A Region-Based Theory for State Assignment in Speed-Independent Circuits

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

***Abstract*— State assignment problems still need satisfactory solutions to make asynchronous circuit synthesis more practical. A well-known example of such a problem is that of complete state coding (CSC), which happens when a pair of different states in a specification has the same binary encoding. A standard way to approach state coding conflicts is to insert new state signals into the original specification in such a way that the original behavior remains intact.**

This paper proposes a method which improves over existing approaches by coupling *generality*, *optimality*, and *efficiency*. The method is based on the use of a class of "ground objects," called *regions*, that play the role of a bridge between state-based specifications (transition systems, TS's) and event-based specifications (signal transition graphs, STG's). We need to deal with both types of specification because designers usually prefer a timing diagram-like notation, such as STG, while optimization and cost analysis work better at the state level.

A region in a transition system is a set of states that corresponds to a place in an STG (or the underlying Petri net). Regions are tightly connected with a set of *properties* that are to be preserved across the state encoding process, namely, 1) trace equivalence between the original and the encoded specification, and 2) implementability as a speed-independent circuit. We will build on a theoretical body of work that has shown the significance of regions for such property-preserving transformations, and describe a set of algorithms aimed at efficiently solving the encoding problem.

The algorithms have been implemented in a software tool called `petrify`. Unlike many existing tools, `petrify` represents the encoded specification as an STG. This significantly improves the readability of the result (compared to a state-based description in which concurrency is represented implicitly by interleaving), and allows the designer to be more closely involved in the synthesis process. The efficiency of the method is demonstrated on a number of "difficult" examples.

I. Introduction

In the last decade, signal transition graphs (STG's) [5], [14], [16], [22], [31], [32], [36], [42], [41] have attracted much of the attention of the asynchronous circuit design community due to their inherent ability to capture the main paradigms of asynchronous behavior: causality, concurrency, and data-dependent and nondeterministic choice. STG's are Petri nets [25] whose events are interpreted with signal transitions of a modeled circuit. Unlike other models, e.g., based on explicit state graph representation, STG's can specify circuit behavior in a compact form by defining *local* causality relations between signal transitions. The relations are represented by the *places* of the Petri net underlying the STG. These places also provide the abstract means of storage of a partial state of the model. The STG model, exactly like "classical" flow table models, although being formally consistent and correct [5], [6], [31], may be incomplete in that it may require some state signals to be added to those initially specified by the designer to ensure implementability. The complete state coding (CSC) problem is thus a fundamental problem in the synthesis of speed-independent control circuits from STG's or from state graphs (SG's) [6]. It is informally defined as follows. An STG satisfies the CSC property if every pair of different states which are assigned the same binary code enables exactly the same set of noninput signals. This condition is crucial for deriving logical functions for noninput signals, so that for each such signal, its *implied* (i.e., new) value in every reachable state is *uniquely* determined by the binary code (in terms of STG signal values) of the state.

The states which violate the above condition 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: 1) preserving behavioral equivalence of the specifications and 2) guaranteeing that both the new and the original noninput signals are implemented without hazards (preserving speed independence). Requirement 1) refers to the language generated by the STG. Requirement 2) implies that the implementability conditions (determinism, commutativity, persistency, deadlock freedom, and consistency) must be preserved in the transformed specification.

A. Related Work

A number of methods for solving the CSC problem are known to date [14], [16]–[18], [21]–[23], [28], [34]–[36], [38], [39], [41]. Reported experimental results and our own experience with the tools available for solving CSC assured us that none of the published methods appears to be successful when applied to general SG's or STG's with more than a few thousand states.

Manuscript received August 30, 1996; revised July 29, 1997. This work was supported in part by CICYT Grant TIC95-0419, EPSRC under Visiting Fellowship Grants GR/J72486, GR/J78334, GR/L24038, and Research Grant GR/J52327, and MURST Project "VLSI Architectures." This paper was recommended by Associate Editor A. Saldanha.

J. Cortadella is with the Universitat Politècnica de Catalunya, Barcelona 08071, Spain.

M. Kishinevsky and A. Kondratyev are with the University of Aizu, Fukushima 965-80, Japan.

L. Lavagno is with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy.

A. Yakovlev is with the University of Newcastle upon Tyne, Newcastle upon Tyne, U.K.

Publisher Item Identifier S 0278-0070(97)08473-X.

Methods from [21], [23], [28], [34], [36], [39] work at the STG level without doing state traversal. They avoid state explosion, and therefore can process large specifications if some additional constraints on an STG are given. Such constraints (e.g., freedom from choice, exactly one rising and falling transition for each signal, etc.) severely limit the design space, and do not produce solutions for many practical specifications. Reference [22] solves the CSC problem by mapping an initial SG into a flow table, and then using classical flow table minimization and state assignment methods. This method is restricted to live and safe free-choice STG's, and cannot process large SG's due to limitations of the classical state assignment methods.

In [35] and [38] a very general framework for state assignment is presented. The CSC problem is formulated as a search for a state variable assignment on the state graph. The correctness conditions for such an assignment are formulated as a set of Boolean constraints. The solution can be found using a Boolean satisfiability solver. Unfortunately, this approach allows handling only relatively small specifications (hundreds of states) because the computational complexity of this method is double exponential in the number of signals in the SG. Although [14] presented a method to improve effectiveness by means of preliminary decomposition of the satisfiability problem, decomposition may produce suboptimal 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 nonfree-choice STG's.

In [16]–[18], another method based on state signals insertion at the SG level was given. Here, first of all, the *excitation regions* are found 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 the excitation regions of the original or previously inserted signals. The main drawback of this approach was its limitation to STG's without choices.

The method described in detail in [40] and [41] is probably the most efficient and general of those published so far. It is based on partitioning the state space into blocks which contain no internal CSC conflicts. This method is essentially based on the concept of an *excitation region* for a signal transition (a set of states in which a signal is enabled to change its value) developed in [16]. Similarly to [16], a coloring procedure is used to find the optimal number of state signals to resolve all of the CSC conflicts between blocks of partitioning. Each of these state signals can be inserted using excitation regions or states that immediately follow excitation regions (switching regions). References [40] and [41] improve in terms of execution time over the previous methods that claimed broader generality (e.g., [37]) by adopting a *coarser granularity* in the exploration of the solution space. This coarser granularity has a price, though: as we will show in Section X, there are examples of STG's which cannot be solved by their method (nor by the previous ones, mainly due to the large number of states), unless changes in the specification (e.g., reductions in concurrency) are allowed.

B. Contribution of This Paper

This paper provides a general theoretical framework for insertion-based resolution of coding conflicts first outlined in [8] and [9]. The transformations described here are applied to abstract SG's, called transition systems (TS's), and to binary encoded SG's. 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 [2], [20], for technology mapping of speed-independent circuits.

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 following two major reasons. First, regions are subsets of states which have a uniform "crossing" (exit–entry) relationship with events in a TS. 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 postregions for the same transition). Second, regions in a TS directly correspond to places in an STG with a reachability graph bisimilar to the TS. This allows us to reconstruct 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 presented in [26], and further applied to efficient generation of Petri nets and signal transition graphs from state-based models [10].

The practical implementation of our method uses symbolic BDD representation of the main objects in the insertion procedure, as described in [29]. 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 to other known methods.

The following features differentiate our method from previous work.

- Our technique for state signal insertion is more general, and allows us to explore more solutions than that of [40] and [41]. It uses regions, their intersections, and unions of intersections for insertion, while the method of [40] and [41] is based on excitation regions and switching regions, which are only particular cases of region intersections.
- The idea of a speed-independence preserving set (SIP set), by which state signals can be inserted without violation of speed-independence properties, is generalized in comparison to [38], as will be shown in detail when discussing Theorem 4.1.
- Our method is proven to be complete for a fairly general class of SG's, thanks to the possibility of iterating the insertion of signals while reducing a measure of "distance" from the satisfaction of CSC. This iteration is shown to converge in Section VII.
- An additional advantage of the theory presented in this paper is back-annotation at the STG level. The result

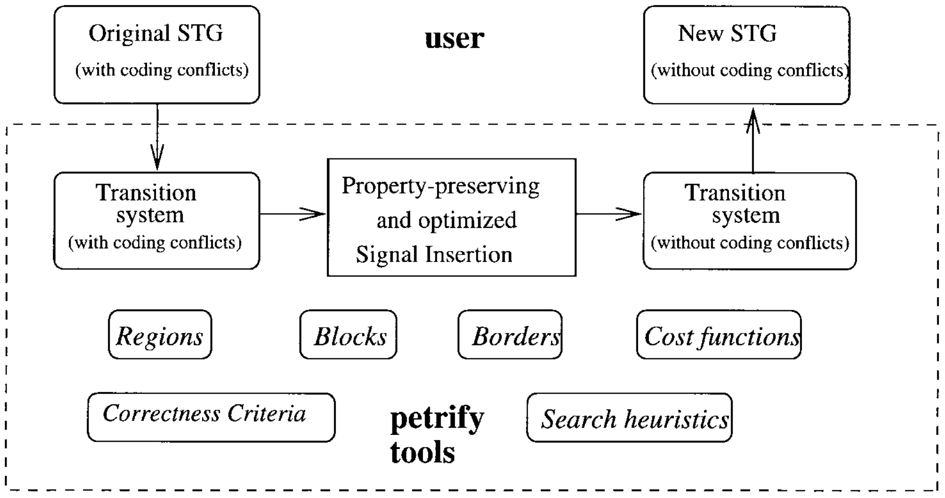

Fig. 1. Use of `petrify` in solving state coding problems by means of signal insertion.

of CSC resolution is shown to the user as a modified STG, so that the impact of state signal insertion on the specification can be more easily analyzed.

From the practical side, we observed that, although the tool `assassin` [42], which implements methods from [36], [38] and [41], 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 h of CPU time on a 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 in a few hours of CPU time. It is worth mentioning that the region-based approach helps `petrify` in handling examples that were traditionally difficult for CSC solution by any other tool (see Section X for more details). The overall context of this approach is captured in Fig. 1.

Although our main focus is on speed-independent circuits, we believe that our results can be extended to other styles of asynchronous circuit design. In particular, quasi-delay-insensitive circuit design [4] can benefit from our techniques since an SG-like intermediate representation also can be derived in that case.

On the other hand, synthesis methods for burst mode FSM's [27], [43] use timing assumptions (also known as *fundamental mode* operation) to ensure absence of hazards in the implemented circuit. In that case, an immediate extension of our results is not possible since we rely on explicit acknowledgment of transitions to ensure absence of hazards, and this is explicitly part of our notion of valid state signal transition insertion. However, since BMFSM's can be translated into STG's and implemented as speed-independent circuits, we can also handle those specifications and synthesize them. The result is probably less efficient than with the methods in [27], [43] (since we do not exploit timing assumptions), but is somewhat safer and more technology-independent. We are currently working toward an explicit incorporation of timing assumptions at the SG level in order to increase the level of optimization at the expense of robustness.

The paper is further organized as follows. Section II introduces both state-based and event-based models. Section III presents the basics of the theory of regions. Section IV is dedicated to property-preserving event insertion, which uses the notion of speed-independence-preserving sets (SIP sets) of states. Section V discusses the issue of selection of SIP sets, based on regions. The methods of state graph transformation to ensure CSC are presented in Section VI, and the completeness of the methods is shown in Section VII. Sections VIII and IX describe implementation aspects. Experiments (in comparison with other tools) are discussed in Section X. The conclusions are drawn in Section XI.

II. State and Event Models

A. Transition Systems

A *transition system* (TS) might be viewed as an abstract state graph, and is formally defined as a quadruple [26] $A = (S, E, T, s_{in})$, where S is a *finite nonempty* set of *states*, E is a set of *events*, $T \subseteq S \times E \times S$ is a *transition* relation, and s_{in} is an *initial state*. The elements of T are called the *transitions* of TS and will be often denoted by $s \xrightarrow{e} s'$ instead of (s, e, s'). The initial state will often be omitted in the following if it is not important.

The *reachability relation* between states is the transitive closure of the transition relation T. A *feasible sequence* is a (possibly empty) sequence of transitions σ 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 $s \xrightarrow{e}$, $\xrightarrow{e} s'$, and $s \xrightarrow{\sigma}$, $\xrightarrow{\sigma} s'$ 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 three basic axioms:

A1) no self-loops;
A2) every event has an occurrence;
A3) every state is reachable from the initial state.

