Structural Methods for the Synthesis of Speed-Independent Circuits

Enric Pastor* Jordi Cortadella* Alex Kondratyev† Oriol Roig*

* Department of Computer Architecture
Universitat Politècnica de Catalunya
08071 Barcelona, Spain

† Computer Architecture Laboratory
The university of Aizu
Aizu-Wakamatsu, 965 Japan

Abstract

Most existing tools for the synthesis of asynchronous circuits from Signal Transition Graphs (STGs) derive the reachability graph for the calculation of logic equations. This paper presents novel methods exclusively based on the structural analysis of the underlying Petri net. This methodology can be applied to any STG that can be covered by State Machines and, in particular, to all live and safe free-choice STGs. Significant improvements with regard to existing structural methods are provided. The new techniques have been implemented in an experimental tool that has been able to synthesize specifications with over 10^{27} markings, some of them being non-free choice.

1 Introduction

Petri nets (PNs) are a powerful formalism to model concurrent systems. As a model, their most interesting feature is the capability of implicitly describing a vast state space by a succinct representation, which gracefully captures the notions of causality, concurrency and conflict between events. Petri nets have also been chosen by many authors as a formalism to describe the behavior of asynchronous circuits by interpreting the events as signal transitions coining the term *Signal Transition Graph* (STG) [14, 3].

Each reachable marking of an STG is assigned a binary code with the value of the circuit signals in that marking. Deriving logic equations from an STG requires the generation of the binary codes for all markings. Currently, most synthesis tools [15, 16] perform an exhaustive token flow analysis to obtain the complete reachability graph of the PN and all binary codes.

Even for small STGs, the reachability graph of highly concurrent systems can be extremely large and, in the worst case, exponential in the size of the STG. Some efforts have been devoted to propose structural methods for synthesis [10, 17], but they have been usually devised for restricted classes of PNs that compromise the potential expressiveness of this formalism.

In this paper we present structural techniques to synthesize *speed-independent* (SI) circuits from STGs. In this framework, "structural" means "at Petri net level" without requiring the explicit generation of the reachability graph. The techniques have polynomial complexity if the underlying PN is free choice [6, 4], and can be efficiently extended to the class of PNs that can be covered by State Machines (SM) [5]. We aim at complementing the existing tools by providing alternative and efficient synthesis methods for SM-coverable STGs, which account for a large fraction of the set of benchmarks we have used.

This work improves the techniques presented in [11, 12] in two directions: (1) by providing new methods to calculate covers for the boolean functions needed for synthesis and (2) by extending these methods to the synthesis of SI-circuits.

2 Overview

Let us assume that we want to derive a logic function for signal y in Fig.1(a) and, in particular, for the set of markings in which transition y+ is enabled—excitation region of y+ (ER(y+)). This region corresponds to all the markings in which place p_5 is marked, and it has been shadowed in the corresponding reachability graph (see Fig.1(b)).

By a simple structural analysis that takes polynomial time [4] we can deduce that the STG has an underlying *free-choice* PN. We can also derive a subset of SMs that cover the net. In this case, two SMs can be obtained, namely, the sets of nodes $SM_1 = \{p_1, x+, p_2, z+, p_3, x-, p_4, z-, p_7, y-\}$ and $SM_2 = \{p_1, x+, p_5, y+, p_6, z-, p_7, y-\}$.

Our purpose is to calculate a set of cubes that safely[1] cover the binary codes in ER(y+). A single cube approximation of the cover of a place can be calculated as follows: if a signal transition can fire while the place remains marked, then the value of the signal is unknown. Since transitions z+ and x− can fire when p_5 is marked, then the value of x and z is unknown in p_5. On the contrary, the value of y can be exactly determined by analyzing the structure of SM_2 and the ordering relation of y+ and y− with p_5. Thus, the cube (-0-) can be derived for p_5. For places p_1 and p_7, the cubes can be exactly calculated by analyzing the ordering relations of the places in SM_1 and SM_2 (see Fig.1(a)).

However, we can easily detect that this cube is an overestimation of ER(y+). Marking (000) is covered by (-0-), but also by the cube of p_1. Using these covers for synthesis would derive an erroneous circuit in which y+ would be enabled in marking (000) also. To overcome this situation we propose two strategies:

1. To refine the place covers by analyzing the concurrent relations with the places of other SMs. In our case, we use the fact that p_5 can only be simultaneously marked with p_2, p_3 or p_4, to obtain a multi-cube cover by intersecting the cover of p_5 with the conjunction of the covers of p_2, p_3 and p_4. Even though several refinements may be needed, this refinement is sufficient to guarantee a correct synthesis (Fig.1(c)).
2. To insert internal signals, similarly as it is done to solve encoding conflicts, disambiguating covers whose intersection produces contradictions for synthesis. As an example, signal v distinguishes the covers of p_1 and p_5 in Fig.1(d).

In general, both methods can be combined to obtain a correct set of covers. In this paper, we present the conditions under which a set of covers can be safely used for synthesis.

To give the idea about the efficiency of the structural approach let us consider two illustrative examples.

Figure 2 presents an autonomous circuit with a C-element closed on its inputs through inverters. A C-element is the basic cell used for the synchronization of processes in asynchronous designs. Its output rises when all its inputs are in "1" and falls when on all inputs are "0", in any other case it remains unchanged. The logic function for a C-element is: $a = x_1 \ldots x_n + a(x_1 + \ldots + x_n)$. In Fig.2 a change on the output of the C-element leads to a burst of input changes. As all inputs are switching concurrently the number of markings in a n-input circuit is 2^{n+1}, while the number of places in the corresponding STG (Fig.2(b)) is only $4n$.

[1] only overestimations that intersect with the *don't care* set are allowed.

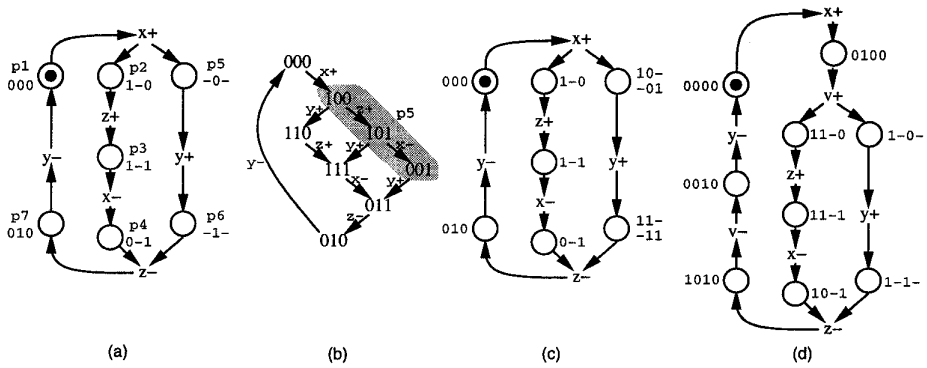

Figure 1: (a) STG and covering cubes for places, (b) reachability graph, (c) refined covers, (d) internal signal insertion.

