Partial order based approach to synthesis of speed-independent circuits[1]

Alex Semenov, Alexandre Yakovlev
Department of Computing Science
University of Newcastle
Newcastle upon Tyne, NE1 7RU
England

Enric Pastor, Marco A. Peña,
Jordi Cortadella
Department of Computer Architecture
Universitat Politèchnica de Catalunya
08071 Barcelona, Spain

Luciano Lavagno
Dip. di Elettronica
Politecnico di Torino
C. Duca degli Abruzzi 24
10129 Torino, ITALY

Abstract

The aim of this paper is to introduce a novel technique for synthesis of speed-independent circuits from their Signal Transition Graph specifications. The new method uses partial order in the form of the STG-unfolding segment to derive the logic implementation. It is based on a new notion of *slice*, which localises the behaviour of a particular signal instance in a structural fragment of the segment. Two approaches are explained in this paper: exact and approximation. Within the approximation approach two strategies for cover derivation are considered. The method is applied to synthesis in three main implementation architectures. The experimental results show the power of the approximation approach in comparison with the existing methods.

[1]This work was supported in part by the SERC grant No. GR/J 52327 and ESPRIT ACiD–WG Nr.21949. Collaboration between University of Newcastle and Universitat Politèchnica de Catalunya was supported by British-Spanish joint research programme (Acciones Integradas) between the British Council and Spanish Ministry of Education and Culture, grant number MDR(1996/97)1159

Partial order based approach to synthesis of speed-independent circuits

1 Introduction

There exists a variety of approaches to synthesis of speed independent circuits from their Signal Transition Graph (STG) specifications. These approaches can be divided according to the library of elements used to implement output signals. For example, [2, 1] uses a Muller C-element for each implementable signal and a network of gates to drive it. Work presented in [7] uses an RS-latch in similar conditions. Early methods, e.g. [4, 12], assume that each signal is implemented as a single complex gate. Later techniques, e.g. [20, 9], attempt to decompose the complex gates preserving the speed-independence[2] of the circuit.

Another taxonomy in synthesis of speed-independent circuits is established by the methods used to obtain the logic functions. Two primary approaches exist to date: State Graph (SG) based and structural methods (eliciting information for the synthesis from the structure of the STG). The first approach constructs a SG as a model representing behavioural properties of the "circuit-to-be". It then proceeds with extracting subsets of states required for implementation. This method is used in such tools as SIS [19] and Assassin [22]. A recently developed tool Petrify (and approach) [5] uses Binary Decision Diagrams (BDDs) to represent the state space. Due to implicit representation of the SG this tool is efficient in synthesising moderate sized examples. As all these methods work with the full SG, they suffer from state explosion, i.e. the number of reachable states grows exponentially with the size of specification.

The structural method of [14] uses State Machine (SM) decompositions of STGs to obtain concurrency relations between signal transitions. Using these relations, this method finds an implementation avoiding the state exploration. Thus it demonstrated impressive results although it is restricted to free-choice specifications. The method described in [23] also uses structural information of the STGs (lock relations between signals) to synthesise circuits.

Partial order techniques have also been applied in synthesis process. Change Diagrams were introduced in [7] for synthesis of speed independent circuits from choice-free (no choice at all) specifications. This significantly restricted their use. Nevertheless, this work was a significant step in the development of this approach – it was first to establish the relationship between the connected sets of states (e.g. excitation regions in the SG) and elements of the event-based description (event instances in the Change Diagram unfolding).

A more recent work of [13] used Petri Net (PN) unfoldings to derive logic functions. This work, however, is based on restoring the state space from the partial order and is therefore also prone to state explosion.

At the same time, work of [17] showed that STG specifications can be verified efficiently using the STG-unfolding segment, based on PN unfoldings. In a vast number of examples the segment can be constructed and verified where the construction of the SG fails. Thus, after the verification is completed the segment can be used for deriving the logic functions of implementation.

This paper proposes a novel approach for synthesis of speed independent circuits from the STG-*unfolding segment* of their specifications. It introduces a new notion of *slice* which helps to *localise* the behavioural information for signal instances within a structural fragment of the segment (the original idea of such localisation comes from [7]). An *exact* approach,

[2]Speed-independent circuits are hazard-free for any delays attached to their gate outputs.

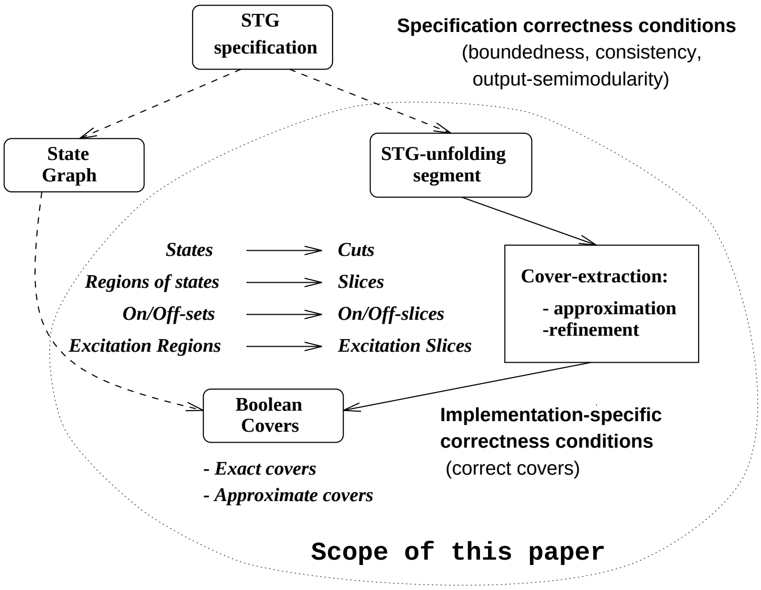

Figure 1: Overview of synthesis issues discussed in the paper.

producing implementations comparable with those of the SG approach, is given. Although it benefits from STG-unfolding segment localisation, the exact approach may still suffer from state explosion. To overcome this problem, an *approximation* method based on temporal relations found in the segment is suggested. However, unlike [14], our approach works with partial order representation of a fragment of system's execution. Therefore, it uses local dependency information available for each instance of every signal transition. Only instances of signal transitions which are concurrent to a particular instance are considered for it. This gives a more accurate initial approximation and a more precise refinement. We illustrate our approach applying it to synthesis in three major implementation architectures. The scope of synthesis issues presented in this paper is summarised in Figure 1.

The paper is organised as follows. Section 2 describes the general approach and architectures for synthesis of speed-independent circuits from their STG specifications. It also defines the implementation-specific (cover) correctness conditions. Section 3 introduces new notions required for synthesis from the STG-unfolding segment such as cuts and slices. Section 4 describes the ways for exact cover extraction from the STG-unfolding. Section 5 presents our main result – the cover extraction method based on approximation and refinement. The results of experiments, showing performance and comparing the new method with the existing ones are in Section 6.

2 Speed-Independent Circuit Implementation

This section introduces the basic concepts required to develop the synthesis of speed-independent circuits. Further reading on Petri nets, Signal Transition Graphs, and logic synthesis can be obtained at [15, 16, 4, 3].