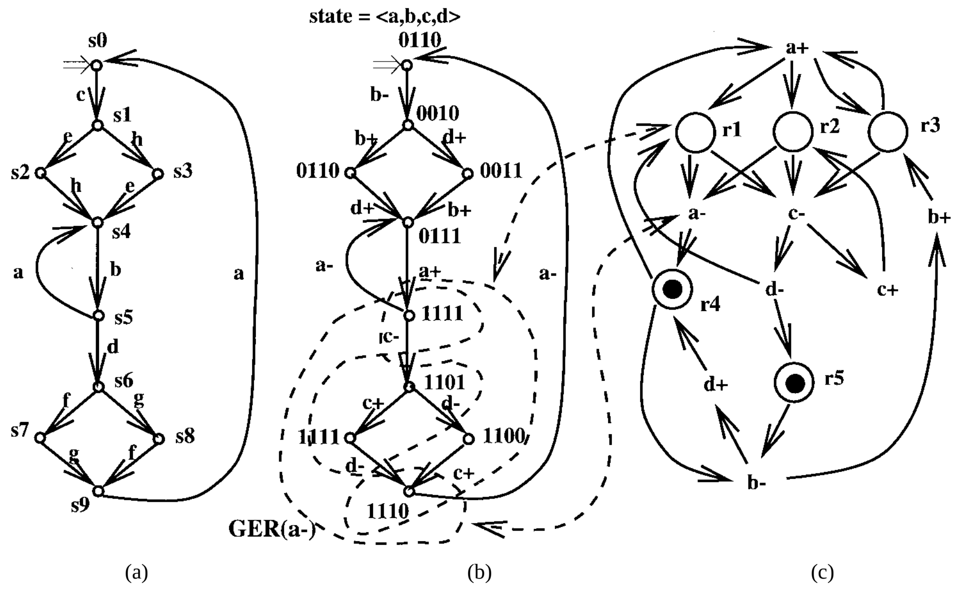

Fig. 2. Example of (a) transition system, (b) corresponding SG, and (c) STG.

A TS is called *deterministic* if, for each state s and each label a, there can be at most one state s' such that $s \xrightarrow{a} s'$. Otherwise, a TS is called *nondeterministic*. In the following, we are interested only in deterministic TS's. An example of a deterministic TS is shown in Fig. 2(a).

B. State Graph

For the purpose of logic synthesis, TS's must be binary encoded. A *state graph* SG is a binary encoded TS. An 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 such that $X_I \cap X_O = \emptyset$. Note that the output signals include both external output and internal signals.

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

Each event $e \in E$ in the SG is labeled with a *signal transition*. The labeling is given by an *event assignment function* λ_E: $E \mapsto X \times \{+, -\}$. Each signal transition can be represented as x_i+ or x_i- for the rising ($0 \to 1$) or falling ($1 \to 0$) 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 $a, b, \cdots$ instead of $x_1, x_2, \cdots$. 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 are satisfied for the assignment functions and every $(s, e, s') \in T$:

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. Fig. 2(b) shows a consistent SG which is obtained by binary encoding of the TS from Fig. 2(a). After binary encoding, for example, event c is mapped into signal transition $b-$ and state $s1$ is mapped into binary code $\langle a, b, c, d\rangle = 0010$.

C. Complete State Coding

In order to derive logic expressions for the output signals, the model's state assignment must be unambiguous with respect to those signals. This property is called *complete state coding (CSC)*.

An SG is said to satisfy the complete state coding requirement if, for any two states $s1$ and $s2$ 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 Fig. 2(b). States $s0$ and $s2$ have the same binary code 0110. Output signal b is enabled in $s1$ and is not enabled in $s2$; therefore, CSC is violated, and we say that states $s0$ and $s2$ are in *CSC conflict*. Although states $s5$ and $s7$ are also assigned the same binary code 1111, they are not in CSC conflict since no output signals are enabled in $s5$ and $s7$.

D. 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 [30] 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 of 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 especially widely used in many applications since they have simple verification algorithms [12] and simple semantics. A *labeled* PN is a PN with a labeling function λ: $T \longrightarrow \mathcal{A}$ which associates 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 labeled with signal transitions $(a+, a-, \cdots)$. 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 Fig. 2(b) is shown in Fig. 2(c).

III. Basics of the Theory of Regions

In this section, we will briefly review the theory of regions (more detail can be found in [11]), and will show how this theory allows us to perform transformations between TS's and PN's (hence, between SG's and STG's).

A. 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, one of the following conditions holds: all transitions labeled with e (1) exit r, or (2) enter r, or (3) do not cross r.

Let us consider the TS shown in Fig. 2(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 *preregion* of event e if there is a transition labeled with e which exits r. A region r is a *postregion* of event e if there is a transition labeled with e which enters r. The set of all preregions and postregions of e is denoted by $^\circ e$ and e°, respectively. By definition, it follows that if $r \in {}^\circ e$, then all transitions labeled with e exit r. Similarly, if $r \in e^\circ$, then all transitions labeled with e enter r. There are two preregions for event a: $r_1 = \{s_5, s_8, s_9\}$ and $r_2 = \{s_5, s_7, s_9\}$. Both of them are minimal regions since every subset of r_1 or r_2 is not a region.

The following propositions state a few important properties of regions [3], [10], [26].

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 $\overline{r} = S - r$ 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.

B. Excitation Regions

While regions in a TS are related to places in the corresponding PN, an excitation region [16] for event a is the largest 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 the *largest* (*largest 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 ER's for a. In the TS from Fig. 2(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 $GER(a) = \{s_5, s_9\}$.

C. Deriving Petri Nets from Transition Systems

The procedure to synthesize a PN from a 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 contain a token in the initial marking m_0 iff $s_{in} \in r_i$.
- The flow relation is constructed as follows: $a \in p_i \bullet$ iff r_i is a preregion of a and $a \in p_i$ iff r_i is a postregion of a.

Fig. 2(c) shows an STG derived for the SG of Fig. 2(b) followed this procedure. In particular, region r_i is mapped into place r_1 of the STG. As proved in [11], this procedure always produces a safe PN with an RG *bisimilar* to the initial TS if the following two conditions are satisfied.

• *Excitation closure:* For each event a: $\bigcup_{r \in {}^\circ a} r = GER(a)$.

• *Event effectiveness:* For each event a: $^\circ a \neq \emptyset$.

Bisimulation between two TS's corresponds to the equivalence of state transition graphs, which is traditionally used in automata minimization, and is formally defined as follows.

Definition 3.1 (Bisimulation [1]): Let $TS_1 = (S_1, E, T_1, s_{in_1})$ and $TS_2 = (S_2, E, T_2, s_{in_2})$ be two TS's with the same set of events. A bisimulation between TS_1 and TS_2 is a binary relation R between S_1 and S_2 such that

ia) for every $s_1 \in S_1$, there exists $s_2 \in S_2$ such that $s_1 R s_2$;

ib) for every $s_2 \in S_2$, there exists $s_1 \in S_1$ such that $s_1 R s_2$;

iia) for every $(s_1, e, s_1') \in T_1$ and for every $s_2 \in S_2$ such that $s_1 R s_2$, there exists $(s_2, e, s_2') \in T_2$ such that $s_1' R s_2'$;

iib) for every $(s_2, e, s_2') \in T_2$ and for every $s_1 \in S_1$ such that $s_1 R s_2$, there exists $(s_1, e, s_1') \in T_1$ such that $s_1' R s_2'$.

Intuitively, conditions ia) and iia) define a simulation of TS_1 by TS_2. Two TS's are said to be *bisimilar* if they can simulate each other, i.e., there exists a bisimulation between them. The relation "*is bisimilar to*" is an equivalence relation

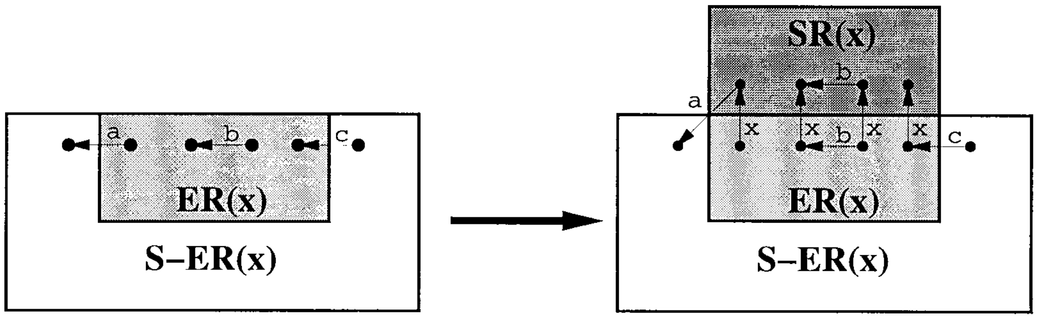

Fig. 3. Insertion of event x from $ER(x)$.

and partitions all TS's into equivalence classes. A TS is said to be *minimal* if no other element in its equivalence class has a set of states with smaller cardinality.

If the excitation closure and the event effectiveness conditions are satisfied, then the TS is said to be an *excitation closed*. If a TS is not excitation closed, then it is always possible to transform it to an excitation-closed form by label splitting (one label a which causes violations of excitation closure is substituted in the TS by a few independent labels $a_1, a_2, \cdots$) or by inserting silent transitions. Therefore, for any TS, an equivalent (in the sense of bisimulation between the TS and the RG) safe PN can be synthesized. Moreover, if the initial TS is excitation closed and minimal, then the synthesized PN will have an RG isomorphic to the original TS. The details of this technique are presented in [10] and [11].

IV. Constrained Tranformations of TS's

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

A. 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 [15], two properties ensure that a deterministic TS allows for a speed-independent implementation: *persistency* and *commutativity*. The persistency 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 Persistency): Let $A = (S, E, T)$ be a transition system. An event $a \in E$ is said to be *persistent* in $r \subseteq S$ iff: $\forall s1 \in r$: $[s1 \xrightarrow{a} \wedge (s1 \xrightarrow{b} s2) \in T] \Longrightarrow s2 \xrightarrow{a}$.

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 $s4 = s5$.

B. 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 the 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 $\{p \downarrow E_1 \colon p \in L(A)\}$.

Let $A' = (S', E', T')$ and $A = (S, E, T)$ be two TS's such that $E \subset E'$. Then, TS's A and A' are *trace equivalent* if $L(A') \downarrow E = L(A)$. Additional 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.

C. Event Insertion

The basic transformation is the *insertion of a single event* into a TS. There can be different schemes for event insertion that preserve trace equivalence [7]. In this paper, we will rely on a simple one which consists of two steps and is similar to [16], [38] and [41].

- 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 Fig. 3.
- 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 let $x \notin E$ 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:

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

The transformation of A to A' using Definition 4.3 splits each state $s \in r$ in S into two states s and s' in S'. All other

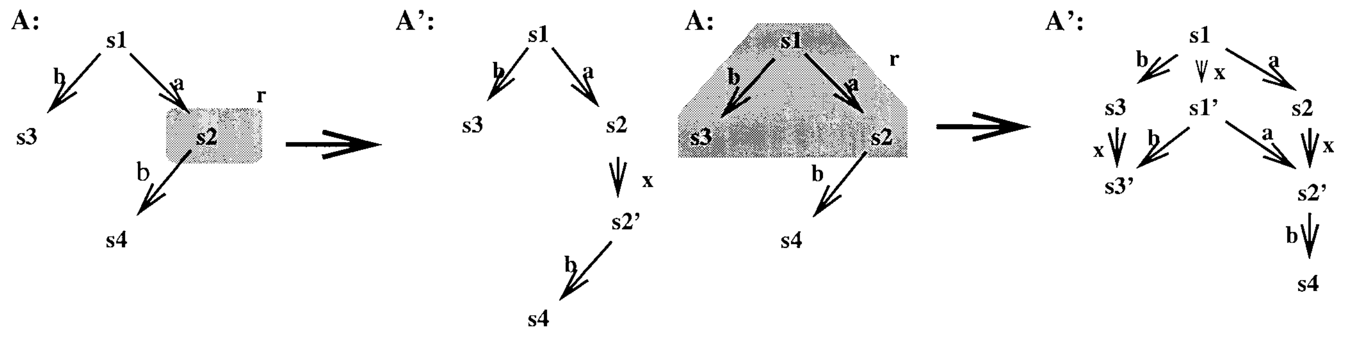

Fig. 4. Set of states r is not persistency preserving.

states $s \notin r$ have only one state in S'. Fig. 3 illustrates how event insertion is performed.