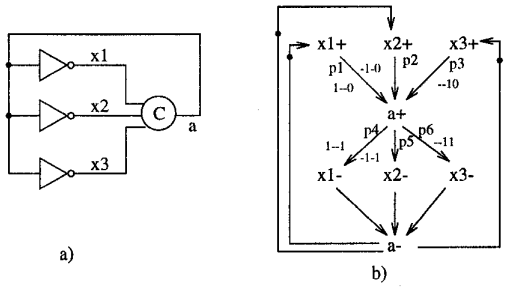

Figure 2: (a)C-element circuit, and (b)its STG.

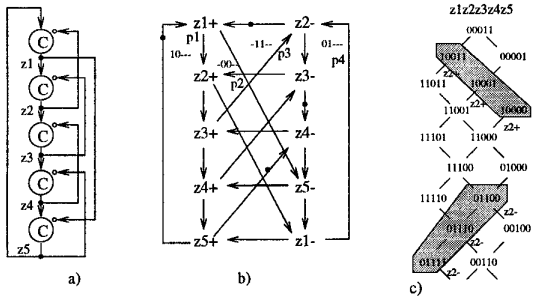

Figure 3: (a)Muller pipeline, (b)STG, and (c)reachability graph.

The use of cover cubes for places is very efficient here because they define the excitation regions for all signal transitions *exactly*. Indeed, given transition x1– *e.g.*, the cube (1--1) of its predecessor place $p4$ (Fig.2(b)) is an exact cover for ER(x1–). For the output transitions exact covers can be also computed. Given transition a+ *e.g.*, its predecessor places p_1, p_2 and p_3, and the intersection of their cubes gives the single marking (1110) where a+ is enabled ((1--0) · (-1-0) · (--10) = (1110)).

Another example is a Muller pipeline of C-elements (see Fig.3(a), where a five cell pipeline is presented). Each cell can be set either to 1 or to 0. State "1" is a *working* state and shows that the cell contains information. State "0" is an *idle* state showing that the cell does not contain useful information. The rules for the pipeline functioning are the following:

- cell i goes to a working state if cell $(i-1)$ is in a working state and cell $(i+1)$ is in an idle state,
- cell i goes to an idle state if cell $(i-1)$ is in an idle state and cell $(i+1)$ is in a working state.

The STG and a piece of its reachability graph for the five cell pipeline are presented in Fig.3(b)(c).

Suppose we would like to obtain the logic function for signal z2. From the cube approximation ER(z2+) can be obtained from the intersection of cubes for places p_1 and p_2, while ER(z2–) from the intersection of cubes for places p_3 and p_4. This gives the following covers: ER(z2+) = (10---) · (-00--) = (100--) and ER(z2–)=(-11--)·(01---)=(011--). These covers are safe because they cover the ER(z2+) and ER(z2–) (shadowed in Fig.3(c)) plus the vertices (10010) and (01101) —known to be in the *dc-set*.

In both considered examples we got the functions for signals from the structural information in the STG rather than by restoration of its reachability graph. These examples are illustrative ones, not always the function derivation procedure will be so simple. However they allow to catch a general view of complexity reduction while using the cover cube approximations.

In the rest of the paper we describe how the aforementioned techniques can be applied in case the covers do not fulfill the synthesis conditions. We will remark the fact that the reachability graph of the specification is never generated during the synthesis of the circuit. The proposed techniques have been implemented in an experimental tool. We will finally report the results obtained from a large set of benchmarks.

3 Definitions

A *Petri net* (PN) is a 4-tuple $\Sigma = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_o \rangle$, where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the flow relation, and M_o is the initial marking. Given a node $x \in \mathcal{P} \cup \mathcal{T}$, its post-set and pre-set are denoted by $x^\bullet$ and ${}^\bullet x$ respectively. A *path* of a PN is a sequence $x_1 \ldots x_r$ of nodes such that $\forall i,\ 1 \leq i < r : (x_i, x_{i+1}) \in \mathcal{F}$. A path is called *simple* if no node appears on it more than once. A *State Machine* (SM) is a PN such that each transition has exactly one input place and one output place. A *Free-choice net* (FC net) is a PN such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition.

A transition t is *enabled* in a marking M, denoted by $M[t\rangle$, when all places in ${}^\bullet t$ are marked. An enabled transition in M *fires*, removing a token from places in ${}^\bullet t$ and adding a token to places in $t^\bullet$, reaching a new marking M' ($M[t\rangle M'$). A marking M is *reachable* from M_o if there is a sequence of *firings* $t_1 t_2 \ldots t_n$ that transforms M_o into M ($M_o[t_1 t_2 \ldots t_n\rangle M$), hence $t_1 t_2 \ldots t_n$ is a *feasible* sequence. The set of reachable markings from M_o is denoted by $[M_o\rangle$. A PN is *live* if every transition can be infinitely enabled through some *feasible* sequence of *firings* from any marking in $[M_o\rangle$. A PN is *safe* if no marking in $[M_o\rangle$ can assign more than one token to any place.

A *live* and *safe* FC net can be decomposed into a potentially exponential number of strongly-connected SM-Components (SMs)

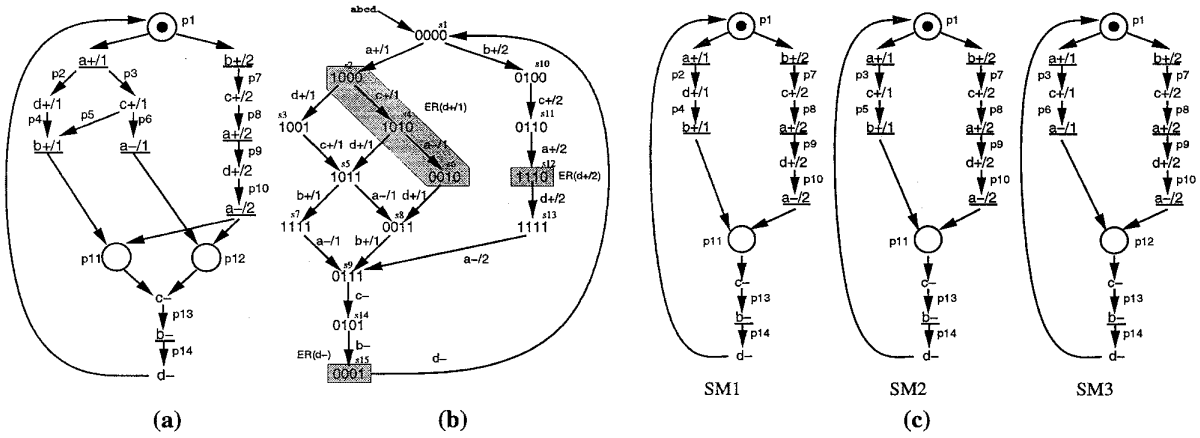

Figure 4: (a)STG example, (b) corresponding Reachability Graph, (c) SM-Cover.