2.1 Basic Synthesis Concepts

A *Petri net* (PN) is a 4-tuple $\langle P, T, F, m_o \rangle$, with sets of places P, transitions T, flow relation F and initial marking m_o. A marking m is represented with a number of tokens $m(p)$ in each place $p \in P$. A *Signal Transition Graph* (STG) [16, 4] is a triple $\langle N, A, L\rangle$ where N is a PN, A is a set of signals, and $L : T \to \{+, -\} \times A$ is a labelling function that assigns a signal switch to each transition in T. An STG is a special case of labelled PN, particularly useful to describe low level asynchronous circuits. The set of transitions represents up $(+a_i)$ and down $(-a_i)$ switching activity. Notation $*a_i$ is used to indicate a signal transition regardless of the direction of the change. Given an element $x \in T \cup P$, its predecessors and successors sets are denoted $\bullet x$ and $x\bullet$ respectively.

An STG is called *k-bounded* iff the number of tokens in any place $p_i \in P$ at any reachable marking does not exceed k. Boundedness guarantees that an STG can be implemented using a finite number of memory elements. An STG is called *output semi-modular* iff no output signal transition $*a_i$ excited at any reachable marking can be disabled by transition of another signal $*a_j$ (also known as *output signal persistency* [7]). If an STG is output semi-modular, then can be implemented without producing unspecified changes of the output signals; that is, without introducing *hazards* [7].

To obtain an implementation for an STG, most of the existing synthesis techniques build its corresponding SG. The SG is derived by constructing the reachability graph for the STG (representing all reachable states), and then assigning a binary code v_i to each state s_i. In order to allow a meaningful interpretation of the model as the behaviour of a (generally sequential) asynchronous circuit, the binary codes must be assigned *consistently*, i.e.

- every arc between two states s_1 and s_2 is labelled with exactly one signal transition $*a_i$,
- if the arc (s_1, s_2) is labelled $+a_i$ $(-a_i)$ then $v_1[i] = 0(1)$ and $v_2[i] = 1(0)$.

An STG is called *consistent* if its SG has a consistent state assignment. States of the SG generated by a consistent STG have two components: marking and binary code. At the circuit level, however, the states are represented by their binary codes only. Thus it may be possible that two states with equal binary codes will be indistinguishable at the circuit level. This situation is often referred as *coding conflict*. The *Complete State Coding* (CSC) condition introduced in [4] requires any two states with equal binary codes to have the same set of excited output signals. If for some signal a_i this requirement is not satisfied, then it is impossible to extract the boolean function for its implementation. It was shown in [4] that STGs satisfying CSC are implementable [3] as speed-independent circuits.

An STG model satisfying the *general specification correctness* conditions (boundedness, consistency, output semi-modularity) and producing a SG with CSC, gives rise to truth tables, which can be obtained from the SG state codes for each output signal. However, the process of obtaining a truth table depends on a particular implementation architecture chosen for each signal. For example, we can extract truth tables for functions associated with the states where a signal is enabled, or it is stable or its implied value is TRUE etc.

The following three architecture types are commonly considered for the synthesis of speed-independent circuits:

1. *Atomic complex gate per signal* (ACGpS)implementation;
2. *Atomic complex gate per excitation function* (ACGpEF) implementation;

[3] Note that it is however possible to obtain the logic implementation only for the signals for which the coding conflict does not arise. In fact, our unfolding-based method starts to extract boolean covers in some sense "blindly", assuming that the model satisfies the CSC conditions. The check of the latter is therefore indirect – the problems with coding conflict are manifested as a failure to extract a correct cover. More on this is in Section 5.

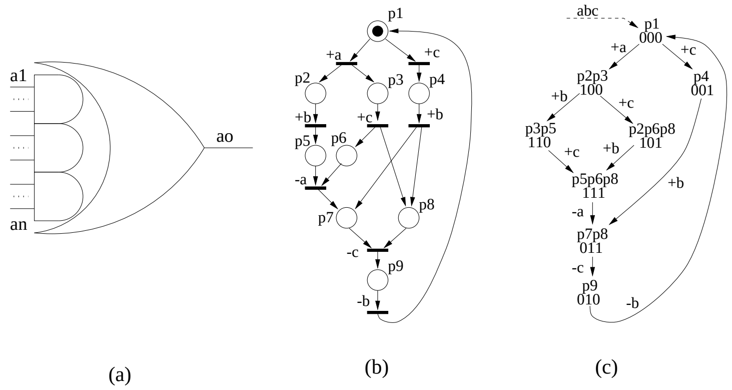

Figure 2: An example of an STG and its corresponding SG.

3. *Atomic complex gate per excitation region* (ACGpER) implementation.

Such implementations are obtained by building cover functions. This process is central for the implementation stage – the obtained covers are then directly associated with the elements in the circuit. A boolean function *covers* a state s_j if the function evaluates to TRUE when the variables have their values equal to the signals at the binary code v_j. A function *covering* a set of states is called a *cover function* (or simply *cover*). Each term of the cover is called *cube* as it may cover several states in the state space.

An *exact* cover for a set of states $\{s_i\}$ can be obtained directly from the set of binary codes v_j, but it will require an explicit enumeration of all the states. Generating exact covers is very costly due to the exponential number of states that may contain highly concurrent STGs — this is known as the state explosion problem. To overcome this, *approximated* covers can be generated using some structural information from the STG, and therefore avoiding the state generation [23, 14]. However, implementations created by using approximated covers require additional checking for its correctness.

The the notion of correctness, applied at the implementation stage, is concerned with the requirement of hazard-freedom with respect to the gates of the synthesized circuit. It is different from the general correctness conditions (discussed above) related to the specification. As a matter of fact, this notion is essentially dependent on the implementation architecture because the sets of states s_i for which the boolean covers are extracted are different for different architectures. Thus, the *architecture-specific correctness criteria* put into correspondence the sets of states s_i and the cover functions. We discuss these criteria for each architecture in more detail in the following subsections.

In order to further demonstrate our unfolding-based method we chose relatively simple cover correctness criteria which, however, guarantee existence of the implementation in all three architectures for a CSC-compliant STG. For instance, in the last two architectures, as a target for boolean covering, we only consider states where signals are excited. Several recent papers, e.g. [14, 9], examine the possibility of expanding the set of covered states with states at which signals are stable. This, however, can be viewed as optimisation aimed at reducing the sizes of customised complex gates. Note that the limits of this paper do not allow us to consider lower level logic decomposition, usually known as technology mapping [20].

2.2 Atomic Complex Gate per Signal

This is the initial architecture for speed-independent circuits studied in [4, 12, 16]. The circuit is implemented as a network of atomic gates, each one implementing one output signal. The boolean function for each gate can be represented as Sum-Of-Products (SOP). A simple example of such gate is presented in Figure 2. Each atomic gate contains a combinational part, and a possibly sequential part implemented as an internal feedback. The delay between its "ANDing" and "ORing" nodes, and the internal feedback is assumed to be negligible. In figures, the gate representation is used to denote the implemented logic function, but the actual implementation is resolved on the transistor level.

Two (mutually complementary) subsets of the reachable states are distinguished in the SG for every signal a_i, *on-set* $On(a_i)$ and *off-set* $Off(a_i)$, which include all states in which the value of the output signal a_i is implied to be TRUE and FALSE, respectively. The remaining (unreachable) subset of combinations of the boolean values of signals forms the *Don't care* set (DC-set).

The implementation is assumed to be derived by building the on-set (the off-set may be chosen if it leads to a simpler circuit). Each state can be represented by a term with one variable for each signal $a_i \in A$ in the STG. The term becomes TRUE only when the values of the variables are equal to those in the binary code assigned to the state. The cover $\mathcal{C}$ for the implementation is obtained from the terms included into the on-set. The DC-set can be used for optimising the size of $\mathcal{C}$ applying standard minimisation tools such as Espresso [19].