D. 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* [7]. Persistency and commutativity, on the other hand, are not automatically preserved, and need a more careful analysis.

Property 4.1: Let $A = (S, E, T)$ be a transition system, let $x \notin E$ be a new event, and let $r \subseteq S$. Let $A' = (S', E', T')$ be a transition system obtained after inserting x by r. Then

1) x is persistent in A',
2) x is commutative in A'.

Proof:

1) (By Contradiction) Assume that $s1\xrightarrow{x}$, $s1\xrightarrow{a}s2$, and x is not enabled in $s2$. Therefore, $s1 \in r$ and $s2 \notin r$ in A. By the definition of event insertion, no transition $s1\xrightarrow{a}s2$ should exist in A', which contradicts the initial assumption.
2) We need to prove that if $s1\xrightarrow{x}s1', s1'\xrightarrow{a}s3', s1\xrightarrow{a}s2$, and $s2\xrightarrow{x}s2'$, then $s2' = s3'$ in A'. $s1, s2 \in r$ in A since x is enabled in $s1$ and $s2$. Therefore, $s1'\xrightarrow{a}s2'$. Since A' is deterministic, then $s2' = s3'$. □

Definition 4.4 (SIP Set): Let $A = (S, E, T)$ be a transition system, let $x \notin E$ be a new event, and let $r \subseteq S$. 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 $A \Longrightarrow a$ is persistent in A',
2) A is commutative $\Longrightarrow A'$ 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.

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

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

Proof: $\Longleftarrow$ Two different cases of violations for condition (1) are possible.

1) $s2 \in r, s4 \notin r$, but $s1 \notin r$ [see Fig. 4(a)].
2) $s2 \in r, s4 \notin r$, but $s1, s3 \in r$ [see Fig. 4(b)].

Let us transform A to a new transition system A' by inserting event x by a set of states r (see Definition 4.3). In case 1), persistency is violated in the new transition system A' for state $s2$ because in $s1$, both events a and b are enabled but the firing of a disables b in state $s2$. In case 2), persistency is also violated in the new transition system A' because the firing of a disables b in $s2$. Thus, condition (1) is necessary for preserving event persistency.

$\Longrightarrow$ Let us call the *image* of $s \in S$ the set of states in S' that correspond to s, i.e., for $s \in S, s \in r$, the image of s is a set of two states $s, s' \in S'$, while for $s \in S, s \notin r$, the image of s is one state with the same name $s \in S'$.[1]

Suppose condition (1) is satisfied in A for the set r, but inserting event x by r leads to a violation of event persistency. It means that in A, there are states $s1$ and $s2$ such that $s1\xrightarrow{a}s2, s1\xrightarrow{b}$, and $s2\xrightarrow{b}$, but b is not persistent in state $s1_{A'}$ which is the image of $s1$. Note that by the rules of event insertion, persistency cannot be violated in A' for the states $s_{A'}, s'_{A'}$: $s_{A'}\xrightarrow{x}s'_{A'}$, which belong to the image of the same state $s \in A$.

When the states $s1$ and $s2$ which cause in A' the persistency violation are different, four cases are possible.

1) $s1, s2 \notin r$ [Fig. 5(a)];
2) $s1 \in r, s2 \notin r$ [Fig. 5(b)];
3) $s1 \notin r, s2 \in r$ [Fig. 5(c)];
4) $s1, s2 \in r$ with two subcases $s4 \notin r$ [Fig. 5(d)] and $s4 \in r$ [Fig. 5(e)].

For all cases in the SG A' obtained by A via the insertion of event x, persistency is preserved (see Fig. 5). □

Theorem 4.2: Let $A = (S, E, T)$ be a commutative transition system, and let $r \subset S$ be a persistency preserving set. Then r is an SIP set iff

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

[1]To distinguish s in A from its image s in A', we sometimes will refer to states in A' by adding a subscript: $s_{A'}$.

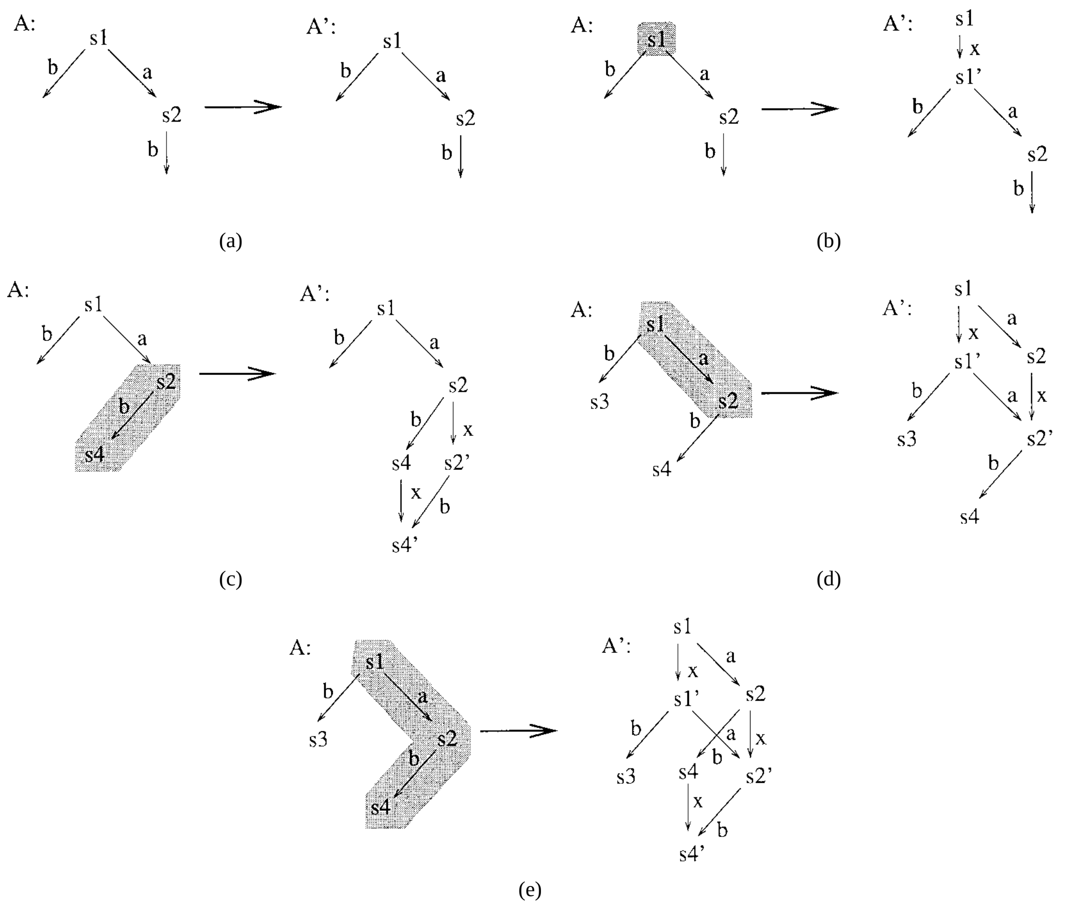

Fig. 5. Adding event x to transition system; cases 1)–4).

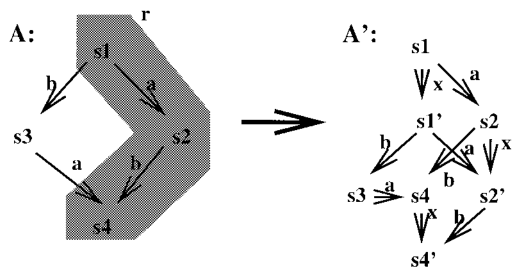

Fig. 6. Commutativity violation after signal insertion.

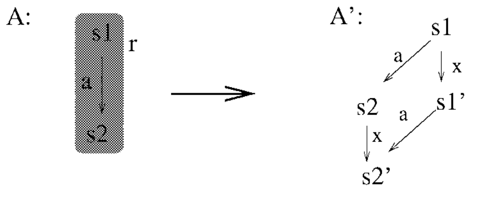

Fig. 7. Transformation of A into A'.

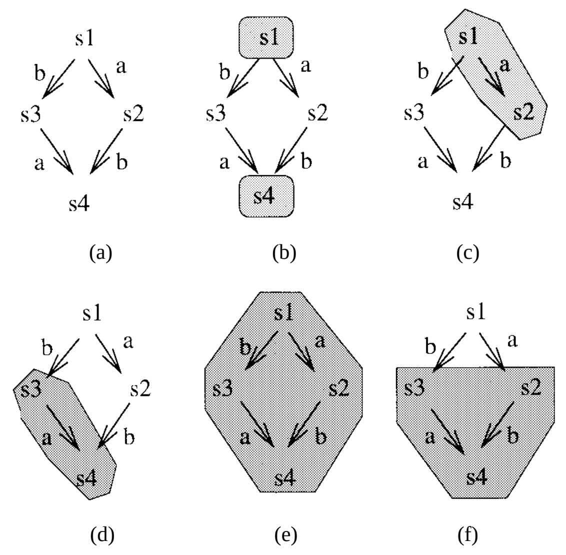

Fig. 8. SIP sets for a state diamond.

Proof: $\Longleftarrow$ Suppose condition (2) is violated. Transformation from A to A' in such case is illustrated by Fig. 6. Clearly, commutativity is violated for state $s1'$ because $s1' \xrightarrow{ab} s4'$, but $s1' \xrightarrow{ba} s4$, where $s4 \neq s4'$.

$\Longrightarrow$ Suppose conditions (1) and (2) are satisfied for a set of states $r \subset S$. Fig. 7 shows that commutativity cannot be violated in A' for events x and a, where x is the newly inserted event while $a \in E$.

Therefore, we need to consider commutativity only between the original events of A. The two different orderings of events a and b, each of which may fire in the same state in A, gives rise to the quadruple of states $s1, s2, s3, s4$ that we will call a *state diamond* [see Fig. 8(a)]. The major cases of legal intersection, according to conditions (1) and (2), of a diamond in A with a set of states r are shown in Fig. 8.

Cases $s1 \in r, s2, s3, s4 \notin r$ and $s4 \in r, s1, s2, s3 \notin r$ are covered by Fig. 8(b), while cases $s1, s3 \in r, s2, s4 \notin r$ and $s2, s4 \in r, s1, s3 \notin r$ are symmetrical to Fig. 8(c) and (d), respectively. All other cases of intersections are forbidden by conditions (1) and (2).

It is easy to check that, by applying the transformation rules to these five major cases (as was done in the proof of Theorem 4.1), we will never get violation of commutativity. □

Theorems 4.1 and 4.2 refine conditions for speed-independence from [35]. It allows us to handle correctly the so-called asymmetric "fake" conflicts between signals [19]. Consider, for example, Fig. 4(a). There is no arc between $s3$ and $s4$. On the other hand, SIP conditions were defined in [35] only with respect to complete diamonds of states. Hence, the conditions stated in [35] are insufficient to detect the violation of persistency in cases like the one shown in Fig. 4(a).

Fig. 4 shows two possible cases of violation of the persistency preserving condition (1) from Theorem 4.1. In both cases, event b becomes nonpersistent. 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 nonpersistencies can arise.

Fig. 6 shows a violation of commutativity when a set of states r does not meet condition (2) of Theorem 4.2. Fig. 8 shows all correct intersections of an SIP set with all state diamonds in an SG.

V. Selecting SIP Sets

This section presents several basic properties which allow us to formulate improved strategies for the selection of SIP sets. In [35] and [38], SIP sets are *selected* by solving a satisfiability problem. Constraining the search space for SIP sets efficiently appears to be problematic since the reduction to the satisfiability problem considers each state in SG individually (it is encoded by two binary signals). The latter quickly leads to unmanageable complexity when solving the satisfiability instance. In [16]–[18], SIP sets are *constructed* from excitation regions of the original signals and previously introduced state signals. Reference [41] generalized this method in such way that both ER's and switching regions (SR's) are used for SIP sets. In this paper, we further generalize this method: SIP sets are *constructed* as regions, intersections, and unions of intersections. We will show below that regions allow us to find valid SIP sets *automatically*, rather than checking for SIP *a posteriori*, which is considerably less efficient. Note that ER's and SR's are particular cases of region intersections. Therefore, our method explores a larger search space for SIP sets, and thus may yield more efficient solutions.