[6]. We call SM-*Cover* a set of SMs such that every place in the PN is included at least in one SM. The number of components in a SM-*Cover* is bounded by the number of places in the PN.

A *Signal Transition Graph* is a triple $\langle \Sigma, S, \Lambda \rangle$, where Σ is a PN, S is a set of signals $\{a, b, c, \ldots\}$, and Λ is a labeling function $\Lambda : T \rightarrow S \times \{+, -\}$, in which the transitions are interpreted as value changes on circuit signals. Rising and falling transitions of a signal a are denoted by a+ and a− respectively, while a∗ denotes a generic transition. Multiple transitions for a signal will be distinguished by means of indices a_1+, a_2+ (in figures, instead of indices for a_1+, a+/1 will be used).

An STG is graphically represented as a directed graph with transition denoted by their names and places by circles, where places that have only one predecessor and successor transition are usually omitted. Transitions of input signals are underlined. The example of a free-choice STG with signals $S = \{a, b, c, d\}$ is shown in Fig.4(a). Fig.4(c) depicts three SMs that cover this STG.

Each marking in the STG is encoded with a *binary code* of signal values by means of a labeling function $\lambda : [M_o\rangle \rightarrow B^{|S|}$. The function λ must *consistently encode* the STG markings, that is, no marking M can have an enabled rising (falling) transition a+ (a−) if $\lambda(M)_a = 1$ ($\lambda(M)_a = 0$).

An STG specification is composed of several signal transitions and causality relations between them. Hence, to derive the correspondence between transitions and the markings of the specification different *signal regions* are defined:

- The *excitation region* $ER(a_i*)$ is the maximal connected set of markings in which transition a_i* is *enabled*.
- The *quiescent region* $QR(a_i*)$ is the maximal connected set of markings that can be reached from $ER(a_i*)$, and the *backward quiescent region* $BR(a_i*)$ is the maximal connected set of markings that can reach $ER(a_i*)$, in both cases without enabling any other transition a_j*.

A transition b_j* and signal b are *trigger* for a_i* if the excitation region $ER(a_i*)$ is reached by firing transition b_j*. Signal b is *concurrent* to a_i* if some b_j* is enabled in $ER(a_i*)$ and the firing of a_i* or b_j* does not disable the other. A *trigger* transition b_j* is *non-persistent* to a_i* if some transition b_l* is *concurrent* with a_i*, otherwise it is *persistent*.

The reachability graph for the STG in Fig.4(a) is depicted in Fig.4(b), shadowing the excitation regions for output signal d. Additionally, its quiescent and backward quiescent regions are:

$QR(d+/1) = \{ s_3\ s_5\ s_7\ s_8\ s_9\ s_{14}\}$, $BR(d+/1) = \{ s_1\}$,
$QR(d+/2) = \{ s_9\ s_{13}\ s_{14}\}$, $BR(d+/2) = \{ s_1\ s_{10}\ s_{11}\}$,
$QR(d-) = \{ s_1\ s_{10}\ s_{11}\}$, $BR(d-) = \{ s_3\ s_5\ s_7\ s_8\ s_9\ s_{13}\ s_{14}\}$.

4 Implementability Conditions for SI

The synthesis conditions for SI-circuits have been exhaustively investigated by Beerel *et al.* [2] and Kondratyev *et al.* [7]. The conditions are applied to the synthesis of a two-level *signal network* (Sum-of-Products) with a memory element on its output. In this work, we will use a similar architecture in which a signal network for each non-input signal a is constructed in four successive steps:

1. Transitions for signal a are partitioned into several sets $T^1_a+, \ldots, T^n_a+, T^1_a-, \ldots, T^m_a-$, exclusively composed of rising and falling transitions —the *transition clusters*.
2. Each cluster T^i_a* is implemented by a *region cover* $R(T^i_a*)$. A region cover is a function that implements the transition switching. The first level of the signal network consists of a set of rising and falling region covers.
3. The rising region covers are combined to create the *set region network* (S_a), while the falling region covers are combined to create the *reset region network* (R_a).
4. Finally, S_a and R_a are connected to a C-element to create the output signal. S_a forces the memory element to switch from 0 to 1, while R_a produces the switch from 1 to 0.

The required two-input C-element implements the output signal a with a next state function $a = S_a\overline{R_a} + a(S_a + \overline{R_a})$. Both the set and reset region networks are created by combining the region covers with a pair of OR-gates. Each region cover may be implemented with complex gates.

Figure 5(a)(b) shows two different implementations for output signal d in Fig.4(a). The first implementation (Fig.5(a)) corresponds to the transition cluster partitioning $T^1_d+ = \{d+/1\}$, $T^2_d+ = \{d+/2\}$ and $T^1_d- = \{d-\}$, which are implemented by region covers $R(T^1_d+) = a + \bar{b}c$, $R(T^2_d+) = a$, and $R(T^1_d-) = \overline{a + b + c}$. This circuit is not SI because if the AND-OR gate for T^1_d+ is slow enough the pulse on input a can propagate to the output S_d.

In Fig.5(b) transitions d+/1 and d+/2 are merged into one cluster T_d+. This makes the overall circuit simpler and SI (the races between inputs a and b take place only within one AND-OR gate).

The *signal regions* are extended for *transition clusters*, *e.g.* $ER(T^i_a*)$ is the union of the excitation regions for the transitions

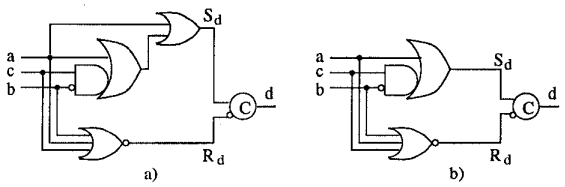

Figure 5: Two different implementations for signal d.

in T_a^i* Similar extensions exist for $QR(T_a^i*)$ and $BR(T_a^i*)$.

The synthesis objective is to simplify as much as possible the complexity of the *region covers* while preserving the correct circuit operation. A *region cover* $R(T_a^i*)$ may cover markings in $QR(T_a^i*)$ or $BR(T_a^i*)$ to minimize its literal count.

However, not all the markings in QR and BR can be freely used because some of them are shared by multiple transitions of the same signal. None of these shared markings can be covered by a region cover. Otherwise it would violate the basic requirement for the correct operation of the signal network, *i.e.* only one rising (falling) region cover of the same signal can be excited simultaneously. The subsets of the quiescent and backward quiescent regions that do not contain shared markings are called *restricted regions*, and denoted by $QR^r(T_a^i*)$ and $BR^r(T_a^i*)$ respectively.

Covering markings in the backward quiescent regions is only possible because of the characteristics of the C-element (as pointed out in [8]). Any marking in $BR^r(T_a^i*)$ covered by $R(T_a^i*)$ must be also covered by some other region cover $R(T_a^j*)$.

