{a} s5$  then  $s4 = s5$ .

**Trace equivalence.** The set of feasible traces of a TS  $A$  is called the *language* accepted by  $A$  and is denoted as  $L(A)$ . If  $p$  is a feasible trace for  $A$ , then its projection on a subset of events  $E_1 \subseteq E$ , denoted as  $p \downarrow E_1$ , is a sequence of events  $p'$  obtained from  $p$  by deleting all events from  $E - E_1$ . If  $L(A)$  is the language accepted by  $A$ , then its projection  $L(A) \downarrow E_1$  is the set of sequences  $\{p \downarrow E_1 : p \in L(A)\}$ .

Let  $A' = (S', E', T')$  and  $A = (S, E, T)$  be two TSs such that  $E \subseteq E'$ . Then, TSs  $A$  and  $A'$  are *trace equivalent* if  $L(A') \downarrow E = L(A)$ . Additionally to trace equivalence, the following properties must be preserved after transforming a TS: persistency, commutativity, determinism, and deadlock freedom. The first three properties guarantee that the new TS allows for a speed-independent implementation. The latter property guarantees that liveness of the initial TS is preserved. It is defined as follows: if state  $s'$  is a deadlock in  $A'$  and is reachable from the initial state  $s'_in$  by a feasible trace  $p'$ , then state  $s$  of the original TS reachable from  $s_in$  by a feasible trace  $p = p' \downarrow E$  is a deadlock in  $A$ .

**Event insertion.** The basic transformation is the *insertion of a single event* into a TS. There can be different schemes of event insertion that preserve trace equivalence [3]. In this paper we will rely on a simple one which consists of two steps and is similar to [9, 26, 30]:

- Choosing in the original TS a set of states  $r$  in which the new event  $x$  will be enabled.  $r$  corresponds to a generalized excitation region of event  $x$  in the new TS and therefore is denoted as  $ER(x)$  in Figure 2.
- Delaying all transitions that exit the set of states  $r$  until event  $x$  fires.

##### Definition 4.3 (Event insertion)

Let  $A = (S, E, T)$  be a transition system and  $x \notin E$  be a new event. Assume that  $r \subseteq S$  is an arbitrary subset of states. Let  $r', r' \cap S = \emptyset$ , be a set of new states such that for each  $s \in r$  there is one state  $s' \in r'$  and vice versa. The insertion of  $x$  in  $A$  by  $r$  produces another transition system  $A' = (S', E', T')$  defined as follows:

$$\begin{aligned} S' &= S \cup r' \\ E' &= E \cup \{x\} \\ T' &= T \cup \{(s \xrightarrow{x} s') | s \in r \wedge s' \in r'\} \cup \\ &\quad \{(s1' \xrightarrow{a} s2') | s1, s2 \in r \wedge (s1 \xrightarrow{a} s2) \in T\} \cup \\ &\quad \{(s1' \xrightarrow{a} s2) | s1 \in r \wedge s2 \notin r \wedge (s1 \xrightarrow{a} s2) \in T\} - \\ &\quad \{(s1 \xrightarrow{a} s2) | s1 \in r \wedge s2 \notin r\} \end{aligned}$$

By Definition 4.3 transforming  $A$  to  $A'$  via adding the new event  $x$  by a set of states  $r$  leads to splitting all states  $s \in r$  in  $S$  into two states  $s$  and  $s'$  in  $S'$ . All other states  $s \notin r$  are in correspondence with only one state in  $S'$ . Figure 2 illustrates how event insertion is performed.

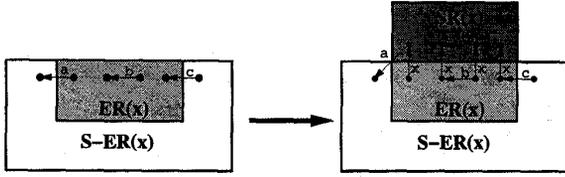


Figure 2: Insertion of event  $x$  from  $ER(x)$

**Speed-independence preserving sets.** It is easy to show that the insertion of event  $x$  by Definition 4.3 always preserves trace equivalence, determinism and deadlock-freedom [3]. Persistency and commutativity, on the other hand, are not automatically preserved, and need a more careful analysis.

**Definition 4.4 (SIP-set)** Let  $A = (S, E, T)$  be a transition system,  $x \notin E$  be a new event and  $r \subseteq S$ . Let  $A' = (S', E', T')$  be a transition system obtained after inserting  $x$  by  $r$ .  $r$  is said to be a speed-independence preserving set (SIP-set) iff:

1.  $\forall a \in E : a$  is persistent in  $A \implies a$  is persistent in  $A'$
  2.  $A$  is commutative  $\implies A'$  is commutative.
- If  $r$  satisfies only condition 1 then  $r$  is a persistency preserving set.

The following theorem determines two conditions for preserving persistency and commutativity.