Example. The synthesis for this architecture is illustrated in Figure 2 for an STG shown in the Figure 2(b). Suppose that an implementation of signal b is required. The on-set of b is found as: $On(b) = \{(p_2, p_3), (p_3, p_5), (p_2, p_6, p_8), (p_5, p_6, p_8), (p_7, p_8), (p_4)\}$. The cover function $\mathcal{C}(b)$ is obtained as: $\mathcal{C}(b) = a\bar{b}\bar{c} + ab\bar{c} + a\bar{b}c + abc + \bar{a}bc + \bar{a}\bar{b}c = a + c$. The DC-set in example in Figure 2(c) is empty so no further minimisation can be done. ■

Obtaining exact covers usually means that all states in the on- or off-set must be known. An approximation algorithm produces approximated covers of the on- and off-sets, and helps avoiding state explosion. Therefore, in this implementation architecture, the covers of on- and off-sets (either exact or approximated) must satisfy the following condition:

Definition 2.1 Given an output signal a_i, two covers $\mathcal{C}_{On}(a_i)$ and $\mathcal{C}_{Off}(a_i)$ are said to be correct iff $\mathcal{C}_{On}(a_i)$ and $\mathcal{C}_{Off}(a_i)$ cover $On(a_i)$ and $Off(a_i)$ respectively and $\mathcal{C}_{On}(a_i) \cdot \mathcal{C}_{Off}(a_i) \subseteq$ DC-set. □

A correct cover may become TRUE when its variables take values corresponding to the combinations belonging to the DC-set.

2.3 Atomic Complex Gate per Excitation Function

The ACGpEF architecture was suggested and studied extensively in a number of papers, e.g. [2, 8, 1]. It assumes that a separate memory element is used to produce an output signal. The Set and Reset *excitation* functions for this memory element are implemented as atomic complex gates. Depending on which memory element is used, the implementations are divided into: **i)** *Standard C-element* implementation, which uses Muller C-element as the memory element (shown in Figure 3(a)), and **ii)** *Standard RS-latch* implementation, where an RS-latch is used.

A *Generalised Excitation Region* (GER) of $*a_i$, denoted as $GER(*a_i)$, is defined as a set of states of SG which has an arc coming out of them which is labelled with $*a_i$. For each signal there exist two GERs found for all positive ($GER(+a_i)$) and negative ($GER(-a_i)$) transitions. GERs represent all states in which signal a_i is excited. In an output semi-modular STG, an

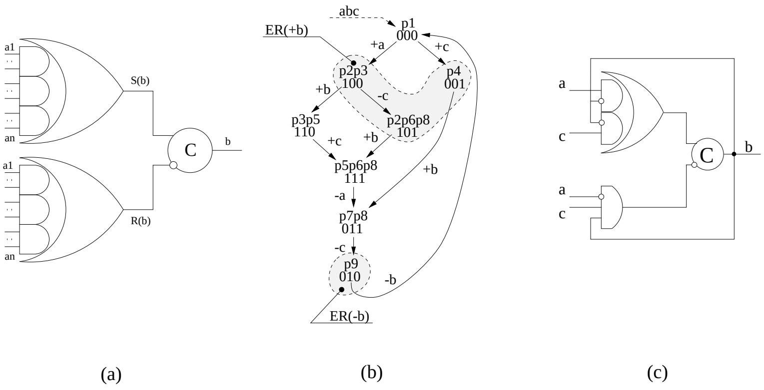

Figure 3: Atomic complex gate per excitation function architecture.

output signal a_i cannot be disabled by firing any other signal. Thus $GER(+a_i)$ and $GER(-a_i)$ can only be left through the firing of $+a_i$ and $-a_i$, respectively.

An implementation is built by finding covers $\mathcal{C}_S$ and $\mathcal{C}_R$ as set and reset functions, obtained from the terms corresponding to the states in $GER(+a_i)$ and $GER(-a_i)$. It is possible to show the existence of an implementation in this architecture for any STG satisfying the CSC condition [4].

Example. Covers $\mathcal{C}_S$ and $\mathcal{C}_R$ for signal b in Figure 3 are obtained as follows: $\mathcal{C}_S(b) = a\bar{b}\bar{c} + a\bar{b}c + \bar{a}\bar{b}c = a\bar{b} + \bar{b}c$ and $\mathcal{C}_R(b) = \bar{a}b\bar{c}$; and yield the implementation shown in Figure 3(c). ■

The cover correctness condition in this architecture is as follows:

Definition 2.2 A set (reset) cover $\mathcal{C}_S(a_i)$ ($\mathcal{C}_R(a_i)$) is said to be correct if the only reachable states covered by $\mathcal{C}_S(a_i)$ ($\mathcal{C}_R(a_i)$) belong to $GER(+a_i)$ ($GER(-a_i)$) and the cover covers all states in this GER. □

Note that this condition does not restrict the correct cover as the one which covers GER exactly. The latter is a special case of the correct cover. An approximated cover may also include states from the DC-set. An implementation of an STG with CSC by means of exact GER covers always exists as they satisfy this correctness condition.

2.4 Atomic Complex Gate per Excitation Region

Signals in this architecture are created using *networks of atomic complex gates to implement set and reset functions of the memory element*. As a result, smaller complex gates are used which are then connected to an OR-gate whose output is in turn fed into the memory element. The basic structure of this architecture is shown in Figure 4(a). Similar to the previous architecture, the memory element can be a Muller C-element or an RS-latch.

To implement a set (reset) function, excitation regions of a signal are defined. An *excitation region* (ER) of a signal a_i, denoted as $ER(*a_i)$, is a *maximal set of connected states* of SG which have an outgoing arc labelled with $*a_i$. Obviously, if there are several $ER_j(*a_i)$ for one signal a_i, then $GER(*a_i) = \cup ER_j(*a_i)$.

Example. Signal b in STG in Figure 2(b) can be implemented in this architecture. In this case the set network contains gates corresponding to the covers obtained for two ERs of $+b$, whereas the reset cover contains only one gate: $\mathcal{C}^1_S(b) = a\bar{b}\bar{c} + a\bar{b}c = a\bar{b}$, $\mathcal{C}^2_S(b) = \bar{a}\bar{b}c$ and $\mathcal{C}_R(b) = \bar{a}b\bar{c}$. ■

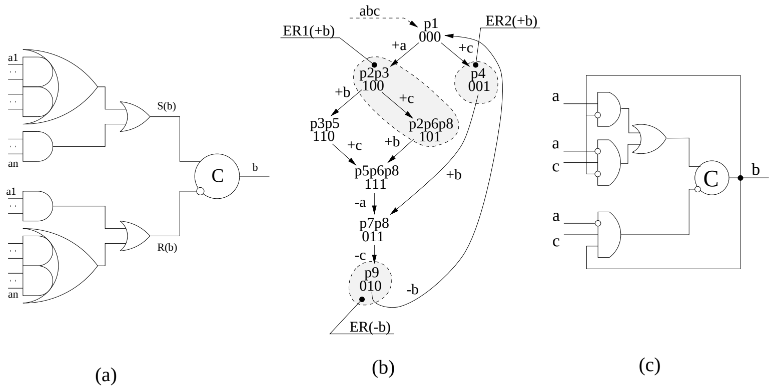

Figure 4: Atomic complex gate per excitation region architecture.