Definition 1 *A set of region covers $R(T_a^i*)$ for the output signal* a *is said to be* correct *if*

1. *Any marking in $ER(T_a^i*)$ is covered by $R(T_a^i*)$.*
2. *$R(T_a^i*)$ does not cover any reachable marking outside $ER(T_a^i*) \cup QR^r(T_a^i*) \cup BR^r(T_a^i*)$.*
3. *Any marking in $BR^r(T_a^i*)$ covered by $R(T_a^i*)$ is also covered by some $R(T_a^j*)$ such that $QR^r(T_a^j*) \cap BR^r(T_a^i*) \neq \emptyset$.*

Informally Def.1 requires that:

1. Every time we arrive to a marking in the excitation region of the cluster T_a^i* the output for $R(T_a^i*)$ has to go up.
2. This will be the only one output in the region network for a that goes up.
3. $R(T_a^i*)$ can go up even before entering $ER(T_a^i*)$, but in this case the opposite input of the C-element has to hold its value to prevent the premature firing of a.

Finally, the region covers must be *monotonic*, *i.e.* guarantee that every output transition in the specification is exactly implemented by a 0 to 1 and a 1 to 0 switch of its corresponding region cover. $R(T_a^i*)$ is said to be *monotonic* if it changes exactly twice in any sequence, where the rising change is at a marking in $BR^r(T_a^i*) \cup ER(T_a^i*)$ and the falling change is produced before reaching the next excitation region of signal a.

Definition 2 *A correct set of region covers $R(T_a^i*)$ for the output signal* a *is said to be* monotonic *if*

1. *$\forall M \in QR^r(T_a^i*)$ covered by $R(T_a^i*)$, $\forall M' \in QR^r(T_a^i*)$ such that $M'[t\rangle M$, M' is also covered by $R(T_a^i*)$,*
2. *$\forall M \in BR^r(T_a^i*)$ covered by $R(T_a^i*)$, $\forall M' \in BR^r(T_a^i*)$ such that $M[t\rangle M'$, M' is also covered by $R(T_a^i*)$.*

The main result proved in [2, 7] was the following:

If all the region covers satisfy the monotonous conditions, the circuit implementation is speed-independent.

The main purpose of the following sections is to show how the monotonous cover conditions can be ensured for the region covers without generating the reachability graph of the STG.

5 Single Cube Approximations

This section studies the relations between the structure of an STG —restricted to *free choice* nets— and its reachable markings. A methodology is presented to implicitly manipulate the STG without explicitly generate its markings, namely:

1. The dynamic behavior of the STG is analyzed by determining the pairwise concurrency between its transitions.
2. The markings in the STG are approximated by sets of cubes.

5.1 Concurrency Relation

The dynamic behavior of a PN is indirectly defined by analyzing which pairs of transitions can or cannot be *concurrently* enabled, namely the *Concurrency Relation*.

Transition's concurrency is defined in terms of reachable markings —if there is a reachable marking where two transitions can fire without disabling each other, the transitions are said to be *concurrent*. Additionally, the *Concurrency Relation* can be extended to places, and places and transitions [12], *e.g.* two places p_1 and p_2 are said to be *concurrent* if there is a reachable marking M in which both of them are marked.

Formally, the *Concurrency Relation* is a binary relation $\mathcal{CR}$ between the nodes of the PN, such that two nodes (u_1, u_2) are *concurrent* if $(u_1, u_2) \in \mathcal{CR}$. The $\mathcal{CR}$ relation can be defined on the set of reachable markings as [12]:

$$(t_1, t_2) \in \mathcal{CR} \Leftrightarrow \exists M : M[t_1 t_2\rangle \wedge M[t_2 t_1\rangle ;$$
$$(p_1, p_2) \in \mathcal{CR} \Leftrightarrow \exists M : M(p_1) = M(p_2) = 1 ;$$
$$(t, p) \in \mathcal{CR} \Leftrightarrow \exists M, M' : M[t\rangle M' \wedge M(p) = M'(p) = 1 .$$

A polynomial-time algorithm for the computation of the $\mathcal{CR}$ relations of a live and safe *free-choice* PN is presented in [9].

The concurrency relation between signals and nodes of the STG can be indirectly studied by using the *Concurrency Relation* of the PN. The *Signal Concurrency Relation* is a binary relation $\mathcal{SCR}$ between signals in S and nodes in the STG, such that signal a and node u_j are *concurrent* if $(a, u_j) \in \mathcal{SCR}$. Formally, $(a, u_j) \in \mathcal{SCR}$ if exists a transition a_i* concurrent to u_j, *i.e.* $(a_i*, u_j) \in \mathcal{CR}$.

5.2 Structural Marking Analysis

This section provides a methodology to approximate the markings in the STG by means of its binary code. The STG markings will be partitioned into a set of regions with respect to its nodes. A single cube is initially generated for each region. This *covering cube* covers the binary codes of all the markings in the region.

The *marked region* $MR(p)$ is the set of markings in which p is marked, *i.e.* $MR(p) = \{M \in [M_o\rangle \mid M(p) = 1\}$. The *enabled region* $ER(t)$ is the set of markings in which t is enabled, *i.e.* $ER(t) = \{M \in [M_o\rangle \mid M[t\rangle\}$. The enabled region of any transition is equivalent to the intersection of the marked regions of its predecessor places, *i.e.* $ER(t) = \bigcap_{p \in {}^\bullet t} MR(p)$.

The *Signal Concurrency Relation* provides the information to decide if the value of a signal a is constant in a marked or enabled region. However, the relative position of the place or transition with respect to the transitions a_i* is needed to determine the signal value.

Let us say that transition a_i* is *next* to a_j* if there exists a feasible sequence in the STG which contains no other transitions of signal a between a_j* and a_i* (pair (a_j*, a_i*) is *adjacent*). A

signal transition a_j* can have several transitions a_i* that are next to it, we will denote this set as $next(a_j*)$. Similarly it can be defined the set $prev(a_j*)$ for predecessor transitions.

Return to the output signal d in our STG example in Fig.4(a). Signal transition d+/1 has only one transition next to it (d−), while $next$(d−) contains both transitions d+/1 and d+/2.

Property 1 *[13] If in a free-choice live consistent STG there is a simple path L between a_i+ (a_i-) and a_j- (a_j+) such that:*

1. *for any place $p \in L$: $(a,p) \notin \mathcal{SCR}$,*
2. *L does not contain transitions of signal a;*

then there exists a feasible sequence $q = q1, a_i+, q2, a_j-$, where subsequence $q2$ does not contain any transition of signal a.

Based on Prop.1 to check whether a_i* is next to a_j* it is sufficient:

1. to find a simple path L between a_j* and a_i*,
2. to check the non-concurrency of all nodes in the path L to signal a.

Both conditions can be directly verified on the structure of the STG. Its complexity is similar to the construction of the graph transitive closure ($O(n^3)$, where n is the number of nodes in the STG).