Property 5.1: If r is a region in a commutative transition system, then r is an SIP set.

The proof of this property is trivial. At the PN level, this property corresponds to the following structural transformation: place r is substituted by two places r and r' with a new intermediate transition labeled 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 an SIP set.

Proof: 1) Violation of condition (1) [Fig. 4(a), (b)]. In both cases, $s2 \in r, s4 \notin r$ and $s2 \xrightarrow{b} s4$. If $b \neq c$, then the firing of b disables c, thus contradicting the assumption of event persistency for c. If $b = c$, then $s1$ belongs to $ER(c)$ [a contradiction of Fig. 4(a)], and $s3$ must be out of $ER(c)$ [a contradiction with Fig. 4(b)].

2) Violation of condition (2) (Fig. 6). If $b \neq c$, then it follows from $s1 \in r$ and $s3 \notin r$ that the firing of b in state $s1$ disables c (a contradiction of persistency of c). If $b = c$, then $s4$ must be out of $ER(c)$ (a contradiction of Fig. 6). □

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 a delay at the gate output before its wire fork does not violate the semimodularity of the circuit [24]. Most of the previous methods for CSC used variations of Property 5.2 [6], [16], [22], [35], [41].

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:

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

Exit borders of regions and the intersection of preregions 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 excitation closed transition system, and let r be a region in A. If all of the exit events of r are persistent, then $EB(r)$ is an SIP set.

Proof: 1) Violations of condition (1) (Fig. 4).

If condition (1) for an 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 it follows from $s4 \notin EB(r)$ 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 of Fig. 4(a)], and $s3$ must be out of r [contradiction of Fig. 4(b)].

2) Violations of condition (2) (Fig. 6). By the same consideration, if $c \neq b$, then c becomes disabled in $s3$. If $c = b$, then $s4$ cannot be in $EB(r)$. □

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 σ and $\sigma 1$ belong to S.

Property 5.4: Let $A = (S, E, T)$ be a commutative excitation closed transition system, and let r_1, r_2 be preregions 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 an SIP set.

Proof: Assume that $A = (S, E, T)$ is a commutative excitation closed transition system. Assume also that r_1, r_2 are preregions 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 an 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 Fig. 4. Since $r_1, r_2 \in \bullet b$, all transitions labeled 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 preregions for b'. Transition $s2 \xrightarrow{b'} s4$ from condition (1) of Theorem 4.1 exits $r_1 \cap r_2$. Hence, $s2 \xrightarrow{b'} s4$ exits either r_1 or r_2. Let us assume, for example, that $s2 \xrightarrow{b'} s4$ exits r_1. Then transition $s1 \xrightarrow{b'} s3$ also exits r_1.

Let $s_2 \in GER(b)$. Since b is persistent and $s2 \xrightarrow{b'} s4$, state $s4 \in GER(b)$ and $s2 \xrightarrow{b'} s4$ is internal to $GER(b)$. In an excitation closed TS, the intersection of preregions for the same event b gives the excitation region for b. Hence, $s2 \xrightarrow{b'} s4$ 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) $\exists s' \in GER(b), s'' \in (r_1 \cap r_2) - GER(b)$: $s_2 \xrightarrow{\sigma} s'' \wedge s' \xrightarrow{\sigma 1} s''$.

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 $GER(b)$. 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.

Consider the second case. Since b is persistent, $r_1 \cap r_2$ is forward connected, and both r_1 and r_2 are preregions for b, then event $b \notin \sigma$. Therefore, state $s2 \in GER(b)$ and both b and b' are enabled in $s2$. Since b is persistent, we may conclude that transition $s2 \xrightarrow{b'}$ is internal for $GER(b)$, and hence is also internal for r_1. We again have reached a contradiction.

Consider the third case. Since b is persistent, $r_1 \cap r_2$ is forward connected, and both r_1 and r_2 are preregions 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 already has been considered.

Commutativity preservation: Let us refer to condition (2) from Theorem 4.2 and Fig. 6. Given a diamond $s1 \xrightarrow{a} s2, s2 \xrightarrow{b} s4, s1 \xrightarrow{b} s3, s3 \xrightarrow{a} s4$, the commutativity property may be violated only in one case: if state $s3 \notin r_1 \cap r_2$ and states $s1, s2, s4 \in r_1 \cap r_2$. In such a case, transition $s1 \xrightarrow{b} s3$ exits $r_1 \cap r_2$, and therefore must exit r_1 or r_2. Assume, for example, that $s1 \xrightarrow{b} s3$ exits r_1. On the other hand, transition $s2 \xrightarrow{b} s4$ 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. □

An important 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.

VI. 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 SG 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 TS 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 an I *partition*, is a partition of S into four blocks [37]: 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 $GER(x+)$ $(GER(x-))$.

Property 6.1: Let A be a consistent SG, and let $I = \langle S^0, S^1, S^+, S^- \rangle$ 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 $S^- \rightarrow S^+$.

Arcs like those shown in Fig. 9 are forbidden by Property 6.1. The proof of this property directly follows from the rules of insertion for events $x+$ and $x-$ (see Definition 4.3).

An I partition can be found in two steps.

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

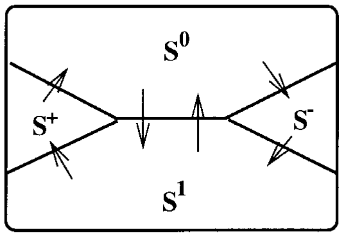

Fig. 9. Illegal transitions in an I partition.

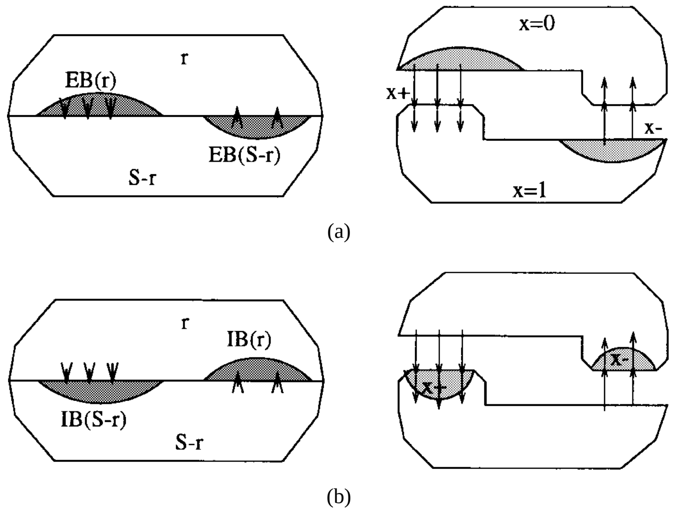

Fig. 10. Signal insertion by (a) exit and (b) input borders.

The boundaries may be defined in two ways: as exit borders or as input borders. Fig. 10(a) shows insertion by exit borders: given a bipartition $\{b, \overline{b}\}$, $GER(x+) = EB(b)$ and $GER(x-) = EB(\overline{b})$ (or vice versa). Fig. 10(b) illustrates insertion by input borders. In this case, $GER(x+) = IB(b)$ and $GER(x-) = IB(\overline{b})$ (or vice versa).

In general, using exit and input borders as insertion sets for new signal transitions does not 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 $\{b, \overline{b}\}$ be a bipartition of an SG state.

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

Let us refer to Fig. 9. If well-formed exit borders are chosen for inserting new signal transitions, then the I partition is defined as follows: $S^0 = b - EB(b), S^+ = EB(b), S^1 = \overline{b} - EB(\overline{b}), S^- = EB(\overline{b})$. Since a transition can exit b only through the $EB(b)$, no arcs $S^0 \to S^1$ and $S^0 \to S^-$ in Fig. 9 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 $S^+ \to S^0$ are not possible either. A similar reasoning holds for $EB(\overline{b})$; hence, none of the illegal transitions from Fig. 9 can occur.

Property 6.2: Let A be a consistent SG with a set of states S partitioned into $\{b, \overline{b}\}$. The SG A' obtained by inserting signal x by exit (input) borders of $\{b, \overline{b}\}$ is consistent iff these borders are well formed.

If the borders of a given partition $\{b, \overline{b}\}$ of S are not well formed, we can still use it by considering *larger* sets of states that guarantee consistency. A constructive way to do this is to start, e.g., from the exit border of set b, and to include all of the successors of every state violating the well formedness from b into the exit border of b. By doing this, the exit border of b monotonously increases until a unique well-formed set is found. Convergence is guaranteed because b is a well-formed exit border of b itself. Uniqueness is guaranteed because the expansion process does not allow any choice.

Given $\{b, \overline{b}\}$, we can define minimal well-formed *extended* EB and IB [denoted MWFEB(b) and MWFIB(b)] as minimal well-formed enlargements of exit and input borders, respectively. MWFEB(b) can be calculated as the least fixed point of the following recursion.

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

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

VII. 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 us to solve all CSC conflicts for a fairly general class of SG's.

Formally, completeness of the method for the class of excitation-closed TS is given by Theorem 7.2, which we will discuss shortly.

Intuitively, it can be explained as follows. A direct synthesis method for speed-independent implementation of STG's without choice has been proposed in [16]. It solves all CSC conflicts by construction. This method can be generalized to any safe STG [7] which is persistent with respect to the transitions of output signals (so-called *output-persistent* STG's). 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 II.

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 *excitation closed* after splitting all GER's into ER's. This result implies that, for each SG which meets these conditions, the procedure of signal insertions based on intersection of regions will eventually converge.

Theorem 7.1: Let SG A be: 1) deterministic, consistent, commutative, and persistent by noninput signals, and 2) excitation closed after splitting all GER's into ER's.

Then there is an SG A', trace equivalent to A, which has no CSC conflicts and is deterministic, consistent, commutative, and persistent by noninput signals.

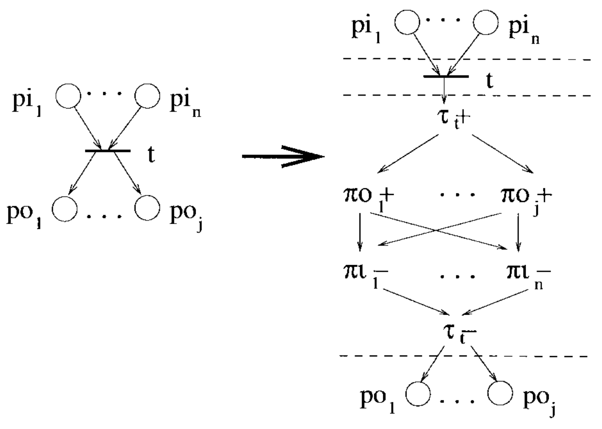

Fig. 11. Transformation for the transition t.

Proof: For any excitation-closed SG, there exists a corresponding safe Petri net without self-loops and with a reachability graph bisimilar to the SG [11]. In fact, this PN is an STG because its transitions are interpreted as the changes of binary signals. Let us denote an STG corresponding to SG A by D. We will eliminate all of the CSC conflicts by adding to the STG D two sets of binary signals.

1) $\pi_1, \cdots, \pi_k$, where k is the number of places in D.

The signals $\pi_1, \cdots, \pi_k$ encode every reachable marking m of D. If in a marking m place p_i has a token, it is encoded by signal $\pi_i = 1$; otherwise, $\pi_i = 0$. Modeling the change of marking under a transition firing, first the signals π_i for which $p_i \in t\bullet$ are set, and then the signals π_j for which $p_j \in \bullet t$ are reset.

2) $\tau_1, \cdots, \tau_l$, where l is the number of transitions in D.

The signals $\tau_1, \cdots, \tau_l$ model the enabling of transitions under the given marking m, i.e., if t_i is enabled under m, the corresponding signal τ_i is set to 1. After the firing of transition t_i, signal τ_i is reset to 0.

The transformation we will apply for the STG D is illustrated by Fig. 11. In Fig. 11, $pi_1, \cdots, pi_n$ and $po_1, \cdots, po_j$ denote the sets of input and output places of transition t.

Let us show that the STG $D1$ obtained after the transformation is: 1) trace equivalent to D, 2) deterministic, commutative, consistent, and persistent to noninput signals, and 3) without CSC conflicts.

1) Trace Equivalence: Trace equivalence is satisfied because, after projecting $D1$ on the set of original signals, we will get exactly STG D.