The correctness condition for this architecture is as follows:

Definition 2.3 A set of covers $\mathcal{C}_S^1(a_i), \ldots, \mathcal{C}_S^n(a_i)(\mathcal{C}_R^1(a_i), \ldots, \mathcal{C}_R^m(a_i))$ for set (reset) function of signal a_i is said to be correct if each $ER_j(+a_i)$ $(ER_j(-a_i))$ is covered by its corresponding $\mathcal{C}_S^j(a_i)$ $(\mathcal{C}_R^j(a_i))$ and the only reachable states covered by $\mathcal{C}_S^j(a_i)$ $(\mathcal{C}_R^j(a_i))$ belong to $ER_j(+a_i)$ $(ER_j(-a_i)$. □

Since the union of all unconnected ERs is a GER, then all states in GER will be covered. Similar to the previous architecture, the covers need not cover their ERs exactly. Exact covers are a special case which always satisfies this condition. In an extreme case, when the set cover consists of only one cover the OR-gate, supposed to merge the outputs of gates implementing ER covers, becomes redundant.

3 Slices in STG-unfolding segment

This section introduces definitions needed for our proposed synthesis method. First, the concept of an STG-unfolding segment is outlined. Then, we introduce the notions of cuts and slices which allows to localise the behavioural information of a particular signal instance.

3.1 STG-unfolding segment

Analysis of STGs using STG-unfolding segment was studied elsewhere [17]. An STG-*unfolding segment* for an STG G, is $G' = \langle T', P', F', \Lambda' \rangle$ where T', P' and F' are sets of transitions, places and the flow relation, respectively; and Λ' is a labelling function which labels each element of G' as an instance of elements of G. G' is a partial order obtained from an STG G by the process of its unfolding. In the unfolding the relations of *conflict*, *concurrency* and *precedence* are used to decide where to instantiate the next element. These relations are constructed during the unfolding process from the basic flow relation F', built from the flow relation F in the original STG. More formally, a transitively closed (w.r.t. F', which defines immediate predecessors of a place or transition instance) set of unfolding elements for an instance x' is called the *history of* x'. For any pair of instances $x'_1, x'_2 \in P' \cup T'$ in the unfolding three relations are defined:

- *Precedence (or Sequence)*, denoted as $x'_1 \preceq x'_2$, iff x'_1 belongs to the history of x'_2.

- *Conflict*, denoted as $x'_1 \# x'_2$, iff there exist two distinct transitions t'_1 and t'_2 in the histories of x'_1 and x'_2, respectively, such that $\bullet t'_1 \cap \bullet t'_2 \neq \emptyset$.

- *Concurrency*, denoted as $x'_1 \| x'_2$, iff x'_1 and x'_2 are neither in conflict nor in sequence.

In contrast to PN-unfolding [11], the STG-unfolding takes into account specific signal interpretation of PN transitions and keeps track of the binary codes reached by transition firing. However, it still examines only a subset of all reachable states of G and thus is more efficient than SG exhaustive analysis for a vast number of examples.

The minimal set (min-set) of transitions needed to fire t' of the STG-unfolding segment, is called *local configuration* of t' and is denoted as $\lceil t' \rceil$. A set of place instances reached by firing all transitions in $\lceil t' \rceil$ is called *postset of* $\lceil t' \rceil$ and is as denoted $\lceil t' \rceil \bullet$. Mapping a postset onto places of the original STG gives a marking of the original STG. Any non-conflicting and transitively closed w.r.t. the precedence relation set of transitions of T' is called *configuration*.

Each instance t' of STG-unfolding segment has a binary code $\xi_{\lceil t' \rceil}$ which is reached by firing transitions in $\lceil t' \rceil$. The postset $C\bullet$ and binary code ξ_C corresponding to a configuration C are calculated from $\lceil t' \rceil \bullet$ and $\xi_{\lceil t' \rceil}$ of transitions comprising it. It was shown in [17] that all states of the SG are represented in the STG-unfolding segment as postsets of some configuration.

Note that the process of constructing the STG-unfolding segment (which is a finite object for a bounded PN) is terminated at the transition instances, called *cut-off points*, whose mapped *final state* $\lceil t' \rceil \bullet$ is equal to the final state of some other instance already put into the segment. There exist several definitions of the cut-off condition [11, 17, 10, 6], different in their attempts to minimize the size of the truncated PN (or STG) unfolding necessary to fully represent the SG.

For each instance t' labelled with signal transition $*a_i$ a set of transitions $next(t')$ is defined as the set of instances labelled with $*a_i$ reachable from t' without any other intermediate transitions of a_i. Set $first(a_i)$ is the set of transitions of a_i first reached from the beginning of the segment. A special transition, called *initial transition*, is introduced in the unfolding to represent the initial state of the STG. This transition, denoted as $\perp$, has a postset which maps onto the initial marking m_0 and has an assigned binary code $\xi_{\lceil \perp \rceil}$ equal to that of the initial state v_0 of the STG.

It was demonstrated in [17] that an STG-unfolding segment can only be constructed for a bounded and consistent STG specification. The last general correctness criterion, *output semi-modularity*, can be checked on the STG-unfolding segment in linear time.

3.2 Cuts of STG-unfolding segment

To represent a state of the SG we define a cut.

Definition 3.1 *A cut of an* STG-*unfolding segment* is a maximal set of mutually concurrent places $p' \in P'$. $\square$

Each cut c thus represents some reachable marking of the original STG. A *sequence relation* is defined between two cuts $\mathsf{c}_1 \preceq \mathsf{c}_2$ if $\forall p'_i \in \mathsf{c}_2, \exists p'_j \in \mathsf{c}_1 : p'_j \preceq p'_i$. Since a cut c represents a marking, then there exists a configuration C in the STG-unfolding segment whose postset is equal to c, i.e. $C\bullet = \mathsf{c}$.

For each instance t' four types of cuts can be found. First two types are minimal cuts, i.e. cuts reached by some minimal run of the STG. The other two types of cuts are maximal cuts, i.e. cuts from which the system cannot make any further progress unless some condition is violated.

Definition 3.2 A cut $\mathsf{c}_e^{min}(t'_k)$ is called a *minimal excitation cut* of t'_k iff $\bullet t'_k \subseteq \mathsf{c}_e^{min}(t'_k)$ and $\forall t'_j, t'_j \| t'_k : (t'_j\bullet) \cap \mathsf{c}_e^{min}(t'_k) = \emptyset$. $\square$

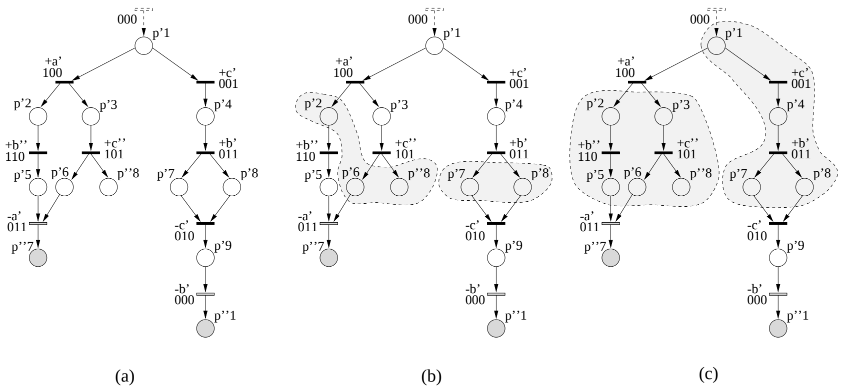

Figure 5: An example of an STG-unfolding segment and illustration of slices and cuts.