Additionally, the STG consistency analysis can be reduced to the calculation of $next$ relation between transitions. Indeed, an STG is consistent if and only if for every transition a_j+ (a_j-) the set $next(a_j+)$ ($next(a_j-)$) contains only negative (positive) transitions of a. To our knowledge, this is the first proposed polynomial method to check consistency for any *free-choice* STG.

Let us introduce one more notion to characterize the position of a place or a transition with respect to adjacent signal transitions. The *Interleave Relation* is a binary relation $\mathcal{IR}$ between place or transition and the pair of adjacent transitions (a_j*, a_i*). A node u_j is *interleaved* with a_j* and a_i* ($u_j \in \mathcal{IR}(a_j*, a_i*)$), if there exists a simple path from a_j* to a_i* containing u_j and not containing nodes concurrent to a. For example, in Fig. 1(a), p_2 is interleaved with (x+, x−), whereas p_5 is not.

The binary codes for the markings in each one of the marked and enabled regions are implicitly determined by computing a cover cube for the region. The *cover cubes* [12] $\mathcal{C}_p$ and $\mathcal{C}_t$ are the smallest cubes, *i.e.* with the greatest number of literals, that respectively cover MR(p) and ER(t).

Property 2 *[13] If in a free-choice live consistent STG place p_k is interleaved with (a_i+, a_j-) then it cannot be interleaved with any pair (a_k-, a_l+).*

Property 2 guarantees that the value of signal a in the marked region of p depends on the position of p with respect to the transitions of a. This value can be determined by the *Interleave Relation* and will be the same for all the adjacent pairs for which p is in $\mathcal{IR}$. This gives a polynomial time algorithm (for *free-choice* STGs) to determine the value of each literal $\mathcal{C}_p^a$ in the cover cube of place p:

$$\mathcal{C}_p^a = \begin{cases} a & \exists \text{ adjacent } (a_i+, a_i-) : p \in \mathcal{IR}(a_i+, a_i-) \ , \\ \bar{a} & \exists \text{ adjacent } (a_i-, a_i+) : p \in \mathcal{IR}(a_i-, a_i+) \ , \\ - & (a,p) \in \mathcal{SCR} \ . \end{cases}$$

The cover cube $\mathcal{C}_t$ is defined in terms of its predecessor places: $\mathcal{C}_t = \bigcap_{p \in {}^\bullet t} \mathcal{C}_p$.

Table 1 depicts the cover cubes for the places and transitions of the example in Fig.4(a). Note the direct relation with the Signal Concurrency Relation between signals and STG nodes.

$\mathcal{C}_{p_1}$	0000	$\mathcal{C}_{p_2}$	-0-0	$\mathcal{C}_{a+/1}$	0000	$\mathcal{C}_{b+/2}$	0000
$\mathcal{C}_{p_3}$	100-	$\mathcal{C}_{p_4}$	-0-1	$\mathcal{C}_{d+/1}$	-0-0	$\mathcal{C}_{b+/1}$	-010
$\mathcal{C}_{p_5}$	-01-	$\mathcal{C}_{p_6}$	1-1-	$\mathcal{C}_{c+/1}$	100-	$\mathcal{C}_{a-/1}$	1-1-
$\mathcal{C}_{p_7}$	0100	$\mathcal{C}_{p_8}$	0110	$\mathcal{C}_{c+/2}$	0100	$\mathcal{C}_{a+/2}$	0110
$\mathcal{C}_{p_9}$	1110	$\mathcal{C}_{p_{10}}$	1111	$\mathcal{C}_{d+/2}$	1110	$\mathcal{C}_{a-/2}$	1111
$\mathcal{C}_{p_{11}}$	-111	$\mathcal{C}_{p_{12}}$	0-1-	$\mathcal{C}_{c-}$	0111	$\mathcal{C}_{b-}$	0101
$\mathcal{C}_{p_{13}}$	0101	$\mathcal{C}_{p_{14}}$	0001	$\mathcal{C}_{d-}$	0001		

Table 1: Cover cubes, with signal order (a b c d), in Fig.4(a).

6 Cover Approximations

Two elements are required to define a precise structural approximation of the *signal regions*:

1. a set of STG nodes that define the structure of the region,
2. a logic function for each node that determines the binary codes of the markings in the region.

6.1 Signal Region Structure

Defining the signal regions in terms of sequences in the STG provides the intuition for its structural analysis. The *excitation region* ER(a_i*) is the set of markings that enable a_i*, *i.e.*

$$\text{ER}(a_i*)=\{M \mid M[a_i*\rangle\}.$$

The *quiescent region* QR(a_i*) is the set of markings reached after a sequence $a_i* \sigma$ that does not enable another transition a_j*, $a_j* \in next(a_i*)$, *i.e.*

$$\text{QR}(a_i*)=\{M \mid [a_i* \sigma\rangle M \wedge \nexists a_j* : [a_j* \in \sigma \vee M[a_j*\rangle]\}.$$

The *backward quiescent region* BR(a_i*) is the set of markings from which there exists an allowed sequence $\sigma\ a_i*$, $|\sigma| > 0$ in which no transition of a is enabled in any proper subset of σ, *i.e.*

$$\text{BR}(a_i*)=\{M \mid M[\sigma\ a_i*\rangle \wedge |\sigma|>0 \wedge \forall \sigma' \subset \sigma : \neg M[\sigma'\ a_j*\rangle\}.$$

Hence, the structure of the signal regions can be defined by means of sets of nodes in the STG.

The *excitation region* ER(a_i*) is defined by the *excitation transition set* ETS(a_i*), containing the transition a_i*. The *quiescent region* QR(a_i*) is defined by the *quiescent place set* QPS(a_i*). A place belongs to QPS(a_i*) if it is "between" a_i* and $a_j* \in next(a_i*)$. This can be expressed by using the *Interleave Relation*, *i.e.*

$$p \in \text{QPS}(a_i*) \Leftrightarrow \exists a_j* \in next(a_i*) : p \in \mathcal{IR}(a_i*, a_j*).$$

Similarly, the *backward quiescent region* BR(a_i*) is defined by the *backward quiescent place set* BPS(a_i*). A place belongs to BPS(a_i*) if it is "between" a_i* and $a_j* \in prev(a_i*)$, *i.e.*

$$p \in \text{BPS}(a_i*) \Leftrightarrow \exists a_j* \in prev(a_i*) : p \in \mathcal{IR}(a_j*, a_i*).$$

Again, both transition and place sets can be extended to transition clusters as the union of the corresponding sets for the transitions in the cluster.

The introduced notions can be illustrated on the example of STG in Fig.4(a). For cluster T_d^1+ the quiescent place set includes all the places "between" d+/1 and d− that gives: QPS(T_d^1+) = $\{p_4, p_{11}, p_{13}, p_{14}\}$. The backward quiescent place set for T_d^1+ consists of places that are between d− and d+/1, *i.e.* BPS(T_d^1+) = $\{p_1, p_2\}$. Applying similar considerations: QPS(T_d^2+) = $\{p_{10}, p_{11}, p_{12}, p_{13}, p_{14}\}$ and BPS(T_d^2+) = $\{p_1, p_7, p_8, p_9\}$.