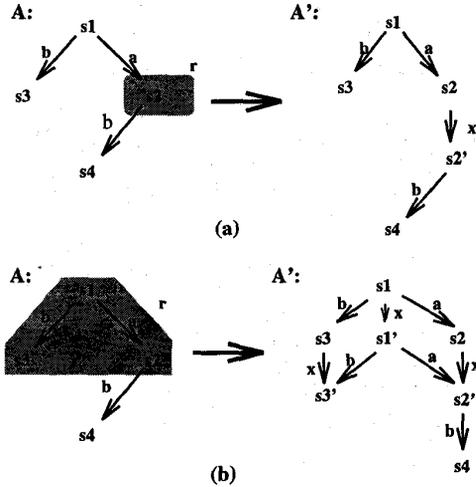


Figure 3: Set of states  $r$  is not persistency preserving.

**Theorem 4.1** Let  $A = (S, E, T)$  be a TS and  $r \subseteq S$  a subset of states.  $r$  is a persistency-preserving set iff

$$[(s1 \xrightarrow{b} s3), (s2 \xrightarrow{b} s4), (s1 \xrightarrow{a} s2) \in T \wedge s2 \in r, s4 \notin r] \implies s1 \in r \wedge s3 \notin r \quad (1)$$

Let  $A = (S, E, T)$  be a commutative transition system and  $r \subseteq S$  be a persistency preserving set. Then  $r$  is a SIP-set iff:

$$s1 \xrightarrow{a} s2, s2 \xrightarrow{b} s4, s1 \xrightarrow{b} s3, s3 \xrightarrow{a} s4 \in T \wedge (s1, s2 \in r \wedge s3 \notin r) \implies s4 \notin r \quad (2)$$

This theorem refines conditions for speed-independence from [23]. It allows to handle correctly the so called asymmetric “fake” conflicts between signals ([11]). Consider, for example, Figure 3.(a), where there is no arc between  $s3$  and  $s4$ . On the other hand, SIP conditions were defined in [23] only with respect to complete diamonds of states. Hence, the conditions stated in [23] are not sufficient to find the violation of persistency in cases like that of Figure 3.a.

Figure 3 shows two possible cases of violation of the persistency preserving condition (1) from Theorem 4.1. In both cases event  $b$  becomes non-persistent. Note that event  $x$  is persistent by construction in the TS obtained after the insertion. Hence, if a persistency preserving set is used for signal insertion, then no new non-persistencies can arise.

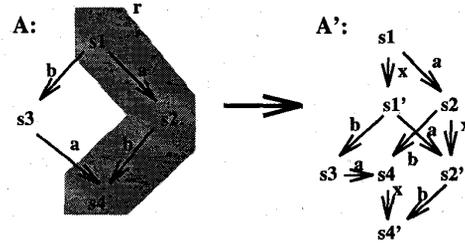


Figure 4: Commutativity violation after signal insertion

Figure 4 shows a violation of commutativity when a set of states  $r$  does not meet condition (2) of Theorem 4.1. Figure 5 shows allowable correct intersections of a SIP-set with all state diamonds in a SG.

## 5 Selecting SIP-sets

This section presents a few basic properties which allow us to formulate improved strategies for selection of SIP-sets. In [23, 26] SIP-sets are selected by solving a satisfiability problem. Efficiently constraining the search space for SIP-sets is problematic since in the reduction to the satisfiability problem each state in SG is considered separately (it is encoded by two binary signals) that quickly leads to unmanageable complexity when solving the satisfiability instance. In [8, 9, 10] SIP-sets are constructed from excitation regions of the original signals and previously introduced state signals. [30] generalized this method in such way that both ERs and switching regions (SRs) are used for SIP-sets. In this paper we further generalize this method: SIP-sets are constructed as regions, their intersections and union of intersections. We will show below that regions ensure to automatically find valid SIP-sets, rather than checking for SIP a posteriori, which is considerably less efficient. Note that ERs and SRs are particular cases of region intersections. Therefore, our method allows to explore a larger search space for SIP-sets and to find more efficient solutions.

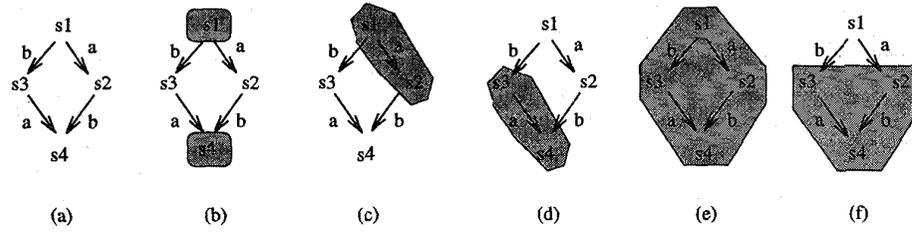


Figure 5: SIP-sets for a state diamond

**Property 5.1** *If  $r$  is a region in a commutative elementary transition system, then  $r$  is a SIP-set.*

The proof of this property is trivial. At the PN level this property corresponds to a following structural transformation: place  $r$  is substituted by two places  $r$  and  $r'$  with a new intermediate transition labelled with  $x$ . Place  $r$  has only one output transition,  $x$ , and all transitions which belong to  $r \bullet$  in the initial PN belong to  $r' \bullet$  in the new PN. Obviously, such transformations cannot violate persistency or commutativity for any event.