Minimal excitation cut represents the state at which t'_k becomes first enabled.

Definition 3.3 A cut $\mathsf{c}_s^{min}(t'_k)$ is called a *minimal stable cut* of t'_k iff $t'_k\bullet \subseteq \mathsf{c}_s^{min}(t'_k))$ and $\forall t'_j, t'_j \| t'_k : \ (t'_j\bullet) \cap \mathsf{c}_s^{min}(t'_k) = \emptyset$. □

Minimal stable cut represents a state which is reached by firing of t'_k enabled after firing of transitions in its $\lceil t'_k \rceil$. Minimal excitation and minimal stable cuts for one instance t'_k are two adjacent cuts which are separated by t'_k. There exists a correspondence between the cuts found in the STG-unfolding segment and the concepts of the Change Diagram theory [7]. For example, the minimal excitation cut corresponds to the minimal entry point of an ER [7] of the signal labelling t'_k.

Definition 3.4 A cut $\mathsf{c}_e^{max}(t'_k)$ is called *maximal excitation cut* of t'_k iff $\not\exists \mathsf{c}_j, \bullet t'_k \subseteq \mathsf{c}_j$: $\mathsf{c}_e^{max}(t'_k) \prec \mathsf{c}_j$. □

Maximal excitation cuts represent states from which no advancement can be made unless t'_k is fired.

Definition 3.5 A cut $\mathsf{c}_s^{max}(t'_k)$ is called *maximal stable cut* of t'_k iff $\mathsf{c}_s^{min}(t'_k) \preceq \mathsf{c}_s^{max}(t'_k)$ and $\not\exists t'_j \in next(t'_k) : \ \exists \mathsf{c}_j, \mathsf{c}_e^{min}(t'_j) \preceq \mathsf{c}_j : \ \mathsf{c}_j \preceq \mathsf{c}_s^{max}(t'_k)$. □

A special case is a maximal stable cut for the initial transition $\perp$. In this case the set of first instances of a_i, $first(a_i)$, is taken instead of $next(t'_k)$. Maximal stable cuts represent states which are reached after firing of t'_k from which firing of some transition leads to a state enabling the next change of the signal a_i labelling t'_k.

Finally, note that each instance t'_k of the STG-unfolding segment uniquely identifies $\mathsf{c}_e^{min}(t'_k)$ and $\mathsf{c}_s^{min}(t'_k)$ and the *sets* of $\mathsf{c}_e^{max}(t'_k)$ and $\mathsf{c}_s^{max}(t'_k)$.

Example. Several cuts are illustrated in Figure 5. Consider a cut $\mathsf{c} = (p'_7, p'_8)$ in Figure 5(b). This cut is a minimal excitation cut for the transition $-c'$ and is a minimal stable cut for $+b'$. Another cut, $\mathsf{c} = (p'_2, p'_6, p''_8)$ is a maximal stable cut for transition instance $+a'$. At the same time this is a maximal excitation cut for the instance $+b''$. ■

3.3 Slices of STG-unfolding segment

To represent a connected set of states we introduce the notion of slice.

Definition 3.6 A *slice of an* STG-*unfolding segment* is a pair $\mathcal{S} = \langle \mathsf{c}^{min}, \mathsf{C}^{max} \rangle$ defined by the *min-cut of the slice* c^{min}, and a *set of max-cuts* C^{max}; such that $\forall \mathsf{c}_i \in \mathcal{S}$

1. $\mathsf{c}^{min} \preceq \mathsf{c}_i$ and $\exists \mathsf{c}_j \in \mathsf{C}^{max} : \mathsf{c}_i \preceq \mathsf{c}_j$, and
2. no two cuts in the set of max-cuts in C^{max} are mutually ordered by the sequence relation.

□

In other words, a slice is defined between one min-cut and a set of max-cuts. Every cut in between the min-cut and a max-cut is encapsulated in the slice $\mathcal{S}$. Furthermore, for any two cuts c_i and c_j, if $\mathsf{c}_i \prec \mathsf{c}_j$, then all cuts between c_i and c_j are also encapsulated by $\mathcal{S}$. Since each cut represents some state in the SG, for any two states s_i and s_j represented as sequential cuts in $\mathcal{S}$, all states on any path from s_i to s_j are also represented as cuts encapsulated into $\mathcal{S}$. The number of cuts in the set of max-cuts corresponds to the number of configurations (non-conflicting runs of the STG) which include configuration producing the min-cut. The elements of the STG-unfolding segment, i.e. places, transitions and arcs, bounded by instances in min-cut and max-cuts are said to belong to the slice.

Example. Slice $\mathcal{S}_1 = \langle (p'_1), \{(p'_7, p'_8)\} \rangle$ in Figure 5(c) encapsulates cut $\mathsf{c} = (p'_4)$. Another slice $\mathcal{S}_2$ is defined between a min-cut (p'_2, p'_3) and a set of max-cuts $\{(p'_5, p'_6, p''_8)\}$ and includes all cuts between them. It is also possible to define a slice between (p'_2, p'_3) and $\{(p'_2, p'_6, p''_8), (p'_3, p'_5)\}$. In this case the slice will include all cuts but the one enabling $-a'$. ■

Each cut is produced by some configuration of the STG-unfolding segment. Thus the binary codes of the SG states represented by cuts encapsulated in a particular slice can be recovered by examining its cuts.

4 Obtaining exact covers from STG-unfolding segment

To implement an output signal in any architecture, specific subsets of states from the SG are required. A slice represents a set of states of the SG in form of the STG-unfolding segment G'. Therefore, the problem is posed as finding a partitioning of the segment into slices representing appropriate states. Once a partitioning was found, the set of states is extracted from the slices by examining all cuts encapsulated by each slice. Required covers are then found as a union of minterms corresponding to the cuts encapsulated by each slice.

To define each slice we need to identify its min-cut and a set of max-cuts. Consider finding these for each architecture.

4.1 Atomic Complex Gate per Signal

As it was shown in [18], the set of slices representing states in the on-set (off-set) can be identified on the STG-unfolding segment using the instances of signal transitions $*a'_i$ and an initial transition $\perp$. The initial transition is used when the signal a_i is at "1" ("0") in the initial state of the STG. The slices are bounded by the minimal excitation slice and a set of maximal stable cuts uniquely determined by each instance. Thus the On-set and Off-set partitioning in an STG-unfolding segment are defined as follows.

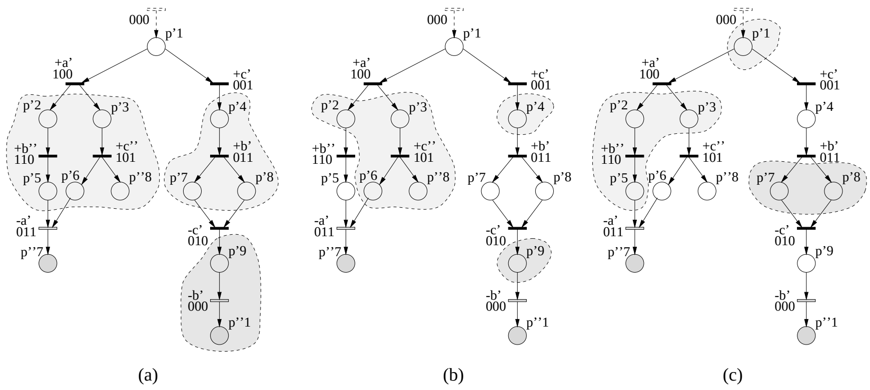

Figure 6: Illustration of synthesis from the STG-unfolding segment.