The *restricted quiescent region* is approximated by the *restricted quiescent place set* QPS^r(T_a^i*), computed as the *quiescent place set* minus the places shared with the *quiescent place sets* of other transitions of the same signal, *i.e.*

$$\text{QPS}^r(T_a^i*) = \text{QPS}(T_a^i*) - \bigcup_{a_j* \notin T_a^i*} \text{QPS}(a_j*) \ .$$

A similar reasoning defines the *restricted backward quiescent place set* $\mathrm{BPS}^r(T_a^i*)$ as,

$$\mathrm{BPS}^r(T_a^i*) = \mathrm{BPS}(T_a^i*) - \bigcup_{a_j* \notin T_a^i*} \mathrm{BPS}(T_a^j*) \ .$$

To produce the restricted quiescent place set for T_d^1+ *e.g.* in Fig.4(a) we need to remove from $\mathrm{QPS}(T_d^1+)$ places p_{11}, p_{13} and p_{14}, because these places are shared with $\mathrm{QPS}(T_d^2+)$. Note, that if we merge transitions $d+/2$ and $d+/2$ into one cluster $T_{d+} = \{d+/1, d_{+2}\}$ then the quiescent place set and the restricted quiescent place set will coincide: $\mathrm{QPS}^r(T_{d+}) = \mathrm{QPS}(T_{d+}) = \mathrm{QPS}(T_d^1+) \cup \mathrm{QPS}(T_d^2+)$.

6.2 Cover Correctness

Each one of the nodes used to structurally define the signal regions has assigned a logic function, named *cover function* and denoted $\mathsf{cv}(u)$. This cover function approximates the binary codes of the markings in the signal region of u.

The proposed methodology starts using the *cover cubes* as initial *cover functions*. However, the $\mathrm{QPS}(a_i*)$ and $\mathrm{BPS}(a_i*)$ sets are imprecise approximations at their boundaries. By definition $\mathrm{QPS}(a_i*)$ e.g. contains all the places "between" a_i* and $a_j* \in next(a_i*)$. However the cover function of input places of a_j* covers also $\mathrm{ER}(a_j*)$. It is easy to see that in $\mathrm{ER}(a_j*)$ the function for signal a has to change its value. Hence, to avoid the overlapping of $\mathrm{QPS}(a_i*)$ and $\mathrm{ETS}(a_j*)$ the predecessor places $p \in {}^\bullet(a_j*)$ used in $\mathrm{QPS}(a_i*)$ should be recomputed into: $\mathsf{cv}(p)_{\mathrm{qps}} = \mathsf{cv}(p) - \mathsf{cv}(a_j*)$. (Note that the refinement of the places of the quiescent region also refines the backward quiescent region.)

In case $\mathsf{cv}(a_j*)$ is *overestimated*, it may result in an *underestimation* of the markings covered by $\mathrm{QPS}(a_i*)$. Overestimations for the transitions of the STG can be detected using the information provided by a SM-Cover of the underlying PN. Checking that the existing cover functions can be properly used to approximate the different signal regions requires the intersection of the cover functions $\mathsf{cv}(p_i)$ and $\mathsf{cv}(p_j)$ for all pairs of different places in each SM of the SM-Cover. Empty intersection for all pairs guarantee that an STG is free of *coding conflicts*.

Property 3 *[13] Given an SM-Cover SMC, the STG is free of coding conflicts if for every SM-Component SM ∈ SMC:*

$$\forall (p_i, p_j) \in SM, i \neq j \ : \ \mathsf{cv}(p_i) \cdot \mathsf{cv}(p_j) = \emptyset.$$

The absence of *coding conflicts* guarantee the completeness of the ETS, QPS, and BPS approximations. However, the cover cubes may still give an overestimated approximation to the signal regions. The following property gives the conditions when such an overestimation is safe.

Property 4 *If the cover cube* $\mathsf{cv}(a_i*)$ *is not intersecting with the covers for the excitation and quiescent regions of other transitions of* a *then* $\mathsf{cv}(a_i*)$ *is a correct cover of* $\mathrm{ER}(a_i*)$. [2]

Property 4 gives only sufficient conditions for a correct cover. If these conditions are not satisfied it does not mean that the correctness of the cover is violated. The forbidden intersection may happen at the *dc-set*. Here we have two possibilities:

1. To be conservative and to consider every intersection as a bad one. Then by adding state signals the covers always can be reduced to non-intersecting (see Section 8 for details).
2. To refine the covers of regions. If the refinement is done up to exact covers then conditions of Property 4 always can be satisfied for an STG without USC conflicts. However the refinement technique leads to a growth of the number of cubes in the cover and is computationally expensive (see for details Section 7).

Another condition that has to be checked for covers is the monotonicity requirement (see Def.2). If the correct cover condition is satisfied for $\mathsf{cv}(a_i*)$ then $\mathsf{cv}(a_i*) \cdot \mathsf{cv}(a_j*) = \emptyset$ and cube $\mathsf{cv}(a_i*)$ has to be turned off somewhere inside $\mathrm{QR}(a_i*)$.

By examining the transitions that are in $\mathcal{IR}(a_i*, a_j*)$ we can find the set $\mathsf{cv}(a_i*)\downarrow$ that contains all the transitions turning off cube $\mathsf{cv}(a_i*)$ for the first time. The monotonicity condition says that after cube $\mathsf{cv}(a_i*)$ is turned off by the transition $t \in \mathsf{cv}(a_i*)\downarrow$ it cannot be turned on again inside $\mathrm{QR}(a_i*)$.

Let us generalize the *Interleaving Relation* for the pairs (t, a_j*), where $t \in \mathsf{cv}(a_i*)\downarrow$. All places that are in $\mathcal{IR}(t, a_j*)$ can be reached only after the firing of t, i.e. after the cube $\mathsf{cv}(a_i*)$ is turned off. Therefore the monotonicity is ensured if cube $\mathsf{cv}(a_i*)$ is never turned on again in the markings that are covered by marked regions of places $p \in \mathcal{IR}(t, a_j*)$. This is characterized formally in the following Property.

Property 5 *The correct cover cube* $\mathsf{cv}(a_i*)$ *is monotonous if for any* $a_j* \in next(a_i*)$, *any* $t \in \mathsf{cv}(a_i*)\downarrow$ *and any place* $p \in \mathcal{IR}(t, a_j*)$ *the cube* $\mathsf{cv}(a_i*)$ *is not intersecting with* $\mathsf{cv}(p)$.

It is easy to see that when the notion of correct cover is extended by backward quiescent regions we are dealing with the requirements on the BR that are somewhat similar to monotonicity. They can be checked in the same way as Property 5 suggests.

Properties 4 and 5 give simple sufficient conditions for the analysis of the monotonous cover requirement by the region covers. Two different techniques to ensure them are demonstrated in next sections.