**Property 5.2** *If  $r$  is an excitation region of event  $c$  in a commutative transition system  $A$  and  $c$  is persistent in  $r$ , then  $r$  is a SIP-set.*

Intuitively, this property can be stated as follows: delaying a persistent event cannot create violations of persistency or commutativity. At the PN level this means that substituting a persistent transition by a sequential composition of two transitions preserves persistency and commutativity. At the circuit level this property corresponds to a well-known fact: inserting delays at the gate outputs before a wire fork does not violate semi-modularity of the circuit [16]. Most of the previous methods for CSC used variations of property 5.2 [2, 9, 14, 23, 30].

**Definition 5.1 (Exit and input border)** *Let  $A = (S, E, T)$  be a transition system. Given a subset of states  $r \subseteq S$ , the exit border of  $r$  (denoted as  $EB(r)$ ) and the input border of  $r$  (denoted as  $IB(r)$ ) are defined as follows:*

$$EB(r) = \{s \in r | \exists a \in E, s' \in S : s \xrightarrow{a} s' \in T \wedge s' \notin r\}$$

$$IB(r) = \{s \in r | \exists a \in E, s' \in S : s' \xrightarrow{a} s \in T \wedge s' \notin r\}$$

Exit borders of regions and intersection of pre-regions of the same event can also be safely used as SIP-sets under the following conditions.

**Property 5.3** *Let  $A = (S, E, T)$  be a commutative elementary transition system and let  $r$  be a region in  $A$ . If all the exit events of  $r$  are persistent then  $EB(r)$  is a SIP-set.*

**Proof:** 1. Violations of Condition 1 (Figure 3).

If Condition 1 for a SIP-set is violated, then  $s2 \in EB(r)$  and  $s4 \notin EB(r)$ . Hence, there exists event  $c$  such that  $s2 \xrightarrow{c} s5$ ,  $s5 \notin r$ . Clearly  $c$  is an exit event for  $r$  and from the properties of a region any state in which  $c$  is enabled belongs to  $r$ . If  $c \neq b$  then from  $s4 \notin EB(r)$  follows that event  $c$ , which is enabled in  $s2$ , becomes disabled in  $s4$ . This contradicts the assumption that all exit events of  $r$  are persistent.

If  $c = b$  then  $s1$  belongs to  $EB(r)$  (contradiction with Figure 3,a) and  $s3$  must be out of  $r$  (contradiction with Figure 3,b).

2. Violations of Condition 2 (Figure 4). By the same consideration if  $c \neq b$  then  $c$  becomes disabled in  $s3$ . If  $c = b$  then  $s4$  cannot be in  $EB(r)$ .  $\square$

A set of states  $S$  is called *forward connected* if for any pair of states  $s_1, s_2 \in S$  there is a state  $s_3 \in S$  ( $s_3$  may coincide with  $s_1$  or with  $s_2$ ) such that  $s_1 \xrightarrow{\sigma} s_3$ ,  $s_2 \xrightarrow{\sigma_1} s_3$  and all states of  $\sigma$  and  $\sigma_1$  belong to  $S$ .

**Property 5.4** *Let  $A = (S, E, T)$  be a commutative elementary transition system and let  $r_1, r_2$  be pre-regions of the same event. If  $r_1 \cap r_2$  is forward connected and all exit events of  $r_1 \cap r_2$  are persistent, then  $r_1 \cap r_2$  is a SIP-set.*

**Proof:** Assume that  $A = (S, E, T)$  is a commutative elementary transition system. Assume also that  $r_1, r_2$  are pre-regions of the same event  $b \in E$ ,  $r_1 \cap r_2$  is forward connected, and all exit events of  $r_1 \cap r_2$  are persistent. Let us prove that  $r_1 \cap r_2$  is a SIP set, i.e.,  $r_1 \cap r_2$  is persistency-preserving and commutativity-preserving.

Assume that  $A'$  is the TS obtained after inserting a new event  $x$  by  $r_1 \cap r_2$ . We need to prove that persistency and commutativity are preserved in  $A'$ .

**Persistency preservation.**

(1) A new event  $x$  and all events from  $E$  which do not exit  $r_1 \cap r_2$  are persistent in  $A'$  by construction.

(2) Consider event  $b$ . Let us refer to condition (1) from Theorem 4.1 and Figure 3. Since  $r_1, r_2 \in \bullet b$ , all transitions labelled with  $b$  must exit both  $r_1$  and  $r_2$ .

Hence,  $s1 \xrightarrow{b} s3$  exits both  $r_1$  and  $r_2$  and therefore,  $s1 \xrightarrow{b} s3$  exits  $r_1 \cap r_2$ . Condition  $s1 \in r_1 \cap r_2 \wedge s3 \notin r_1 \cap r_2$  is satisfied and persistency holds by Theorem 4.1.

(3) Consider event  $b'$  other than  $b$  such that  $b'$  exits  $r_1 \cap r_2$  but  $r_1$  or  $r_2$  are not pre-regions for  $b'$ . Transition  $s_2 \xrightarrow{b'} s_4$  from condition (1) of Theorem 4.1 exits  $r_1 \cap r_2$ . Hence,  $s_2 \xrightarrow{b'} s_4$  exits either  $r_1$  or  $r_2$ . Let us assume, for example, that  $s_2 \xrightarrow{b'} s_4$  exits  $r_1$ . Then transition  $s_1 \xrightarrow{b'} s_3$  also exits  $r_1$ .