Definition 4.1 A set of slices $\mathbb{S}_{On}$ ($\mathbb{S}_{Off}$) in an STG-unfolding segment G' is called *On-set (Off-set) partitioning of G' w.r.t. a signal a_i* iff for each instance $+a'_i$ ($-a'_i$) its corresponding slice $\mathcal{S}^j_{On}(+a'_i)$ ($\mathcal{S}^j_{Off}(-a'_i)$) is defined with $\mathsf{c}_e^{min}(+a'_i)$ ($\mathsf{c}_e^{min}(-a'_i)$) as its min-cut and $\mathsf{C}_s^{max}(+a'_i)$ ($\mathsf{C}_s^{max}(-a'_i)$) as a set of its max-cuts.

If signal a_i is at "1" ("0") in the initial state of the STG, then the set of slices $\mathbb{S}_{On}$ ($\mathbb{S}_{Off}$) also includes slice defined with $\mathsf{c}_s^{min}(\perp)$ as a min-cut and $\mathsf{C}_s^{max}(\perp)$ as its set of max-cuts. □

The on-set cover is obtained from the On-set partitioning by extracting the binary codes corresponding to the cuts encapsulated by the slices:

$$\mathcal{C}_{On} = \sum \mathcal{C}_{On}(+a'_i), \text{ where } \mathcal{C}_{On} = \bigcup \xi_{C_j},\ C_j\bullet = \mathsf{c}_j \in \mathcal{S}_{On}(+a'_i)$$

($\mathcal{S}_{On}(\perp)$ is also used, if required). In other words, the on-set cover is found as the union of all covers found for the slices of on-set partitioning. Each slice cover is obtained by taking the union of all minterms corresponding to the binary codes of configurations C_j whose postset is a cut c_j encapsulated by $\mathcal{S}_{On}(+a'_i)$. Slice $\mathcal{S}_{On}(+a'_i)$ is called *on-set slice.*

Example. Consider synthesising signal b from an example in Figure 5. The on-set partitioning of the STG-unfolding segment is shown in Figure 6(a). There are two instances $+b'$ and $+b''$ and one instance $-b'$. Thus there are two slices $\mathcal{S}^1_{On}(+b') = \langle (p'_4), \{(p'_7, p'_8)\}\rangle$ and $\mathcal{S}^2_{On}(+b'') = \langle (p'_2, p'_3), \{(p'_5, p'_6, p''_8)\}\rangle$ representing states from the on-set and one slice $\mathcal{S}_{Off} = \langle (p'_9), \{(p''_1)\}\rangle$. The on-set cover is obtained from slices as $\mathcal{C}_{On} = \{100, 101, 110, 111\} \cup \{001, 011\}$ which after standard boolean transformation gives $\mathcal{C}_{On} = \{1--, --1\} = a + c$. If the off-set implementation were chosen, then the cover would be $\mathcal{C}_{Off} = \{010, 000\} = \bar{a}\bar{c}$. ■

4.2 Atomic Complex Gate per Excitation Function

A set of slices representing the GERs of $*a_i$ is found in the STG-unfolding segment using the instances of this signal $*a'_i$ [18]. Any cut representing a state belonging to GER of $*a_i$ is encapsulated between the minimal excitation cut and the set of maximal excitation cuts, uniquely identified by one of its instances $*a'_i$. By finding all slices representing states in GER we obtain a cover satisfying the correctness condition for this architecture.

Definition 4.2 A set of slices $\mathbb{S}_S(a_i)$ ($\mathbb{S}_R(a_i)$) in an STG-unfolding segment G' is called *Set (Reset) partitioning of the* STG-*unfolding segment G' w.r.t. signal a_i* iff for each instance $+a'_i$

$(-a_i')$ its corresponding slice $\mathcal{S}_e(+a_i')$ ($\mathcal{S}_e(-a_i')$) is defined with $\mathsf{c}_e^{min}(+a_i')$ ($\mathsf{c}_e^{min}(-a_i')$) as its min-cut and $\mathsf{C}_e^{max}(+a_i')$ ($\mathsf{C}_e^{max}(-a_i')$) as a set of its max-cuts. □

Each slice $\mathcal{S}_e(*a_i')$ is called *excitation slice of* $*a_i'$. Similar to the previous architecture, the cover for set function is obtained from the binary codes corresponding to the cuts encapsulated by slices:

$$\mathcal{C}_S(a_i) = \sum \mathcal{C}_S(+a_i'), \text{ where } \mathcal{C}_S(+a_i') = \bigcup_j \xi_{C_j},\ C_j\bullet = \mathsf{c}_j :\ \mathsf{c}_j \in \mathcal{S}_e(+a_i').$$

As in the previous architecture, the covers are found by taking the union of minterms corresponding to the binary codes of configurations whose postsets are the cuts encapsulated by slices $\mathcal{S}_e^k(+a_i')$. The reset function is obtained similarly using the $-a_i'$ instances.

Example. Consider again synthesis of signal b from example in Figure 6(b). For each instance on b the excitation slices are found as $\mathcal{S}_e(+b') = \langle(p_2', p_3'), \{(p_2', p_6', p_8'')\}\rangle$ and $\mathcal{S}_e(+b'') = \langle(p_4'), \{(p_4')\}\rangle$ for GER of $+b$ and $\mathcal{S}_e(-b') = \langle(p_9'), \{(p_9')\}\rangle$ for the opposite signal transition. After recovering the binary codes the covers for set and reset functions are: $\mathcal{C}_S = \{100, 101\} \cup \{001\} = \{10-, -01\} = a\bar{b} + \bar{b}c$ and $\mathcal{C}_R = \{010\} = \bar{a}b\bar{c}$. ■

4.3 Atomic Complex Gate per Excitation Region

The partitioning for this architecture is the same as for the Atomic Complex Gate per Excitation Function. The interpretation of the covers found for each slice is, however, different. Only covers whose intersection is non-empty represent one connected ER. Thus, after the covers for all excitation slices $\mathcal{S}_e(*a_i')$ are found the final implementation is obtained as a set of covers [18] corresponding to each connected ER:

$$\text{for each } ER_k(*a_i):\ \mathcal{C}_S^k(*a_i) = \sum \mathcal{C}_e(*a_i')$$

where

$$\forall \mathcal{C}_e(*a_i'), \mathcal{C}_e(*a_i'') \text{ iff } \mathcal{C}_e(*a_i') \in \mathcal{C}_S^k(*a_i) \text{ and } \mathcal{C}_e(*a_i') \cdot \mathcal{C}_e(*a_i'') \neq \emptyset \text{ then } \mathcal{C}_e(*a_i'') \in \mathcal{C}_S^k(*a_i).$$

That is, if any two covers $\mathcal{C}_e(*a_i')$ and $\mathcal{C}_e(*a_i'')$ found for two slices $\mathcal{S}_e(*a_i')$ and $\mathcal{S}_e(*a_i'')$ intersect, then they cover states belonging to one connected ER.