2) Consistency of State Assignment in D1: From trace equivalence, it follows that the consistency of state assignment with respect to the original signals is preserved in $D1$. For signals $\tau_1, \cdots, \tau_l$, the consistency of state assignment directly follows by construction.

Let us consider the consistency of signals $\pi_1, \cdots, \pi_k$. Signal π_i fires from 1 to 0 while modeling the firing of the transition for which p_i is an input place. According to the rules of transformation, at that moment, signal π_i is always at 1, and consistency cannot be violated for the falling transitions of π_i. It also cannot be violated for the rising transitions of π_i because π_i is set to 1 only in the markings where place p_i receives the token, and as STG D is safe, no new rising transition of π_i can happen before the falling transition will fire.

3) Determinism, Signal Persistency, and Commutativity: Determinism, commutativity, and signal persistency of the original signals of D is guaranteed in $D1$ by the trace equivalence between D and $D1$. Signal persistency of the added signals $\pi_1, \cdots, \pi_k$ and $\tau_1, \cdots, \tau_l$ follows from the fact that no input place of any transition of the added signals is shared by some other transition. Commutativity and determinism of the added signals are also satisfied.

4) Complete State Coding: Let us separate all of the markings of STG $D1$ into two sets:

a) *settled* markings, in which all signals $\tau_1, \cdots, \tau_l$ are equal to 0;

b) *transient* markings in which some of signals $\tau_1, \cdots, \tau_l$ are equal to 1.

Suppose the binary states s_1 and s_2 correspond to the reachable markings m_1 and m_2 of $D1$ and have the same code. We have the following cases.

1) m_1 and m_2 are settled markings, $m_0 \xrightarrow{q1} m_1, m_0 \xrightarrow{q2} m_2$. Let $q1'$ and $q2'$ be the transition sequences obtained by projecting $q1$ and $q2$ on the signals of D (i.e., $m'_0 \xrightarrow{q1'} m'_1, m'_0 \xrightarrow{q2'} m'_2$ in D). If $m'_1 = m'_2$, then the settled markings $m1$ and $m2$ in $D1$ are modeling the firing of the same transition t. Three possible cases of such markings are shown by the dashed lines in Fig. 11. All of them correspond to different binary states, and cannot be the sources of CSC conflicts.

2) m_1 is a settled marking, and m_2 is a transient marking. Then $s1$ and $s2$ are distinguished by signals $\tau_1, \cdots, \tau_l$.

3) m_1 and m_2 are both transient markings. For every transition t_i of D, let us consider the set of events in $D1$ that models the change of marking due to the firing of t_i, i.e., those events π_j+ and π_j- "in between" τ_i+ and τ_i- for which $p_j \in t_i\bullet$ and $p_r \in \bullet t_i$. Let us call this set the *transient set* of t_i, and denote it by $TrS(t_i)$. If the codes of s_1 and s_2 coincide, then the same signals $E = \{\pi_{i1}, \pi_{i2}, \cdots\}$ are equal to 1 in s_1 and s_2.

Transient sets of different transitions t_i and t_j ($\pi_i, \pi_j \in E$) cannot contain the same signals. Indeed, if $\pi_r+, \pi_r- \in TrS(t_i) \cap TrS(t_j)$, then two concurrent transitions t_i and t_j in D have the same output or input place p_r, and thus D is unsafe. From the nonintersection of different transient sets of concurrent transitions, it follows that s_1 and s_2 cannot have the same binary code.

Consideration of cases 1)–3) proves that all binary states of $D1$ have different codes that, in turn, ensure the satisfaction of the complete state coding property. □

Theorem 7.1 states that, for each SG which meets the above-listed conditions, the procedure of signal insertion based on the intersection of regions will eventually converge. Hence, one can always derive a speed-independent circuit that will implement this SG. However, it does not provide any realistic upper bound for the number of additional signals which are required to be inserted.

Let us take 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 $\{b, \overline{b}\}$ is a bipartition of a set

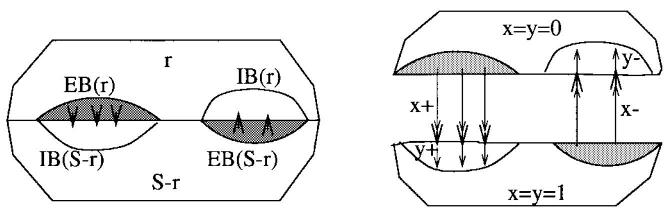

Fig. 12. Signal insertion using both exit and input borders.

of states. Assume that an I partition is constructed from $\{b, \overline{b}\}$ by exit borders, i.e., $S^+ = \text{MWFEB}(b)$, $S^- = \text{MWFEB}(\overline{b})$, and $S^0 = b - S^+$, $S^1 = \overline{b} - S^-$ [see Fig. 10(a)].

Clearly, all of 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 $\text{MWFEB}(b)$ and $\text{MWFEB}(\overline{b})$. Each state $s \in \text{MWFEB}(b)$ $[s \in \text{MWFEB}(\overline{b})]$ is mapped into two states s and s' in the new SG such that $s \overset{x*}{\rightarrow} s'$. Signal x has different values in s and s'. Thus, no CSC conflict in $\text{MWFEB}(b)$ and $\text{MWFEB}(\overline{b})$ is solved by x. Then we can use one more state signal y and another insertion scheme (by input borders) to distinguish these conflicts.

This method is illustrated in Fig. 12. At first, one additional state signal x is inserted by minimal well-formed exit borders of b and $\overline{b}$, and then another state signal y is inserted by minimal well-formed input borders of the partition $\{b', \overline{b'}\}$ inherited from $\{b, \overline{b}\}$ by a new SG. The only CSC conflicts which are not solved by such insertion of two signals are those which exist between $\text{MWFEB}(b)$ and $\text{MWFIB}(\overline{b})$, $\text{MWFEB}(\overline{b})$ and $\text{MWFIB}(b)$.

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

$$\left(s1 \in \text{MWFEB}(b) \wedge s2 \in \text{MWFIB}(\overline{b})\right) \\ \vee \left(s1 \in \text{MWFIB}(b) \wedge s2 \in \text{MWFEB}(\overline{b})\right).$$

The following theorem gives an upper bound on the number of state signals.

Theorem 7.2: Let $\{b, \overline{b}\}$ be a bipartition of an SG state. All distinguishable pairs of states $\{s1, s2\}$ $(s1 \in b, s2 \in \overline{b})$ will obtain different binary codes after inserting two state signals by MWFEB and MWFIB (the method from Fig. 12).

Proof: Let us insert state signal x by EB's of b and $\overline{b}$. Let A_x denote the resulting SG, and let S^1 and S^0 denote the sets of states in which x is equal to 1 and 0, respectively.

Any state $s1 \in b - \text{MWFEB}(b)$ has its image $s1$ in A_x such that $s1 \in S^0$. Any state $s2 \in b - \text{MWFEB}(\overline{b})$ has its image $s2$ in A_x such that $s2 \in S^1$.

Any state $s1 \in \text{MWFEB}(b)$ has its images $s1$ and $s1'$ $(s1 \overset{x+}{\rightarrow} s1')$ in A_x and $s1 \in S^0, s1' \in S^1$. Similarly, state $s2 \in \text{MWFEB}(\overline{b})$ has its images such that $s2 \in S^1, s2' \in S^0$.

Let us consider images b_x and $\overline{b_x}$ of b and $\overline{b}$ in A_x. Insertion of signal x simply delays firing of exit signals from b and $\overline{b}$, and thus exit signals of b_x by which we enter $\overline{b_x}$ are the same as for b. Then the image $s1_{A_x}$ of state $s1 \in \text{MWFIB}(b), s1 \notin \text{MWFEB}(b)$ will also belong to $\text{MWFIB}(b_x)$. For state $s1 \in \text{MWFIB}(b) \cap \text{MWFEB}(b)$, only one of its images, $s1_x$, belongs to $\text{MWFIB}(b_x)$, while the other, $s1'_x$, does not. Indeed, since no new arcs between b_x and $\overline{b_x}$ are created, no new states can be involved in their input borders. This means that, apart from preserving SIP conditions and well formedness, each state from $\text{MWFIB}(b)$ will have exactly one corresponding state in $\text{MWFIB}(b_x)$.

Consider insertion of state signal y by the $\text{MWFIB}(b_x)$ and $\text{MWFIB}(\overline{b_x})$. According to the values of state signals x and y, we can separate all states from A_{xy} in four sets $S^{00}, S^{10}, S^{01}, S^{11}$.

Let us examine all of the cases of state positions in TS A with respect to the borders of b and $\overline{b}$ (see Fig. 13).

1) $s \in b - (\text{MWFIB}(b) \cup \text{MWFEB}(b))$. It has only one image: $s_{A_{xy}}$: $s_{A_{xy}} \in S^{00}$ (see state $s1$ in Fig. 13).
2) $s \in \overline{b} - (\text{MWFIB}(\overline{b}) \cup \text{MWFEB}(\overline{b}))$. It has only one image: $s_{A_{xy}}$: $s_{A_{xy}} \in S^{11}$ (see state $s4$ in Fig. 13, for example).
3) $s \in \text{MWFEB}(b), s \notin \text{MWFIB}(b)$. It has images $s_{A_{xy}} \in S^{00}$ and $s'_{A_{xy}} \in S^{10}$ (see state $s2$ in Fig. 13).
4) $s \in \text{MWFEB}(\overline{b}), s \notin \text{MWFIB}(\overline{b})$. It has images $s_{A_{xy}} \in S^{11}$ and $s'_{A_{xy}} \in S^{01}$ (see state $s11$ in Fig. 13).
5) $s \in \text{MWFIB}(b), s \notin \text{MWFEB}(b)$. It has images $s_{A_{xy}} \in S^{00}$ and $s''_{A_{xy}} \in S^{01}$ (see state $s12$ in Fig. 13).
6) $s \in \text{MWFIB}(\overline{b}), s \notin \text{MWFEB}(\overline{b})$. It has images $s_{A_{xy}} \in S^{10}$ and $s''_{A_{xy}} \in S^{11}$ (see state $s3$ in Fig. 13).
7) $s \in \text{MWFEB}(b) \cap \text{MWFIB}(b)$. It has images $s_{A_{xy}} \in S^{01}$, $s''_{A_{xy}} \in S^{00}$ and $s'_{A_{xy}} \in S^{10}$ (see state $s5$ in Fig. 13).
8) $s \in \text{MWFEB}(\overline{b}) \cap \text{MWFIB}(\overline{b})$. It has images $s_{A_{xy}} \in S^{10}$, $s''_{A_{xy}} \in S^{11}$ and $s'_{A_{xy}} \in S^{01}$ (see state $s8$ in Fig. 13).

From the conditions of the theorem, it follows that $EB(b)$ and $IB(\overline{b})$ ($EB(\overline{b})$ and $IB(b)$) cannot contain states with the same binary codes. Hence, sets $S^{01} \cap b$ and $S^{01} \cap \overline{b}$ ($S^{10} \cap b$ and $S^{10} \cap \overline{b}$) have no mutual CSC conflicts.

Clearly, all states from $S^{00} \in b$ and $S^{11} \in \overline{b}$ are also distinguished by the value of signals x, y. Thus, all CSC conflicts between blocks b and $\overline{b}$ are resolved. □

It follows from the proof that if $s1$ and $s2$ are distinguishable by partition $\{b, \overline{b}\}$ in SG A, then the corresponding states in A' are distinguishable by partition $\{b', \overline{b'}\}$, where b' and $\overline{b'}$ are "images" of b and $\overline{b}$ in the new SG, A'. Hence, if all CSC conflicts can be distinguished by k bipartitions, then no more than $2k$ state signals are needed for solving all CSC conflicts in a SG.

This shows that our procedure for solving CSC conflicts is monotonous, and every next SG is better than the previous one in terms of CSC conflicts. Given an SG, a minimal number of partitions k which solve all CSC conflicts can be calculated. It gives an upper bound on the number of state signals which are necessary for CSC. Moreover, it is easy to show, by using the same direct synthesis method of [16], that the number of places of an STG or, equivalently, the cardinality of an irredundant cover of minimal regions of an SG gives a very loose *worst case* upper bound for the minimal number of partitions solving all CSC conflicts.

The following corollary from Theorem 7.2 states the conditions for implementability of an SG as a speed-independent circuit.