Let  $s_2 \in GER(b)$ . Since  $b$  is persistent and  $s_2 \xrightarrow{b'} s_4$ , state  $s_4 \in GER(b)$  and  $s_2 \xrightarrow{b'} s_4$  is internal to  $GER(b)$ . In an elementary TS the intersection of pre-regions for the same event  $b$  gives the excitation region for  $b$ . Hence,  $s_2 \xrightarrow{b'} s_4$  is internal to  $r_1$  and we have reached a contradiction.

Let  $s_2 \in (r_1 \cap r_2) - GER(b)$ . Since  $r_1 \cap r_2$  is forward connected three cases are possible:

- (1)  $\exists s' \in GER(b) : s_2 \xrightarrow{\sigma} s'$ ,
- (2)  $\exists s' \in GER(b) : s' \xrightarrow{\sigma} s_2$ , and
- (3)  $\exists s' \in GER(b), s'' \in (r_1 \cap r_2) - GER(b) : s_2 \xrightarrow{\sigma} s'' \wedge s' \xrightarrow{\sigma^1} s''$ .

Let us consider the first case. Event  $b'$  is persistent and since  $b'$  exits  $r_1$ , the following condition holds:  $b' \notin \sigma$ . Therefore, there is a state  $s_5$  such that  $s' \xrightarrow{b'} s_5$ . Since  $s' \in GER(b)$  and  $b$  is persistent the following holds:  $s_5 \xrightarrow{b}$ . Therefore,  $s_5 \in GER(b)$  and  $s' \xrightarrow{b'} s_5$  is internal for  $GER(b)$  transition. Hence,  $s' \xrightarrow{b'} s_5$  is internal to  $r_1$  and we have reached a contradiction with the assumption that  $b'$  exits  $r_1$ .

Let us consider the second case. Since  $b$  is persistent,  $r_1 \cap r_2$  is forward connected and both  $r_1$  and  $r_2$  are pre-regions for  $b$ , then event  $b \notin \sigma$ . Therefore, state  $s_2 \in GER(b)$  and both  $b$  and  $b'$  are enabled in  $s_2$ . Since  $b$  is persistent we may conclude that transition  $s_2 \xrightarrow{b'}$  is internal for  $GER(b)$  and hence is also internal for  $r_1$ . We again have reached a contradiction.

Let us consider the third case. Since  $b$  is persistent,  $r_1 \cap r_2$  is forward connected and both  $r_1$  and  $r_2$  are pre-regions for  $b$ , event  $b$  is enabled in  $s''$  which implies that  $s'' \in GER(b)$ . Therefore, we have reduced the third case to the first case, which has been already considered.

**Commutativity preservation.** Let us refer to condition (2) from Theorem 4.1 and Figure 4. Given a diamond  $s_1 \xrightarrow{a} s_2, s_2 \xrightarrow{b} s_4, s_1 \xrightarrow{b} s_3, s_3 \xrightarrow{a} s_4$ , the commutativity property may be violated only in one case: if state  $s_3 \notin r_1 \cap r_2$  and states  $s_1, s_2, s_4 \in r_1 \cap r_2$ . In such case transition  $s_1 \xrightarrow{b} s_3$  exits  $r_1 \cap r_2$  and therefore must exit  $r_1$  or  $r_2$ . Assume for example that  $s_1 \xrightarrow{b} s_3$  exits  $r_1$ . On the other hand, transition  $s_2 \xrightarrow{b} s_4$  is internal for  $r_1 \cap r_2$  and hence does not exit  $r_1$ . We have reached a contradiction with the definition of a region.  $\square$

A significant consequence of these properties is that the good candidates for insertion can be built

on the basis of regions and their intersections, since they guarantee to preserve equivalence and speed-independence. One may also conclude that SIP-sets for event insertion can be built very efficiently from regions rather than states.

## 6 Transformations of State Graphs

The binary encoding of a TS to obtain an SG implies additional constraints for inserting new events: each inserted event has to be interpreted as a signal transition and therefore consistency of state assignment must be preserved. Any event insertion scheme which preserves trace equivalence (like those in Definition 4.3) also preserves consistency for the original signals. Special care must be taken to ensure consistency of the new signals (that are usually called *state signals*).

A specific class of SGs transformations can be defined as follows:

1. Insertion is made by signals not by events. Therefore, instead of inserting a single event two signal transitions of a new signal are inserted at each step:  $x+$  and  $x-$ . Two sets of states for insertion,  $GER(x+)$  and  $GER(x-)$ , are defined simultaneously such that  $GER(x+) \cap GER(x-) = \emptyset$ .
2. Similar to TSs transformations, both sets for insertion,  $GER(x+)$  and  $GER(x-)$ , must be SIP-sets. In addition, consistency of state assignment for signal  $x$  is required.

Given an SG with a set of binary states  $S$ , a partition for the insertion of signal  $x$ , called *I-partition*, is a partition of  $S$  into four blocks ([25]):  $S^0, S^1, S^+$  and  $S^-$ .  $S^0(S^1)$  defines the states in which  $x$  will have the stable value 0 (1).  $S^+(S^-)$  defines  $GER(x+)$  ( $GER(x-)$ ).