Notably, a *fake conflict* [10] situation may result in two slices with non-intersecting slices although the they represent one ER. This situation is detected during STG verification and can be avoided by forcing the union of such covers, although we do not discuss it in detail here.

Example. Coming back to the implementation of signal b, there are two excitation slices for $+b$. Thus all three ERs (two for $+b$ and one for $-b$) can be implemented as separate complex gates in the set and reset functions respectively. On the other hand, consider implementation of c (Figure 6(c)). There are also two excitation slices for $+c$: $\mathcal{S}_e(+c') = \langle(p_1'), \{(p_1')\}\rangle$ and $\mathcal{S}_e(+c'') = \langle(p_2', p_3'), \{(p_3', p_5')\}\rangle$. However, $+c'$ and $+c''$ are in fake conflict and these two slices represent one ER which is covered by $\mathcal{C}_S(c) = \bar{b}\bar{c} + a\bar{c}$. This, along with the reset cover $\mathcal{C}_R(c) = \bar{a}bc$ obtained from the excitation slice of $-c'$, completes the implementation in this architecture. ■

5 Deriving covers from STG-unfolding segment efficiently

The synthesis procedure described in the previous Section has one drawback. If many concurrent transitions belong to a slice, then examining all cuts will suffer from exponential explosion of states. Recall that the correctness criteria for each architecture allow the implementing cover to be greater than the exact cover. This can be exploited in an approximation method for covering the desired slices (on-, off- and excitation slices). In this section we describe the approximation approach. First, we examine possible strategies for this approach in the STG-unfolding segment. Then we describe its key procedures: the *initial approximation of the cover function* and the *refinement*.

5.1 Strategies for deriving the covers

There are two specific sets of reachable states for each cover $\mathcal{C}$ satisfying the correctness requirement:

- *Positive set*, denoted as $\mathcal{P}$-set, which is a set of states which $\mathcal{C}$ **must** cover; and
- *Negative set*, denoted as $\mathcal{N}$-set, which is a set of states which $\mathcal{C}$ **must not** cover.

The choice of the $\mathcal{P}$-set and $\mathcal{N}$-set comes from the cover correctness conditions and is made for the architectures as follows:

- ACGpS implementation: $\mathcal{P}$-set is taken as the on-set and $\mathcal{N}$-set is taken as the off-set of a particular signal a_i;
- ACGpEF and ACGpER implementations: $\mathcal{P}$-set is taken as the $GER(+a_i)$ ($GER(-a_i)$) and $\mathcal{N}$-set is taken as the rest of reachable states for the set (reset) function of a particular signal a_i.

It is convenient to introduce a set of states capturing the states in which a signal is stable. The *generalised quiescent region of* $+a_i$ *(*$-a_i$*)*, denoted $GQR(+a_i)$ ($GQR(-a_i)$), is a set of states at which signal a_i is stable at "1" ("0"). Then the $\mathcal{N}$-set for the set (reset) function in ACGpEF and ACGpER implementations is chosen as $GQR(+a_i) \cup GER(-a_i) \cup GQR(-a_i)$ ($GQR(-a_i) \cup GER(+a_i) \cup GQR(+a_i)$).

The cover is obtained by finding the partitioning representing the $\mathcal{P}$-set. However, the $\mathcal{N}$-set also corresponds to some slices. The interpretation of both sets and the covers and slices in the STG-unfolding segment suggests (at least) two possible strategies for deriving the cover.

5.1.1 Negative set approximation

This strategy assumes that the approximation is found from the STG-unfolding segment not only for $\mathcal{C}^*_{\mathcal{P}}$, covering the states in the $\mathcal{P}$-set, but also for $\mathcal{C}^*_{\mathcal{N}}$ which covers the states in the $\mathcal{N}$-set. Thus the partitioning of the segment for $\mathcal{C}_{\mathcal{N}}$ is also required. Assume that the approximations $\mathcal{C}^*_{\mathcal{P}}$ and $\mathcal{C}^*_{\mathcal{N}}$ were constructed so that all states in $\mathcal{P}$-set and $\mathcal{N}$-set, respectively, are covered. By making sure that $\mathcal{C}^*_{\mathcal{P}} \cdot \mathcal{C}^*_{\mathcal{N}} = \emptyset$ we guarantee that no state from $\mathcal{N}$-set is covered by $\mathcal{C}^*_{\mathcal{P}}$. However, if the intersection of approximations $\mathcal{C}^*_{\mathcal{P}}$ and $\mathcal{C}^*_{\mathcal{N}}$ is not empty, then the cover approximations must be *refined*. The refinement procedure must produce covers which cover less number of combinations. Eventually, in the worst case, after full refinement, it must produce exact covers for $\mathcal{P}$-set and $\mathcal{N}$-set. If the intersection of $\mathcal{P}$-set and $\mathcal{N}$-set is empty, then the exact covers for these sets will also have empty intersection. If the STG does not satisfy CSC condition, then the intersection of $\mathcal{P}$-set and $\mathcal{N}$-set will be non-empty. Hence, if the refinement procedure terminates with fully refined covers but their intersection is still non-empty, then this STG does not satisfy the CSC condition.

This strategy finds approximations $\mathcal{C}^*_{\mathcal{P}}$ and $\mathcal{C}^*_{\mathcal{N}}$ which cover the required sets of states and partition the set of combinations which are allowed to be covered by them into two disjoint sets. It, therefore, produces *pessimistic covers* as it does not allow the combinations from the DC-set to be shared by both covers. The pseudo-code of the procedure implementing this strategy is shown in Figure 7(a).

For example, in ACGpS architecture the $\mathcal{P}$-set and $\mathcal{N}$-set are taken as the on- and off-set. On- and off-set are disjoint by construction for a CSC-compliant STG. If approximations $\mathcal{C}^*_{On}$ and $\mathcal{C}^*_{Off}$ cover on- and off-set respectively, and $\mathcal{C}^*_{On} \cdot \mathcal{C}^*_{Off} = \emptyset$, then these covers satisfy the correctness criterion set out in Definition 2.1.

> **proc** *Find cover($\mathcal{P}$-set, $\mathcal{N}$-set)*
>   *Find initial approximations $\mathcal{C}^*_{\mathcal{P}}$ and $\mathcal{C}^*_{\mathcal{N}}$*
>   **while** $\mathcal{C}^*_{\mathcal{P}} \cdot \mathcal{C}^*_{\mathcal{N}} \neq \emptyset$ **do**
>   **for each** x', x'' used to find $\mathcal{C}^*_{\mathcal{P}}$ and $\mathcal{C}^*_{\mathcal{N}}$ **do**
>   *Refine cover* (x', x'')
>   **end do**
>   **end do**
>   **return** $\mathcal{C}^*_{\mathcal{P}}$
> **end do**

(a)

> **proc** *Find cover($\mathcal{P}$-set, $\mathcal{N}$-set)*
>   *Find initial approximation $\mathcal{C}^*_{\mathcal{P}}$*
>   *Evaluate $\mathcal{C}^*_{\mathcal{P}}$*
>   **while** some cut from $\mathcal{S}$ is in $\mathcal{N}$-set **do**
>   *Refine $\mathcal{C}^*_{\mathcal{P}}$*
>   *Evaluate $\mathcal{C}^*_{\mathcal{P}}$*
>   **end do**
>   **return** $\mathcal{C}^*_{\mathcal{P}}$
> **end do**

(b)

Figure 7: Procedures for two cover deriving strategies

5.1.2 Positive set cover evaluation

This strategy assumes that only the approximation $\mathcal{C}^*_{\mathcal{P}}$ covering the $\mathcal{P}$-set is found in the STG-unfolding segment. The cover $\mathcal{C}^*_{\mathcal{P}}$ is then *evaluated* to by finding where in the STG-unfolding segment it becomes TRUE. The evaluation finds a set of slices $\mathcal{S}$ which represent all states in which the cover becomes TRUE. If no cut in $\mathcal{S}$ represents a states from the $\mathcal{N}$-set, then the cover satisfies the correctness criterion. The pseudo-code of the procedure for this strategy is shown in Figure 7(b). Otherwise, approximation $\mathcal{C}^*_{\mathcal{P}}$ is refined until no cut represents a state from the $\mathcal{N}$-set. When $\mathcal{C}^*_\mathcal{P}$ is fully refined but such a cut still exists, the refining procedure aborts and reports the CSC violation.