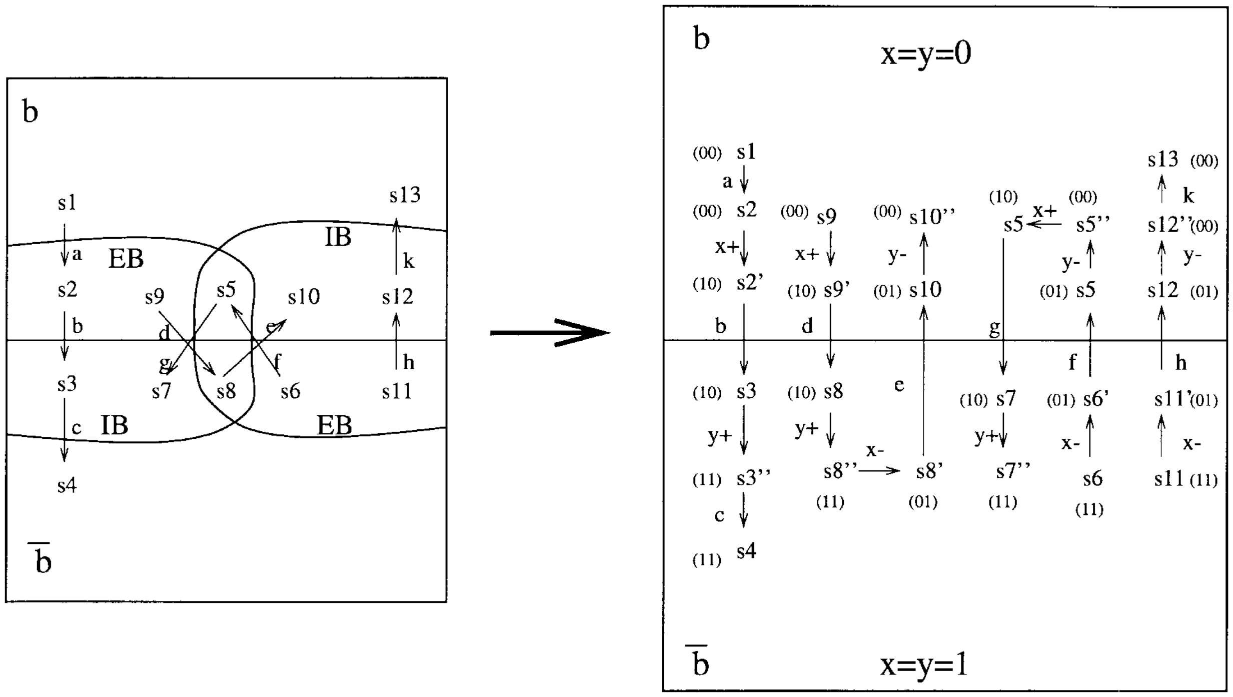

Fig. 13. Insertion of state signals by EB's and IB's.

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 $\{b, \overline{b}\}$ distinguishing $s1$ and $s2$ such that minimal well-formed extended exit and input borders of b and $\overline{b}$ 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 7.2, 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 [16] that, if an SG is excitation closed after splitting the GER into ER's, then all CSC conflicts are distinguished by some bipartition, and therefore the method based on EB and IB insertion can be successfully applied. For nonexcitation-closed SG's, 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.

VIII. A Heuristic-Search Strategy to Solve CSC

This section describes a strategy to solve CSC based on the theory of regions and insertion of events presented in the previous sections.

The main algorithm for the insertion of one state signal is as follows.

1) Generate a set of I partitions that preserve speed independence.
2) Estimate the cost of the generated I partitions.
3) Select the best I partition.
4) Increase the concurrency of the inserted signal

The following sections will describe how the set of configurations is explored, how the cost of an I partition is estimated, and how the concurrency of the inserted signal is increased.

A. Generation of I Partitions

As described in Section IV, the type of event insertions sought aims at preconditioning or postconditioning some of the existing events in the TS. Fig. 14 illustrates different types of insertions of the event x with regard to event a. Let us assume that t_1, t_2, and t_3 are concurrent (similarly for t_4, t_5, and t_6). The insertions of the figure can be obtained as follows:

$$\begin{aligned}\text{case (b): } \mathrm{ER}(x) &= \mathrm{MWFEB}(p_1) = \mathrm{MWFEB}(p_2)\\ &= \mathrm{MWFEB}(p_3)\\ \text{case (c): } \mathrm{ER}(x) &= p_1 \cap p_2\\ \text{case (d): } \mathrm{ER}(x) &= \mathrm{MWFEB}(p_4 \cap p_5 \cap p_6)\\ \text{case (e): } \mathrm{ER}(x) &= \mathrm{MWFEB}(p_4 \cap p_5).\end{aligned}$$

Cases (b), (d), and (e) can be obtained by defining ER(x) as the MWFEB of the intersection of a subset of the preregions or postregions of event a. Case (c) is the only one that cannot be obtained by using only MWFEB's (although it can be obtained using input borders; this option is not considered in the current implementation). However, the last step of the algorithm will allow us to cover this type of case by allowing us to enlarge the concurrency of the inserted event. It should be obvious that case (c) can be obtained from case (b) by allowing t_3 to be concurrent with x.

1) Exploring the Space of I Partitions: Each block of states defines an I partition. According to the type of insertions

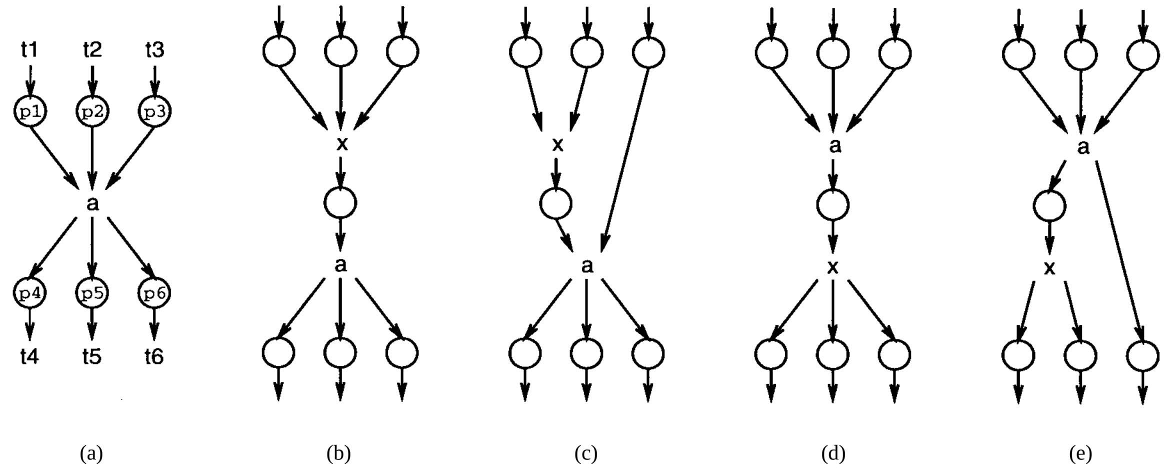

Fig. 14. Different types of event insertions.

```
bricks = calculate_all_bricks ()
frontier = good_blocks = {the best FW bricks}
repeat
    new_frontier = ∅
    for each bl ∈ frontier do
        for each br ∈ bricks adjacent to bl do
            new_bl = bl ∪ br
            if cost(new_bl) < cost(bl) then
                good_blocks = good_blocks ∪ new_bl
                new_frontier = new_frontier ∪ new_bl
    frontier = select the best FW blocks from new_frontier
until new_frontier = ∅
return the best block in good_blocks
```

Fig. 15. Heuristic search to find a block for event insertion.

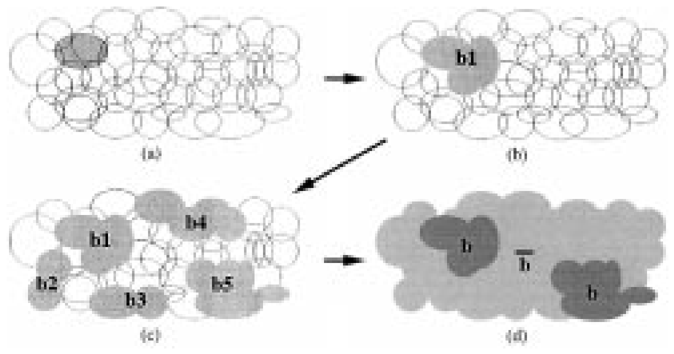

Fig. 16. (a) Brick, (b) block as the union of adjacent bricks, (c) connected blocks, and (d) final block after the union of disconnected blocks.

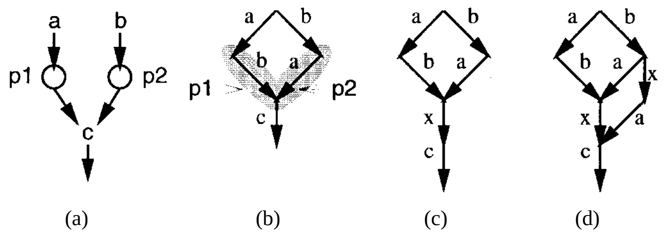

Fig. 17. (a) Petri net, (b) transition system, (c) insertion with ER$(x) = p_1 \cap p_2$, and (d) insertion with ER$(x) = p_2$.

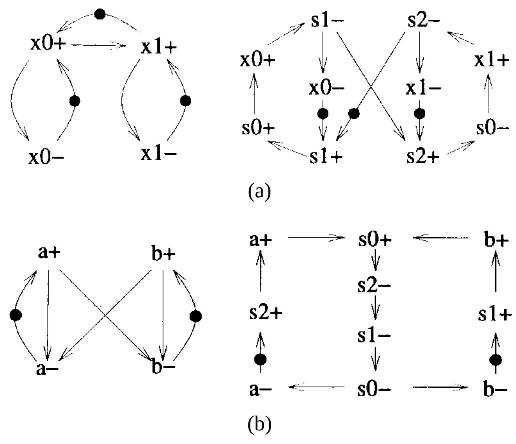

Fig. 18. Difficult examples to solve CSC.

described in Fig. 14, the EB of a block must be either a region or the intersection of some pre-/postregions of the same event. We consider these objects to be the "bricks" of the blocks, and we explore the space of blocks by calculating unions of bricks.

Fig. 15 presents an algorithm similar to the $\alpha - \beta$ pruning strategy commonly used in game-playing applications [20]. Initially, all bricks of the TS are calculated by: 1) obtaining all minimal regions of the TS minimal regions, and 2) calculating all possible intersections of pre-/postregions of the same event. Since the number of pre- and postregions of an event is usually small, an exhaustive generation is feasible.

The best block for event insertion is obtained as the union of adjacent bricks. At each iteration of the search, a frontier of FW (frontier width) "good" blocks is kept. Each block is enlarged by adjacent bricks, and the newly obtained blocks are considered candidates for the next iteration only if they are "better," according to the cost function, than their ancestors. FW is a parameter that can be tuned by the designer to define the degree of exploration of the configuration space, similar to defining the level of expertise of a chess-playing program, trading off the quality of the solution and the computational cost to find it. Finally, the best block generated during the search is chosen.

The execution of the previous algorithm would give a connected block of states as depicted in Fig. 16(b) (block b_1). In the most general case, a disconnected set of states may be appropriate to solve CSC. For this reason, the algorithm is iteratively executed with the rest of bricks of the TS (not intersecting with previously calculated blocks) until all states

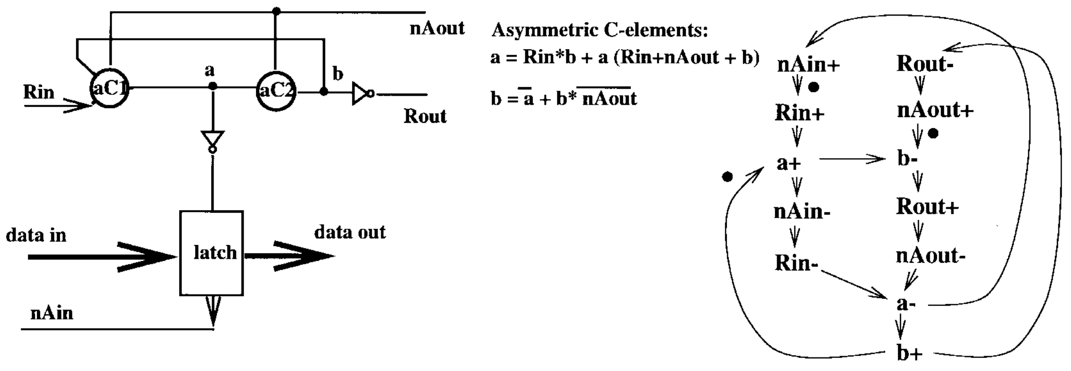

Fig. 19. AMULET2 pipeline.

with CSC conflicts have been covered by some block [e.g., blocks b_1–b_5 in Fig. 16(c)].