Let  $A$  be a consistent SG and let  $I = \{S^0, S^1, S^+, S^-\}$  be an I-partition of  $A$ . SG  $A'$  obtained by inserting signal  $x$  by partition  $I$  is consistent iff the only allowed arcs crossing boundaries of the partition blocks are the following:  $S^0 \rightarrow S^+ \rightarrow S^1 \rightarrow S^- \rightarrow S^0, S^+ \rightarrow S^-$  and  $S^- \rightarrow S^+, S^+ \rightarrow S^0$  and  $S^- \rightarrow S^1$ .

Arcs like those shown in Figure 6 are forbidden. The proof of this property trivially follows from the rules of insertion for events  $x+$  and  $x-$  (see Definition 4.3).

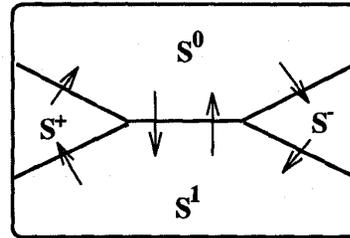


Figure 6: Illegal transitions in an I-partition

An I-partition can be found in two steps:

- Find a bipartition  $\{b, \bar{b}\}$ , ( $\bar{b} = S - b$ ) of  $S$ . The value of signal  $x$  is constant inside blocks  $b$  and  $\bar{b}$ .
- Choose  $GER(x+)$  and  $GER(x-)$  at the boundaries of blocks  $b$  and  $\bar{b}$  respectively.

The boundaries might be defined in two ways: as exit borders or as input borders. Figure 7,a shows insertion by exit borders: given a bipartition  $\{b, \bar{b}\}$ ,  $GER(x+) = EB(b)$  and  $GER(x-) = EB(\bar{b})$  (or vice versa). Figure 7,b illustrates insertion by input borders. In this case  $GER(x+) = IB(b)$  and  $GER(x-) = IB(\bar{b})$  (or vice versa).

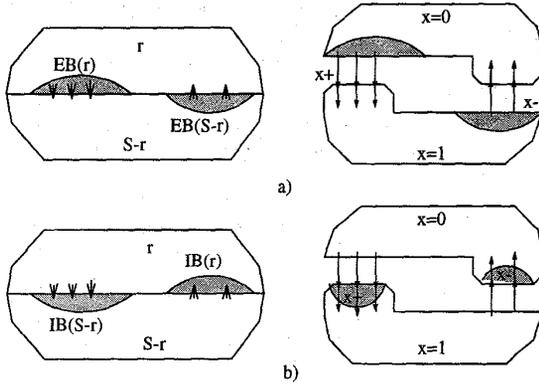


Figure 7: Signal insertion by exit (a) and input (b) borders

In general, using exit and input borders as insertion sets for new signal transitions does not always guarantee the consistency for the new signal. It may be necessary to enlarge the exit border  $EB(b)$  with those states of the block  $b$  which are directly reachable from  $EB(b)$ . Similarly, for input border  $IB(b)$  an enlargement is required with those states of  $b$  from which  $IB(b)$  can be entered. Such enlargement is not necessary if a border is *well-formed*.

**Definition 6.1** Let  $\{b, \bar{b}\}$  be a bipartition of a SG states.

1. The exit border  $EB(b)$  is called *well-formed* iff  $\forall s \in EB(b) : [\forall s' \xrightarrow{a} s' : s' \in \bar{b} \cup EB(b)]$  (similarly for  $EB(\bar{b})$ );
2. The input border  $IB(b)$  is called *well-formed* iff  $\forall s \in IB(b) : [\forall s' \xrightarrow{a} s : s' \in \bar{b} \cup IB(b)]$  (similarly for  $IB(\bar{b})$ ).

Let us refer to Figure 6. If well-formed exit borders are chosen for inserting new signal transitions, then the I-partition is defined as follows:  $S^0 = b - EB(b)$ ,  $S^+ = EB(b)$ ,  $S^1 = \bar{b} - EB(\bar{b})$ ,  $S^- = EB(\bar{b})$ . Since a transition can exit  $b$  only through the  $EB(b)$ , no arcs  $S^0 \rightarrow S^1$  and  $S^0 \rightarrow S^-$  in Figure 6 are possible. Due to the well-formedness of  $EB(b)$  it is not possible to return from  $EB(b)$  to  $b - EB(b)$ , hence arcs

$S^+ \rightarrow S^0$  are not possible either. A similar reasoning holds for  $EB(\bar{b})$ , hence none of the illegal transitions from Figure 6 can occur.

**Property 6.1** Let  $A$  be a consistent SG with a set of states  $S$  partitioned into  $\{b, \bar{b}\}$ . The SG  $A'$  obtained by inserting signal  $x$  by exit (input) borders of  $\{b, \bar{b}\}$  is consistent iff these borders are well-formed.

If the borders of a given partition  $\{b, \bar{b}\}$  of  $S$  are not well-formed, we can still use it by considering larger sets of states that guarantee consistency. Namely, given  $\{b, \bar{b}\}$ , we can define minimal well-formed extended EB and IB (denoted  $MWFEEB(b)$  and  $MWFEIB(b)$ ) as minimal well-formed enlargements of exit and input borders respectively.  $MWFEEB(b)$  can be calculated as the least fix point of the following recursion:

1.  $MWFEEB(b) = EB(b)$
2.  $[s \in MWFEEB(b) \wedge s' \in b \wedge s \rightarrow s'] \Rightarrow s' \in MWFEEB(b)$

A similar recursion can be applied for calculating  $MWFEEB(\bar{b})$ ,  $MWFEIB(b)$ , and  $MWFEIB(\bar{b})$ . Minimal well-formed extended borders hence are minimal sets of states for signal transition insertion which guarantee consistency.