7 Refinement of signal region covers

The initial cover cube approximations can be rough —only the fact of concurrency between places (transition) and the signal is used. This binary concurrency relation is not sufficient because from a concurrent to b and a concurrent to c nothing can be said about the joint concurrency of a, b and c (nodes b and c can be ordered *e.g.*). To exploit more exactly the structure of casual relations between STG nodes we can refine the initial approximation for place or transition cover cubes by other cubes in the STG.

The set of SM-Components that covers the STG is complete in the sense that no information about the STG is lost under such a partition. One SM-Component reflects the causal relations between the STG nodes only partially, however their cover set represents it completely. This observation is the main one under the idea of refinement.

Formally, the refinement of the cover cube $\mathsf{cv}(p)$ of place p [3] by a SM-component SM results in the cover that is obtained as the intersection of $\mathsf{cv}(p)$ with the sum of the cover cubes of the places $p_i \in$ SM that are concurrent to p, *i.e.*

$$\mathsf{cv}^{\mathrm{ref}}(p) = \sum_{p_i \in \mathrm{SM} : (p, p_i) \in \mathcal{CR}} \mathsf{cv}(p) \cdot \mathsf{cv}(p_i) \ .$$

Such a refinement procedure is safe for free-choice live STGs, that means that no marking from the marked region of a place can be lost while making a refinement [13].

Indeed place p is refined only by the places of SM that are concurrent to it. Then the only marking that can be removed via refinement is the marking M which marks p but no place $p_i \in$ SM, $(p_i, p) \in \mathcal{CR}$. But the latter contradicts the liveness of the STG because under marking M SM will contain no token.

Refining by using the SM-Components in a SM-Cover permits more STGs to satisfy Prop.3 by eliminating *fake coding conflicts*.

[2] Here for clarity the simplified notion of correct cover is used without extension to the backward quiescent region.

[3] transition refinement is obtained as the intersection of the refined cover cubes of its input places.

A coding conflict between p_i and p_j ($\mathsf{cv}(p_i) \cdot \mathsf{cv}(p_j) \neq \emptyset$) is a *fake coding conflict* if exists a SM-Component SM covering p_i, where place p_i has no coding conflicts. In that case we can conclude that $\mathsf{cv}(p_j)$ is overestimated and should be refined by using SM.

Obviously, several undetected overestimations may remain. Making a sufficiently large number of refinement steps we will arrive to the exact cover for every place, however such an approach has two shortcomings:

1. It increases the number of cubes for the place cover. In extreme it can be comparable to the number of markings in the marked region.
2. The question about the minimal set of SM-Components that is sufficient for exact refinement is an open one. It is still needed to be proved that the set of SM-Components that cover STG can always make a refinement exact.

Due to the growth of the cube number in the cover, the application of the refinement technique is restricted to 1-2 iterations. Refinement is applied in those cases when for sure the one-cube cover cannot exist, in persistency violations *e.g.* [13]. The idea of refinement was illustrated in Section 2 where one iteration was sufficient to get the exact cover for the non-persistent transition of signal y.

8 Insertion of state signals

When the cover functions do not fulfill the conditions for synthesis and we do not like to perform the refinement for the covers of places and transitions, state signals can be inserted.

This situation will be detected as follows. Given a SM-Component SM, all places that belong to SM define a complete partition of all reachable markings of the STG (since no more than two places of the same state machine can be marked simultaneously). A state signal will be inserted when, for two places p_1 and p_2 of SM we have that $\mathsf{cv}(p_1) \cdot \mathsf{cv}(p_2) \neq \emptyset$. This non-empty intersection can be produced by two facts:

1. The STG has no USC, or
2. The overestimation of the covers produce *fake* conflicts.

In both cases, the insertion of state signals can disambiguate the contradictions among covers. This was illustrated in the example of Fig.1(a), in which signal v was inserted to disambiguate the covers of places p_1 and p_5.

The algorithm for state signal insertion used in our tools is similar to the one proposed in [11], and it is based on the bi-partition of the SM-Components that contain the places with intersecting covers. The algorithm works on the structure of the STG and has $O(n^3)$-time complexity, n being the number of places and transitions of the net.

9 Experimental Results

9.1 Structural Synthesis Methodology

This section combines the synthesis conditions and the structural signal region analysis into a heuristic minimization algorithm. The objective is to reduce the number of cubes and literals required to implement a set of monotonous region covers by using gates in a given gate library. The reductions are achieved by sequentially applying logic minimizations. Each successfully applied minimization reduces the complexity of the implementation, but assuring that the transformed implementation remains correct. The required gates are matched against the gate library, and finally are optimized by using a boolean matching technology mapping algorithm.

Practically, the signal region analysis methodology allows the implementation of multiple signal transitions by the same region cover combining several transitions of a signal into a *transition cluster*. Each transition cluster is implemented by a single region cover $R(T_a*)$. The utilization of transition clusters allows a better usage of the available complex-gates in the libraries.

Three sets of logic transformations can be applied in order to reduce the complexity of the region covers, namely:

- The expansion of each region cover by means of the iterative elimination of its literals towards the restricted quiescent and backward place sets, and *dc-set*.
- The analysis of the interrelations between the rising or falling region covers. Then, detecting pairs of region covers that can be merged (two transition clusters are joined into a single one) reducing the number of required region covers.
- Use the structure of the signal networks to combine the rising and falling region covers with the memory element.

Region cover expansion as well as *complete* region cover derivation are well known techniques used in [2, 7] that are directly applied here. Traditionally, pairs of single-cube region covers with the same support at distance 1 are substituted by its merging consensus. However, region cover merging extends this transformation to multi-cube region covers taking advantage of the existence of complex gates. Additionally, the region covers and the memory element are combined in a particular case in which $\mathsf{a} = v_1v_2\,\overline{v_1}\overline{v_2} + \mathsf{a}(v_1v_2 + \overline{v_1}\overline{v_2})$, is substituted by $\mathsf{a} = v_1v_2 + \mathsf{a}(v_1 + v_2)$. This transformation reduces two AND gates and one C-element to only one C-element implementation. We also apply a similar transformation in which $\mathsf{a} = v_1v_2\,\overline{v_1\overline{v_2}} + \mathsf{a}(v_1v_2 + \overline{v_1\overline{v_2}})$, is substituted by $\mathsf{a} = v_1v_2 + \overline{v}_1\mathsf{a}$, reducing two AND gates and one C-element to only one gated-latch implementation.

9.2 Area of the circuits

Table 2 compares the area results of several synthesis tools including our methodology. The first column depicts the number of markings for the benchmark, while the columns labeled SYN, FCG and S3C reports the area obtained by the synthesis methodologies developed at Stanford [1], Aizu [8], and our methodology. The results show that the combination of the structural approach and the new logic minimization techniques results in significant improvements[4]—23% area reduction with respect to [1]— in short CPU times —less than 8 secs. for the worst case (pe-send-ifc).

If we take into account that some of the new minimization techniques were not used by SYN, we can at least claim that using structural methods does not negatively influence on the the quality of the circuits obtained.