The final block for insertion is calculated as the union of disconnected blocks. A greedy block-merging approach guided by the cost function is used. In Fig. 16(d), a block $b = b_1 \cup b_5$ has been obtained.

2) Increasing Concurrency: Given a block b, S^+, and S^- are initially calculated as the MWFEB of b and $\overline{b}$, respectively. This leads to a solution with minimum concurrency of the inserted event. Concurrency can be increased by enlarging S^+ and/or S^-. This is illustrated in the example of Fig. 17. Let us assume that $b = p_1 \cup p_2$. In this case, MWFEB$(b) = p_1 \cap p_2$, which produces the event insertion of Fig. 17(c). By enlarging ER(x), e.g., ER$(x) = p_2$, the event x is made concurrent with event a [Fig. 17(d)].

In our approach, when the best configuration for event insertion has been calculated, S^+ and S^- are greedily enlarged by adding bricks that are adjacent to them. The enlargement is only accepted if the new configuration improves the cost of the solution.

IX. Cost Function

During the exploration of the space of configurations to solve CSC, a cost function is used to determine the candidates that must survive. This cost function is used in the construction of connected blocks, merging of disconnected blocks, and increase of concurrency.

The main objective of the cost function is to guide the search toward a correct and inexpensive solution of the CSC problem. Given that a great amount of configurations are explored, the cost function must not be computationally expensive.

Rather than deriving a function that yields a real number, the cost function is an algorithm that considers several implementation factors when comparing two different configurations. The following factors are considered for the insertion of signal x (in order of priority).

- ER$(x+)$ and ER$(x-)$ must be SIP blocks.
- The insertion of x must not modify the specification of the environment (e.g., x cannot be inserted before input events).
- The number of solved CSC conflicts must be maximized.
- The estimated complexity of the logic of the circuit must be minimized.

TABLE I
Results for STG's with a Large Number of States

benchmark	places	trans.	signals	states	CPU
amulet5-pipe	60	50	25/29	21330/30750	2822
master-read	37	26	13/19	8932/23322	449
master-read×2	74	52	26/38	$8.0 \times 10^7/5.4 \times 10^8$	5327
par8	43	36	18/26	$3.9 \times 10^5/1.7 \times 10^6$	608
par16	83	68	34/40	$1.5 \times 10^{11}/2.8 \times 10^{12}$	6814

In the evaluation of the last two factors, some degree of freedom is allowed. For example, if the relative difference of CSC conflicts disambiguated by two configurations is similar, the one with the cheapest circuit complexity is considered to be better.

With this approach, the most computationally expensive criterion (estimation of logic) is only evaluated when configurations are guaranteed to be correct and make tangible progress toward solving CSC.

A. Estimation of Logic

For each TS generated after the insertion of a new state signal, an approximate estimation of the complexity of the circuit is calculated.

The estimation is oriented toward a speed-independent realization based on monotonous covers [20], and is done as follows. For each ER of a noninput signal, a sum-of-products expression is calculated, and minimized with the don't-care set of the SG and the quiescent region that follows the ER (i.e., the set of states in which the signal remains stable after having been fired [20]). Even though the calculated cover must not necessarily be monotonous, our experiments have shown that it is monotonous in more than 80% of cases, and can be considered as a very approximate estimation of the complexity of a monotonous cover in the rest of the cases.

The complexity of the circuit is estimated as the sum of the number of literals of each cover.

X. Experimental Results

The region-based approach presented in this paper has been integrated in `petrify`, a tool for the synthesis of Petri nets. This section presents different experimental results that illustrate the main features of the approach. One of the main advantages of the tool is the possibility of retrieving an STG after having solved CSC. This allows the designer to analyze

TABLE II
EXPERIMENTAL RESULTS (CPU IN SECONDS)

		`assassin`		`petrify` (fast)			`petrify` (slow)	
benchmark	states	area/sig.	CPU	area/sig.	option	CPU	area/sig.	CPU
adfast	44	390/2	0.1	358/2		2.5	358/2	4.5
nak-pa	56	456/1	0.2	456/1		2.0	456/1	2.7
alloc-outbound	17	350/2	0.0	260/2		1.9	260/2	2.4
nowick	18	340/2	0.0	428/1		1.1	428/1	1.3
ram-read-sbuf	36	406/1	0.1	406/1		2.2	406/1	4.1
sbuf-ram-write	58	764/2	0.2	512/2	n	8.2	**406/2**	16.0
sbuf-read-ctl	15	244/1	0.0	244/1		0.4	244/1	0.6
mux2	99	1386/6	1.1	1616/5		225.1	1616/5	451.3
postoffice	58	1094/4	0.5	962/2		66.5	962/2	133.8
duplicator	20	294/2	0.0	294/2		1.1	294/2	1.7
spec_seq4	20	236/2	0.0	236/2	u	2.1	236/2	2.1
seq_mix	20	324/2	0.0	334/3	n	2.7	**324/2**	2.4
seq8	36	480/4	0.1	534/3	u	18.8	534/3	25.1
trcv-bm	44	826/3	0.2	782/2		21.4	782/2	28.1
tsend-bm	41	1010/3	0.3	946/2	n	20.6	**840/2**	41.5
ircv-bm	44	842/3	0.1	798/2		29.5	798/2	42.7
mod4_counter	16	648/2	0.0	626/3		1.5	626/3	0.8
master-read	1882	750/1	326.4	718/1		25.6	718/1	27.8
mmu	174	698/3	2.9	796/3		16.4	**716/3**	29.2
mr0	302	845/5	14.7	594/3	n	29.6	594/3	51.8
mr1	190	868/2	9.8	682/2		6.3	682/2	10.5
mmu0	174	886/3	2.7	756/3		24.7	756/3	44.4
mmu1	82	700/2	0.6	586/2		12.2	586/2	20.7
par_4	628	506/4	57.1	506/4		37.5	506/4	48.9
divider8	18	848/7	0.1	904/7	n	4.8	904/7	9.5
vme2int	74	1014/3	0.3	1022/3		47.9	**1014/3**	122.9
combuf2	11	270/2	0.1	262/2		0.9	262/2	1.1
Total		17475/74	417.6	16618/66		613.5	16308/65	1127.9

the obtained solutions, and choose the most convenient one by trading off several tuning parameters (e.g., increase the concurrency of the inserted signals with respect to other transitions, or reduce the estimated logic at the expense of increasing the number of state signals).

A. Difficult CSC Examples

We have used several benchmarks that no other automatic tool, such as `sis` [32] or `assassin` [42], has been able to solve. Some of them are even difficult to solve manually by expert designers. Our approach has succeeded in solving all of them. One of the major reasons for this qualitative improvement is that our approach can deal successfully with *secondary conflicts*, i.e., with cases in which any set of states required to separate conflicting states also contains some conflicting states. It is known that neither the approach of [22] nor the approach of [38] can solve those cases without major modifications or manual intervention. On the other hand, the approach of [41], whose completeness has never been shown, has not been able to solve them *in practice*.

Fig. 18 depicts two examples whose reachability graph has eight states each. Our region-based approach obtained the best solution of those that we could obtain manually (none of them with fewer than three state signals), while neither `sis` nor `assassin` was able to solve them.

Interestingly, the example in Fig. 18(b) is unsafe (the arcs $a+ \rightarrow b-$ and $b+ \rightarrow a-$ are 2-safe), but the resulting STG is safe. Nevertheless, the concurrency among the original signals of the specification is never reduced, i.e., the projection of the final reachability graph onto the signals of the environment is always equivalent to the original reachability graph.

B. Highly Concurrent Systems

One of the most important features of the CSC algorithm implemented in `petrify` is the capability of managing extremely large state graphs generated from STG's with high concurrency. Two factors are essential for this capability:

- the symbolic representation and manipulation of the state graph by means of *ordered binary decision diagrams* [29], and
- the exploration of blocks of states at the level of regions rather than states.

Fig. 19 shows the circuit of a pipeline cell proposed for the AMULET2 processor [13] and its STG specification. We obtained a specification of a five-cell pipeline by a parallel composition of five initial STG's. Then all internal signals b_i were excluded. This implied violations of the CSC property, and `petrify` produced a CSC solution. Neither `sis` nor `assassin` was able to complete the CSC solution for this specification.

Table I presents the results obtained for the AMULET2 pipeline and some examples with a vast state space. The master-read$\times$2 example has been artificially built from the parallel composition of two identical state graphs (by com-

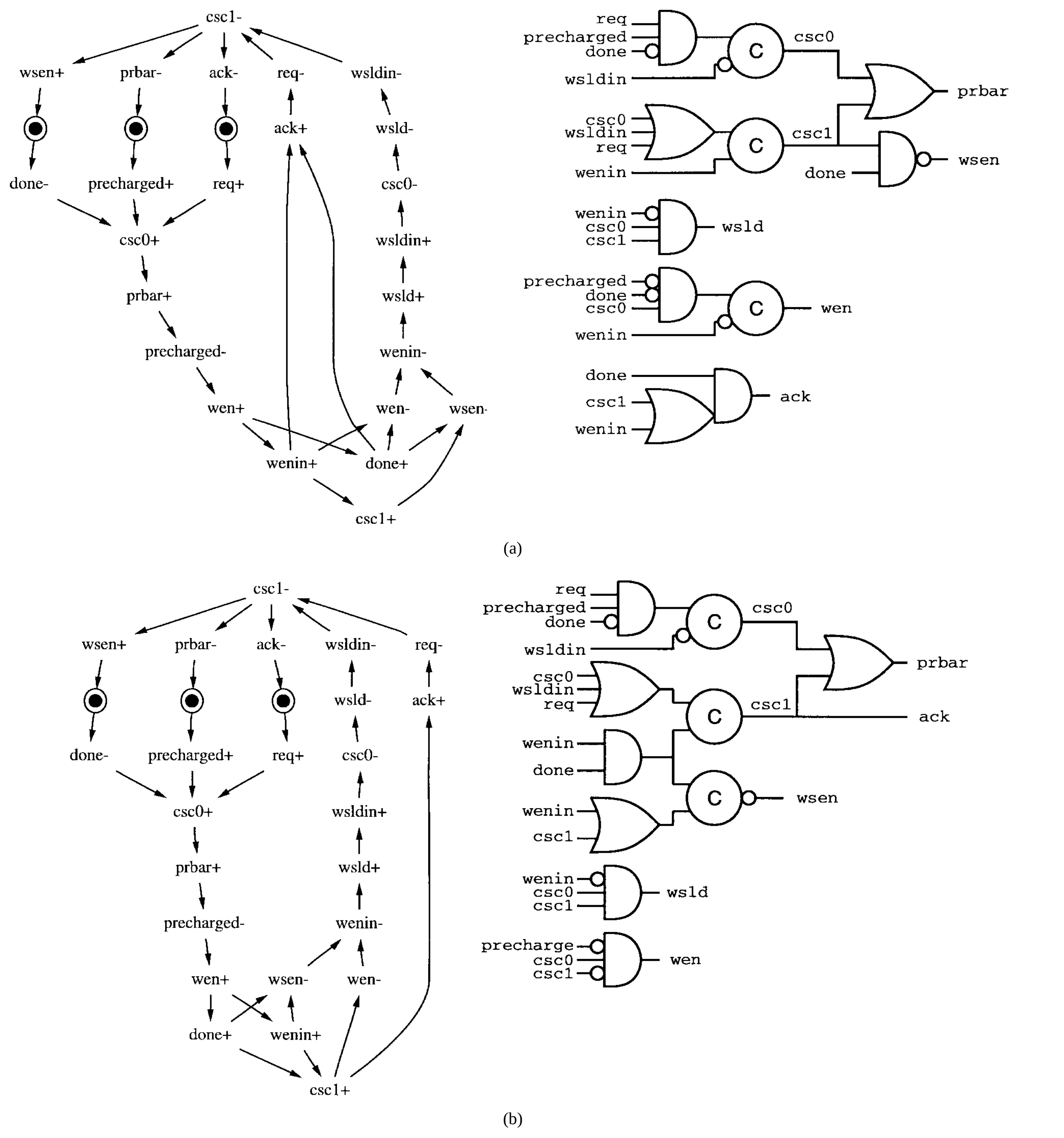

Fig. 20. The sbuf-ram-write example. (a) Solution with increased concurrency. (b) Solution with reduced concurrency.

posing two disconnected STG's). par8 and par16 are obtained by composing several parallelizer components used for the translation of the Tangram asynchronous circuit specification language [33].