## 7 Completeness of the method

In this section we will show that the method for CSC solution using region-based signal insertion is complete, i.e. it allows to solve all CSC conflicts, for a fairly general class of SGs.

A direct synthesis method for speed-independent implementation of STGs without choice has been proposed in [9]. It solves all CSC conflicts by construction. This method can be generalized to any safe STG [3] which is persistent with respect to the transitions of output signals (so called *output-persistent* STGs). Hence, this direct method can be applied to any SG for which a safe and output-persistent STG can be generated using regions as described in Section 2.

Generating such an STG is possible if an SG satisfies the following conditions: (1) it is deterministic, consistent, commutative and persistent by output signals, and (2) it is *elementary* after splitting all GERs into ERs. This result implies that for each SG which meets these conditions, the procedure of signal insertions based on intersection of regions will eventually converge.

Let us have a closer look at an I-partition, in order to estimate an upper bound on the number of state signals needed to solve all CSC conflicts. Assume that  $\{b, \bar{b}\}$  is a bipartition of a set of states. Assume that an I-partition is constructed from  $\{b, \bar{b}\}$  by exit borders, i.e.,  $S^+ = MWFEEB(b)$ ,  $S^- = MWFEEB(\bar{b})$  and  $S^0 = b - S^+$ ,  $S^1 = \bar{b} - S^-$  (see Figure 7,a).

Clearly, all the states from  $S^0$  and  $S^1$  will differ in the new SG obtained after the transformation by the

value of signal  $x$ . However, this is not the case for the states from  $MWFEEB(b)$  and  $MWFEEB(\bar{b})$ . Each state  $s \in MWFEEB(b)$  ( $s \in MWFEEB(\bar{b})$ ) is mapped into states  $s$  and  $s'$  in the new SG, such that  $s \xrightarrow{x^*} s'$ . Signal  $x$  has different values in  $s$  and  $s'$ . Thus, no CSC conflict in  $MWFEEB(b)$  and  $MWFEEB(\bar{b})$  is solved by  $x$ . Then we can use one more state signal  $y$  and another insertion scheme (by input borders) to distinguish these conflicts.

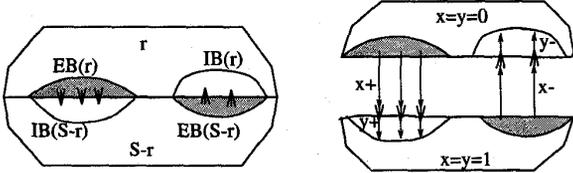


Figure 8: Signal insertion using both exit and input borders

This method is illustrated in Figure 8. At first, one additional state signal,  $x$ , is inserted by minimal well-formed extended exit borders of  $b$  and  $\bar{b}$  and then another state signal,  $y$ , is inserted by minimal well-formed extended input borders of the partition  $\{b', \bar{b}'\}$  inherited from  $\{b, \bar{b}\}$  by a new SG. The only CSC conflicts which are not solved by such insertion of two signals are those which exist between  $MWFEEB(b)$  and  $MWFEEB(\bar{b})$ ,  $MWFEEB(b)$  and  $MWFEIB(b)$ .

**Definition 7.1** Let  $\{b, \bar{b}\}$  be a bipartition of a SG states. Let  $s1$  and  $s2$  have the same binary code,  $s1 \in b$ , and  $s2 \in \bar{b}$ . States  $\{s1, s2\}$  are said to be distinguishable by partition  $\{b, \bar{b}\}$  if the following condition does not hold:

$$(s1 \in MWFEEB(b) \wedge s2 \in MWFEIB(\bar{b})) \vee (s1 \in MWFEIB(b) \wedge s2 \in MWFEEB(\bar{b}))$$

**Theorem 7.1** Let  $\{b, \bar{b}\}$  be a bipartition of a SG states. All distinguishable pairs of states  $\{s1, s2\}$  ( $s1 \in b, s2 \in \bar{b}$ ) will obtain different binary codes after inserting two state signals by  $MWFEEB$  and  $MWFEIB$  (the method from Figure 8).

The proof of this theorem can be found in [3]. It follows from the fact that if  $s1$  and  $s2$  are distinguishable by partition  $\{b, \bar{b}\}$  in SG  $A$ , then the corresponding states in  $A'$  are distinguishable by partition  $\{b', \bar{b}'\}$ , where  $b'$  and  $\bar{b}'$  are "images" of  $b$  and  $\bar{b}$  in the new SG,  $A'$ . Hence, if all CSC conflicts can be distinguished by  $k$  bi-partitions, then no more than  $2k$  state signals are needed for solving all CSC conflicts in a SG.

Moreover, it is easy to show, by using the same direct synthesis method of [9], that the number of places of an STG or, equivalently, the cardinality of an irredundant cover of minimal region of an SG gives a very loose worst-case upper bound for the minimal number of partitions solving all CSC conflicts.

The following corollary of Theorem 7.1 states the conditions for implementability of an SG as a speed-independent circuit.