9.3 Running time: structural vs. state-based

In order to illustrate the effectiveness of structural over state-graph-based methods, we have run some experiments for some STGs with a large state space and compared the running times with SIS[15] and ASSASSIN[16].

Table 3 reports the CPU times in a SUN SPARC20 workstation. All the examples fulfilled the CSC property. For SIS we report the running time for the command `astg_to_f`. For ASSASSIN we report the running time for the commands `assa_stg_to_sg` and `assa_haz_logic`. The superiority of structural methods is evident.

Interestingly, the dining philosophers benchmark is one of the examples that illustrates that non-free-choice STGs can also be synthesized if a cover of state machines can be found for the net. Another scalable example is the Muller pipeline (see for the description Section 2 and Fig.3). Its STG contains no choice places and the circuit realization is a chain of C-elements. The column labeled SMs indicates the number of SM-Components required to cover the net for the dining philosophers and the Muller pipeline examples.

[4] all synthesis results have been formally verified to be speed-independent.

STG	States	SYN	FCG	S3C	STG	States	SYN	FCG	S3C
chu133	24	232	216	208	wrdatab	216	784	744	488
chu150	26	248	232	128	xyz	8	200	192	136
chu172	12	168	112	104	alloc-outbound	21	400	-	308
converta	18	376	320	258	mp-forward-pkt	22	320	-	256
ebergen	18	352	280	120	nak-pa	58	336	-	320
full	16	112	112	80	pe-rcv-ifc(*)	65	1304	-	1146
hazard	12	248	240	80	pe-send-ifc(*)	117	1632	1632	1122
hybridf	80	152	152	130	ram-read-sbuf	39	432	-	360
nowick	20	456	456	274	rcv-setup	14	144	-	120
qr42	18	352	280	120	sbuf-ram-write	64	320	-	304
rpdft	22	224	-	160	sbuf-read-ctl	19	296	-	258
trimos-send	336	648	-	552	sbuf-send-ctl	27	280	-	226
vbe10b	256	792	784	608	sbuf-send-pkt2	28	504	-	364
vbe5b	24	192	192	208	sendr-done	9	80	-	82
vbe5c	24	200	200	152					
Total							11784		9076
Total⋆							6496	6144	4216

Table 2: Area results (∗ non-free-choice STGs, ⋆ totals restricted to STGs synthetized by FCG).

STG	states	cubes	area	CPU SIS	CPU ASSASSIN	CPU S3C
tsbmsi	1024	143	744	43	256	1
tsbmSIBRK	4730	298	1136	1876	12219	8
master-read-1	2254	117	610	250	1252	2
master-read-2	18856	2786	1052	> 9 h.	> 24 h.	34
master-read-3	21848	2802	1102	> 9 h.	> 24 h.	35
par4	1303	95	500	80	271	1
par8	1.7×10^6	342	920	-	-	7
par16	2.8×10^{12}	1190	1768	-	-	45

(a)

STG	states	SMs	CPU
phil3(*)	864	9	2
phil10(*)	7.4×10^9	30	120
phil20(*)	5.5×10^{19}	60	1455
muller10	420	16	1
muller50	1.7×10^{11}	96	85
muller100	1.2×10^{27}	196	1437

(b)

Table 3: CPU time (in seconds) for synthesis: (a) comparison with SIS and ASSASSIN, (b) scalable examples (∗ non-free-choice STGs).

10 Conclusions

Structural techniques for the analysis and synthesis of STGs are essential when the size of the state space becomes unmanageable by algorithms that work at state graph level.

This paper has presented new methods to synthesize STGs with underlying free choice PN. The proposed algorithms have polynomial complexity in the size of the net and can be easily extended to the class of PNs that can be covered by SM-Components. Although the calculation of a SM-Cover cannot be calculated in polynomial time for non-free-choice nets, some recent symbolic techniques have been proved to be very efficient [5]. Future work will be devoted to fully characterize the class of Petri nets that can be handled by the presented techniques.

The experimental results show that the proposed methods obtain efficient implementations in short CPU times. Some of the existing tools have not been able to synthesize the largest circuits, whereas the presented method has managed to do it in few seconds.

Acknowledgments

This work has been supported by the Ministry of Education of Spain (CICYT) under contract TIC 95-0419, Departament d'Ensenyament de la Generalitat de Catalunya. We are also indebted to Michael Kishinevsky for many valuable discussions.

References

[1] Peter A. Beerel. *CAD Tools for the Synthesis, Verification, and Testability of Robust Asynchronous Circuits.* PhD thesis, Stanford University, August 1994.

[2] Peter A. Beerel and Teresa H. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proc. ICCAD*, pages 581–586, November 1992.

[3] Tam-Anh Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, June 1987.

[4] J. Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, Cambridge, Great Britain, 1995.

[5] F. García-Vallés and J.M. Colom. A boolean approach to the state machine decomposition of Petri nets with OBDD's. In *Proc. of the 1995 IEEE Int. Conf. on Systems, Man and Cybernetics*, October 1995.

[6] M. Hack. *Analysis of production schemata by Petri nets.* M.s. thesis, MIT, February 1972.

[7] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proc. DAC*, pages 56–62, June 1994.

[8] A. Kondratyev, M. Kishinevsky, and A. Yakovlev. Monotonous cover transformations for speed-independent implementation of asynchronous circuits. Technical Report 94-2-002, The University of Aizu, Japan, July 1994.

[9] A. Kovalyov and J. Esparza. A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. Technical Report TUM-I9528, SFB-Bericht Nr. 342/15/95 A, Technische Universität München, 1995.

[10] K.-J. Lin and C.-S. Lin. Automatic synthesis of asynchronous circuits. In *Proc. DAC*, pages 296–301, June 1991.

[11] Enric Pastor and Jordi Cortadella. An efficient unique state coding algorithm for signal transition graphs. In *Proc. ICCD*, pages 174–177, October 1993.

[12] Enric Pastor and Jordi Cortadella. Polynomial algorithms for the synthesis of hazard-free circuits from signal transition graphs. In *Proc. ICCAD*, pages 250–254, November 1993.

[13] Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig. Structural algorithms for the synthesis of speed-independent circuits from signal transition graphs. Technical Report RR-95/35, UPC/DAC, October 1995.

[14] L. Ya. Rosenblum and A. V. Yakovlev. Signal graphs: From self-timed to timed ones. In *International Workshop on Timed Petri Nets*, pages 199–206, July 1985.

[15] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuits synthesis. Technical Report M92/41, UCB/ERL, May 1992.

[16] C. Ykman-Couvreur, B. Lin, and H. De Man. ASSASSIN: A synthesis system for asynchronous control circuits. Technical report, IMEC, September 1994. User and Tutorial manual.

[17] C. Ykman-Couvreur, Bill Lin, Gert Goossens, and Hugo De Man. Synthesis and optimization of asynchronous controllers based on extended lock graph theory. In *Proc. EDAC*, pages 512–517, February 1993.