Our approach is particularly effective in these cases because we use implicit methods for SG traversal [29]. It is well known that the memory and CPU requirements of such methods depend more on the problem structure than on the absolute number of states. Fortunately, most practical problems (e.g., most problems derived from the composition of small functional modules) behave well with implicit methods.

Table I describes the characteristics of each STG, as well as the number of signals and states before and after solving the CSC problem. Current tools exploring solutions at state level are not able to manage examples with more than 10^6 states.

C. Comparison with Other Approaches

Table II reports the results obtained with `petrify` in comparison with the ones presented in [42]. For each benchmark, the area estimated by `assassin` (using a library in which the minimum size inverter has an area of 16 units) and the number of inserted stated signals are reported.

The software tool `petrify` allows the designer to play with different options to seek solutions efficient in area, delay, or number of inserted signals. One of the options disables the capability of increasing the concurrency of the newly inserted signals (as illustrated in Fig. 17). In some cases, this results in slight area improvements at the expense of reducing the speed of the circuit.

Another interesting option consists of enabling/disabling the union of disconnected blocks to find a block for insertion [see Fig. 16(d)]. This may result in a reduction of the number of state signals, although not necessarily a reduction in the complexity of the circuit. Enabling the union of blocks is especially interesting in examples such as counters, where a logarithmic encoding of states can provide tangible area savings.

The default options used by `petrify` are: allow increase of concurrency and disable union of blocks. The column `option` in Table II indicates whether some different option has been used to reduce the area of the circuit (`n` when not allowing concurrency, and `u` when enabling union of blocks).

The columns with the heading `petrify (slow)` report the results obtained when increasing the frontier width (parameter FW in Fig. 15) of the heuristic search from one block (default) to five blocks. Those examples in which area improvements have been obtained are highlighted.

The quality of the results is comparable to those obtained by `assassin`, with an overall improvement of 7% in area. `assassin` may obtain a better result than our method in some cases because both *heuristically* estimate the complexity of the logic when deciding where and how to insert state signals. This means that the actual cost may be different from the estimate, or that (as in example `nowick`) more state signals may yield smaller logic. However, we claim that the contribution of our work mainly lies in the ability to substantially extend the class of specifications that can be successfully handled (in terms of both difficulty and size), rather than in the area decrease for very small examples.

The CPU time for a BDD-based approach is generally longer when the state space is small since a threshold overhead is always involved with the management of BDD data structures. However, when the state space is large (say, larger than 500 states), the CPU time of `petrify` is always shorter. It can be observed that the CPU 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 [29].

Fig. 20 depicts the graphical output generated by `petrify` after solving CSC for the sbuf-ram-write example. Two state signals have been inserted: *csc0* and *csc1*. Solution (a) presents more concurrency for the transition *csc1+* at the expense of some area increase. This example illustrates the possibilities offered to the designer to choose the most convenient solution for each application.

XI. Conclusions

We have presented a theoretical framework for insertion of new events into an asynchronous behavioral 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 labeled 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 the 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 [10]. The combination of the latter functionality with the algorithms for state-encoding event insertion allows one to solve CSC for large-scale 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] A. Arnold, *Finite Transition Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

[2] P. A. Beerel and T. H.-Y. Meng, “Automatic gate-level synthesis of speed-independent circuits,” in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992.

[3] L. Bernardinello, G. De Michelis, K. Petruni, and S. Vigna, “On synchronic structure of transition systems,” Tech. Rep., Univ. Milano, Milano, Italy, 1994.

[4] S. Burns and A. Martin, “A synthesis method for self-timed VLSI circuits,” in *Proc. Int. Conf. Computer Design*, 1987.

[5] T.-A. Chu, “On the models for designing VLSI asynchronous digital systems,” *Integration: VLSI J.*, vol. 4, pp. 99–113, 1986.

[6] ——, “Synthesis of self-timed VLSI circuits from graph-theoretic specifications,” Ph.D. dissertation, Mass. Inst. Technol., Cambridge, June 1987.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, “A region-based theory for state assignment in asynchronous circuits,” Tech. Rep. 95-2-006, Univ. Aizu, Japan, Oct. 1995.

[8] ——, “Complete state encoding based on the theory of regions,” in *Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 1996, pp. 36–47.

[9] ——, “Methodology and tools for state encoding in asynchronous circuit synthesis,” in *Proc. Design Automation Conf.*, June 1996, pp. 63–66.

[10] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, “Synthesizing Petri nets from state-based models,” in *Proc. ICCAD’95*, Nov. 1995, pp. 164–171.

[11] ——, “Deriving Petri nets from finite transition systems,” Tech. Rep. UPC-DAC-1996-19, Dept. Comput. Architecture, Univ. Politècnica Catalunya, Spain, June 1996.

[12] J. Esparza and M. Nielsen, “Decidability issues for Petri nets,” *Petri Nets Newsletter*, vol. 94, pp. 5–23, 1994.

[13] S. B. Furber and P. Da, “Four-phase micropipeline latch control circuits,” *IEEE Trans. VLSI Syst.*, vol. 4, pp. 247–253, June 1996.

[14] J. Gu and R. Puri, “Asynchronous circuit synthesis with Boolean satisfiability,” *IEEE Trans. Computer-Aided Design*, vol. 14, Aug. 1995.

[15] R. M. Keller, “A fundamental theorem of asynchronous parallel computation,” *Lecture Notes in Comput. Sci.*, vol. 24, pp. 103–112, 1975.

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

[17] M. A. Kishinevsky, A. Y. Kondratyev, and A. R. Taubin, “Formal method for self-timed design,” in *Proc. European Design Automation Conf. (EDAC)*, 1991.

[18] A. Kondratyev, “Design of self-timed circuits from change diagrams,” (in Russian), Ph.D. dissertation, LETI Leningrad, 1987.

[19] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig, and

A. Yakovlev, "Checking signal transition graph implementability by symbolic BDD traversal," in *Proc. European Design and Test Conf. (ED&TC)*, Paris, France, Mar. 1995, pp. 325–332.

[20] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, "Basic gate implementation of speed-independent circuits," in *Proc. Design Automation Conf.*, June 1994, pp. 56–62.

[21] A. Y. Kondratyev, L. Y. Rosenblum, and A. V. Yakovlev, "Signal graphs: A model for designing concurrent logic," in *Proc. 1988 Int. Conf. Parallel Processing*, Pennsylvania State Univ. Press, 1988.

[22] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits.* Norwell, MA: Kluwer 1993.

[23] K.-J. Lin and C.-S. Lin, "On the verification of state-coding in STG's," in *Proc. ICCAD'92*, Santa Clara, CA, Nov. 1992.

[24] D. E. Muller and W. C. Bartky, "A theory of asynchronous circuits," in *Ann. Computing Lab. Harvard Univ.*, 1959, pp. 204–243.

[25] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[26] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan, "Elementary transition systems," *Theor. Comput. Sci.*, vol. 96, pp. 3–33, 1992.

[27] S. M. Nowick and D. L. Dill, "Automatic synthesis of locally-clocked asynchronous state machines," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991.

[28] E. Pastor and J. Cortadella, "An efficient unique state coding algorithm for signal transition graphs," in *Proc. Int. Conf. Computer Design*, Oct. 1993.

[29] E. Pastor, O. Roig, J. Cortadella, and R. Badia, "Petri net analysis using Boolean manipulation," in *15th Int. Conf. Appli. Theory of Petri Nets*, Zaragoza, Spain, June 1994.

[30] C. A. Petri, "Kommunikation mit Automaten," Ph.D. dissertation, Institut für Instrumentelle Mathematik, Bonn, 1962 (Tech. Rep. Schriften des IIM 3).

[31] L. Y. Rosenblum and A. V. Yakovlev, "Signal graphs: From self-timed to timed ones," in *Int. Workshop Timed Petri Nets*, Torino, Italy, 1985.

[32] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Tech. Rep. UCB/ERL M92/41, Univ. California, Berkeley, May 1992.

[33] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*. Cambridge: Int. Ser. Parallel Computing, Cambridge Univ. Press, 1993.

[34] P. Vanbekbergen, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1990, pp. 184–187.

[35] ——, "Synthesis of asynchronous controllers from graph-theoretic specifications," Ph.D. dissertation, Katholieke Univ. Leuven, IMEC, Belgium, Sept. 1993.

[36] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1426–1438, Nov. 1992.

[37] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man, "A generalized state assignment theory for transformations on signal transition graphs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 112–117.

[38] ——, "A generalized state assigment theory for transformation on signal transition graphs," *J. VLSI Signal Processing*, vol. 7, no. 1–2, pp. 101–116, 1994.

[39] A. V. Yakovlev and A. Petrov, "Petri nets and parallel bus controller design," in *Int. Conf. Appl. Theory of Petri Nets*, Paris, France, IEEE Computer Society, June 1990.

[40] C. Ykman-Couvreur and B. Lin, "Efficient state assignment framework for asynchronous circuit synthesis," in *Proc. Int. Conf. Computer Design*, Oct. 1995.

[41] ——, "Optimized state assignment for asynchronous circuit synthesis," in *2nd Working Conf. Asynchronous Design Methodologies*, May 1995, pp. 118–127.

[42] C. Ykman-Couvreur, B. Lin, and H. De Man, "ASSASSIN: A synthesis system for asynchronous control circuits," Tech. Rep., IMEC, User and Tutorial Manual, Sept. 1994.

[43] K. Y. Yun and D. L. Dill, "Automatic synthesis of 3D asynchronous state machines," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992.

Jordi Cortadella (S'87–M'88) received the M.S. and Ph.D. degrees in computer science from the Polytechnic University of Catalonia, Barcelona, Spain, in 1985 and 1987.

He is an Associate Professor at the Department of Computer Architecture, Polytechnic University of Catalonia. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits, computer arithmetic, and parallel architectures. He has coauthored more than 50 research papers in technical journals and conferences.

Michael Kishinevsky (M'95–SM'96), photograph and biography not available at the time of publication.

Alex Kondratyev (M'94–SM'97), photograph and biography not available at the time of publication.

Luciano Lavagno (S'88–M'93) graduated magna cum laude in electrical engineering from Politecnico di Torino, Italy, in 1983. In 1992 he received the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley.

From 1984 to 1988 he was with CSELT Laboratories, Torino, Italy, where he was involved in an ESPRIT project that developed a complete high-level synthesis system. In 1988 he joined the Department of Electrical Engineering and Computer Science of the University of California at Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. He is currently an Assistant Professor at the Politecnico di Torino, Italy, and a research scientist at Cadence Berkeley Laboratories. He has also been a consultant for various EDA companies, such as Synopsys and Cadence. His research interests include the synthesis of asynchronous and low-power circuits, the concurrent design of mixed hardware and software systems, and the formal verification of digital systems. He is the author of a book on asynchronous circuit design and has published over 60 journal and conference papers.

In 1991 Dr. Lavagno received the Best Paper award at the 28th Design Automation Conference in San Francisco, CA. He has served on the technical committees of several international conferences in his field (namely the Design Automation Conference, the International Conference on Computer Aided Design, and the European Design Automation Conference).

Alexandre Yakovlev received the M.Sc. and Ph.D. degrees in computing science from Electrotechnical University of St. Petersburg, Russia.

At the Electrotechnical University, he worked in the area of asynchronous and concurrent systems since 1980, and in the period between 1982 and 1990 held positions of Assistant and Associate Professor at the Computing Science Department. He first visited Newcastle in 1984–1985 for research in VLSI and design automation. After coming back to Britain in 1990 he worked for one year at the Politechnic of Wales (now University of Glamorgan). Since 1991 he has been a Lecturer, and quite recently a Reader in computing systems design, at the Newcastle University Department of Computing Science, where he is heading the VLSI Design research group. His current interests and publications are in the field of modelling and design of asynchronous, concurrent, real-time and dependable systems.