**Corollary 7.1** Let  $A$  be a deterministic, consistent, commutative and output-persistent SG. Assume that for every pair of states  $s1, s2$  with the same binary code there exists a partition  $\{b, \bar{b}\}$  distinguishing  $s1$  and  $s2$  such that minimal well-formed extended exit and input borders of  $b$  and  $\bar{b}$  are SIP-sets. Then there exists a finite sequence of SIP insertions that yields an SG  $A'$  such that: (1)  $A'$  is trace equivalent to  $A$ , (2)  $A'$  is deterministic, consistent, commutative and output-persistent, and (3)  $A'$  satisfies CSC.

Indeed, if every pair of states with the same binary code is distinguishable by some partition, then according to Theorem 7.1 all CSC conflicts in this SG can be solved by inserting state signals. The consistency is preserved since signals are inserted by well-formed borders. Since these borders are SIP-sets (by the condition of Corollary 7.1), then commutativity and persistency are also preserved.

It follows from the direct synthesis method of [9], that if an SG is elementary after splitting the GER into ERs, then all CSC conflicts are distinguished by some bipartition, and therefore the method based on EB and IB insertion can be successfully applied. For non-elementary SGs the completeness of our approach is an open problem. However, the authors are not aware of any example which will be irreducible within the proposed approach.

## 8 Experimental results

The region-based approach presented in this paper has been integrated in petrify, a tool for the synthesis of Petri nets [4]. An experimental algorithm to calculate bi-partitions and insert state signals has been implemented. The algorithm uses heuristic search techniques to explore the space of valid bi-partitions and is guided by a cost function that attempts to find bi-partitions that maximize the number of solved CSC conflicts at low cost (i.e. minimizing the number of trigger signals of the new state signal).

This section presents some experimental results that illustrate the main features of the approach. The major advantages of our tool are:

- It can handle a wider class of STG and SG specifications than existing tools. This is due to the completeness of our method and to an efficient search strategy based on manipulating regions rather than states.

One of the examples that was traditionally difficult to solve (even by hand by expert designers) is depicted in Figure 9. Our region-based approach obtained the best solution among those we could obtain manually. Interestingly, the example is unsafe (the arcs  $a+ \rightarrow b-$  and  $b+ \rightarrow a-$  are 2-bounded), which shows that the applicability of the region-based approach extends beyond the class of elementary TS.

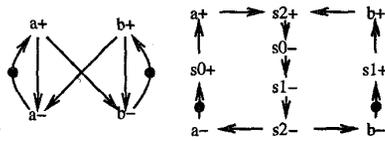


Figure 9: Difficult example to solve CSC

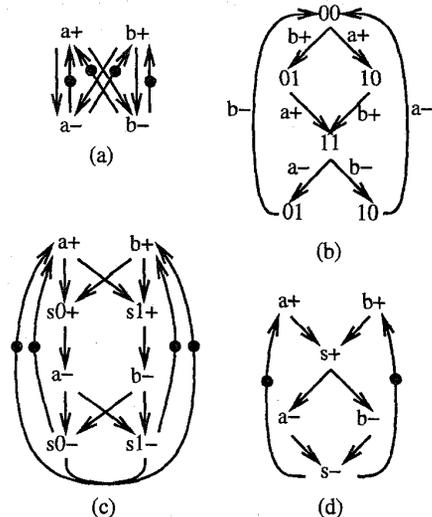


Figure 10: (a) STG, (b) SG, (c) solution obtained by ASSASSIN [28], (d) solution obtained by petrify.

- It can explore efficient solutions by manipulating sets of states at the level of regions, their intersections and unions of intersections.

Figure 10,a and 10,b depict an STG with CSC conflicts and its state graph. Figure 10,c presents a solution with the insertion of state signals based on a restricted exploration of only excitation and switching regions (as in [30]). In this case, no solution can be found with only one state signal. Moreover, the final solution obtained by ASSASSIN [28] manifests some asymmetry (between the positive and the negative phase of the transitions) produced by the dependency of the approach on the initial state.

Petrify can solve all CSC conflicts with only one state signal, deriving the STG shown in Figure 10,d. The state signal is inserted at the intersection of regions (post-regions of  $\{a+, b+\}$  and  $\{a-, b-\}$  respectively). The final STG can be implemented with a single C element.

- It can manage extremely large SGs generated by highly concurrent STGs.

Two factors are essential for this capability:

1. the exploration of blocks of states at the level of *regions* rather than states

2. the symbolic representation and manipulation of the state graph and regions by means of *Ordered Binary Decision Diagrams* [4]

One of the examples we used is a 16-process parallelizer built by means of the composition of 15 Tangram parallelizers [21]. After projecting the STG onto the input/output signals, an STG with 83 places, 68 transitions, 34 signals and  $1.5 \times 10^{11}$  markings (states) was obtained. To solve CSC, 16 state signals were inserted (the optimal solution in terms of number of state signals) and an STG with  $2.8 \times 10^{12}$  markings was retrieved. Even with such vast space of states, petrify was able to solve CSC in less than 4 hours of CPU time.

Further experimental results have shown that the running time used by petrify does not depend on the number of states, but on the complexity of the underlying Petri net which directly determines the number of variables used to encode the states [19].

## 9 Conclusions

We have presented a theoretical framework for insertion of new events into an asynchronous behavioural specification with the purpose of resolving state encoding conflicts. Our theory is based on the combination of two fundamental concepts. One is the notion of regions of states in a Transition System (an abstract labelled SG). The second concept is a speed-independence preserving set (SIP-set), which is strongly related to the implementability of the model in logic. Regions and their intersections can serve as bricks for efficient generation of SIP-sets.

The theory presented in this paper has been used in developing algorithms for the software tool petrify, which was originally created as a program for synthesizing Petri net-based specifications from state-based models [4]. The combination of the PN synthesis functionality with the framework for state-encoding event insertion described in this paper allows to solve CSC for large asynchronous specifications which were not solvable by any previously known approach. It also allows the user to view the result of the transformation applied to the Transition System, in the form of an STG. We are currently working on the application of the theory of event insertion to solving other state encoding problems involved in asynchronous synthesis.

## References

- [1] L. Bernardinello, G. De Michelis, K. Petruni, and S. Vigna. On synchronic structure of transition systems. Technical report, Universita di Milano, Milano, 1994.
- [2] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, MIT, June 1987.
- [3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. A region-based theory

- for state assignment in asynchronous circuits. Technical Report 95-2-006, University of Aizu, Japan, October 1995.
- [4] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri nets from state-based models. In *Proc. of ICCAD'95*, pages 164–171, November 1995.
  - [5] J. Esparza and M. Nielsen. Decidability issues for petri nets. *Petri Nets Newsletter*, 94:5–23, 1994.
  - [6] Jun Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, August 1995.
  - [7] R.M. Keller. A fundamental theorem of asynchronous parallel computation. *Lecture Notes in Computer Science*, 24:103–112, 1975.
  - [8] M. A. Kishinevsky, A. Y. Kondratyev, and A. R. Taubin. Formal method for self-timed design. In *Proceedings of the European Design Automation Conference (EDAC)*, 1991.
  - [9] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky. *Concurrent Hardware. The Theory and Practice of Self-Timed Design*. John Wiley and Sons Ltd., 1994.
  - [10] A. Kondratyev. *Design of self-timed circuits from change diagrams*. PhD thesis, LETI Leningrad, 1987. (in Russian).
  - [11] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig, and A. Yakovlev. Checking Signal Transition Graph implementability by symbolic BDD traversal. In *Proceedings of the European Design and Test Conference (ED&TC)*, pages 325–332, Paris, France, March 1995.
  - [12] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the Design Automation Conference*, pages 56–62, June 1994.
  - [13] A. Y. Kondratyev, L. Y. Rosenblum, and A. V. Yakovlev. Signal graphs: a model for designing concurrent logic. In *Proceedings of the 1988 International Conference on Parallel Processing*. The Pennsylvania State University Press, 1988.
  - [14] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits*. Kluwer Academic Publishers, 1993.
  - [15] K.-J. Lin and C.-S. Lin. On the verification of state-coding in STGs. In *Proceedings of ICCAD'92*, Santa Clara, CA, November 1992.
  - [16] D. E. Muller and W. C. Bartky. A theory of asynchronous circuits. In *Annals of Computing Laboratory of Harvard University*, pages 204–243, 1959.
  - [17] M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.
  - [18] E. Pastor and J. Cortadella. An efficient unique state coding algorithm for signal transition graphs. In *Proceedings of the International Conference on Computer Design*, October 1993.
  - [19] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In *15th International Conference on Application and Theory of Petri Nets*, Zaragoza, Spain, June 1994.
  - [20] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn, Institut für Instrumentelle Mathematik, 1962. (technical report Schriften des IIM Nr. 3).
  - [21] Kees van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*. International Series on Parallel Computing. Cambridge University Press, 1993.
  - [22] P. Vanbekbergen. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. In *Proceedings of the International Conference on Computer-Aided Design*, pages 184–187, November 1990.
  - [23] P. Vanbekbergen. *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications*. PhD thesis, Katholieke Universiteit Leuven, Imec, Belgium, September 1993.
  - [24] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. *IEEE Transactions on Computer-Aided Design*, pages 1426–1438, November 1992.
  - [25] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A generalized state assignment theory for transformations on Signal Transition Graphs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 112–117, November 1992.
  - [26] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A generalized state assignment theory for transformations on Signal Transition Graphs. *Journal of VLSI Signal Processing*, 7(1-2):101–116, 1994.
  - [27] A. V. Yakovlev and A. Petrov. Petri nets and parallel bus controller design. In *International Conference on Application and Theory of Petri Nets, Paris, France*. IEEE Computer Society, June 1990.
  - [28] C. Ykman-Couvreur, B. Lin, and H. De Man. ASSASIN: A synthesis system for asynchronous control circuits. Technical report, IMEC, September 1994. User and Tutorial manual.
  - [29] Ch. Ykman-Couvreur and B. Lin. Efficient state assignment framework for asynchronous state graphs. In *Proceedings of the International Conference on Computer Design*, 1995.
  - [30] Ch. Ykman-Couvreur and B. Lin. Optimized state assignment for asynchronous circuit synthesis. In *Second Working Conference on Asynchronous Design Methodologies*, pages 118–127, May 1995.