For example, suppose that $\mathcal{C}^*_S(a_i)$ was found for the ACGpEF implementation. Any state s_j from the $\mathcal{N}$-set $= GQR(+a_i) \cup GER(-a_i) \cup GQR(-a_i)$ falls into one of the two following categories:

- s_j belongs to $GQR(+a_i) \cup GER(-a_i)$, in which case the value of the element $v_j[i] = 1$ is opposite to the value of $v_k[i] = 0$ of the state belonging to the $\mathcal{P}$-set $= GER(+a_i)$; or

- s_j belongs to $GQR(-a_i)$, in which case the values of $v_j[i]$ and $v_k[i]$ are equal, but s_j does not enable a_i.

Thus the cover correctness is checked by examining the cuts c_k of the slices $\mathcal{S}$ and checking that s_k represented by c_k does not satisfy either of the above conditions.

The evaluation of the cover is done for each cube of the cover. It may use a branch-and-bound algorithm which splits a slice (the whole segment in the beginning) into subslices according to the literals present in each cube.

In this paper we concentrate on the first approximation strategy.

5.2 Cover approximation

The slices in the STG-unfolding segment represent two types of states: those where a signal transition $*a_i$ is excited ($GER(*a_i)$) and those in which the signal a_i is stable ($GQR(*a_i)$). At the STG-unfolding level, we have instances of places and transitions. We can construct our slice cover approximations by means of the cover approximation for the place and transition instances which belong to the slice. The cover approximations can be divided into those for place instances p', representing the cuts containing these instances, and for transition instances t', representing the cuts enabling t'. Obviously, it is possible to find an *exact cover for an instance* x' which will only cover cuts including p' or enabling t'.

5.2.1 Cover approximation for transition instance

Consider, first, obtaining cover for a transition instance $*a'_i$ in the STG-unfolding segment. All cuts in which $*a'_i$ is excited are represented in its excitation slice $\mathcal{S}_e(*a'_i)$. Any cut inside $\mathcal{S}_e(*a'_i)$, reachable from the minimal excitation cut $\mathsf{c}_e^{min}(*a'_i)$, can only be reached by firing transitions which are concurrent to $*a'_i$. If a signal instance $*a'_j$ is concurrent to $*a'_i$, then the value of its corresponding element in the binary code may take values of both "0" and "1". The *transition instance cover approximation* $\mathcal{C}^*_e(a'_i)$ is found from the binary code ξ assigned to the cut $\mathsf{c}_e^{min}(*a'_i)$. The literals corresponding to the signals whose instances belong to $\mathcal{S}_e(*a'_i)$ and are concurrent to $*a'_i$ are substituted by "–" (don't care). Approximation reduces the number of literals in cover $\mathcal{C}^*_e(*a'_i)$ and increases the number of combinations covered by $\mathcal{C}^*_e(*a'_i)$.

Example. Consider the cover approximation for $+d'$ in Figure 8(a). The binary code of its minimal excitation cut $\mathsf{c}_e^{min}(+d') = (p'_2, p'_3, p'_4)$ is found from the binary code of its local configuration as $\xi = \{1000000\}$ (the order of signals is $abcdefg$). There are four signals $\{b, c, e, f\}$ whose instances belong to the slice and are concurrent to $+d'$. Thus the cover approximation for $+d'$ will be $\mathcal{C}^*_e(+d') = \{1--0--0\} = a\bar{d}\bar{g}$. ■

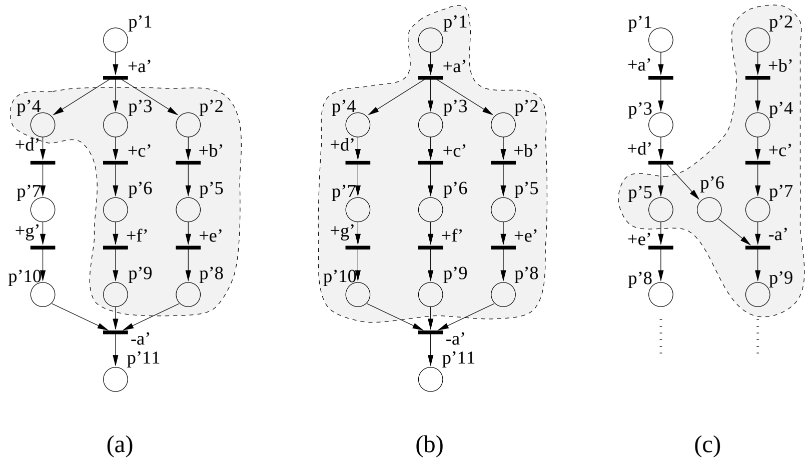

Figure 8: Illustration of cover approximation and refinement.

Property 5.1 The cover approximation $\mathcal{C}_e^*(+a_i')$ ($\mathcal{C}_e^*(-a_i')$) obtained from the STG-unfolding segment of an STG satisfying the general correctness criteria always has the value of the element corresponding to a_i set to "0" ("1").

Proof: Follows from the fact that in an STG-unfolding segment constructed for an STG satisfying general correctness criteria no two instances of a_i can be concurrent. Thus the element corresponding to a_i will remain equal to that of $\xi[i]$ assigned to the minimal excitation cut $\mathsf{c}_e^{min}(+a_i')$ ($\mathsf{c}_e^{min}(-a_i')$), i.e. "0" ("1"). □

5.2.2 Cover approximation for place instance

The *place instance cover approximation* $\mathcal{C}_m^*(p')$ for each p' is obtained from the binary code $\xi_{\lceil t_k' \rceil}$ assigned to its preceding transition. Similar to transition instance cover approximation, any marking at which p' is marked can only be reached by firing transitions concurrent to p'. The literals corresponding to signals whose instances belong to $\mathcal{S}$ and are concurrent to p' are replaced by "–".

A place cover approximation must not cover markings outside the slice $\mathcal{S}$. The set of concurrent transitions is chosen so that it never leads to a cut which is outside the bounds of $\mathcal{S}$, i.e. its min-cut and the set of max-cuts. There may be several max-cuts. Thus for a place belonging to a max-cut, there may be several possibilities to choose the set of concurrent transitions. In this case, the place cover approximation is found as the union of all such possibilities: $\mathcal{C}_m^*(p_l') = \sum \mathcal{C}_{t_k'}^*(p_l')$, where t_k' is the excluded concurrent transition.

Example. Consider approximation of the cover for places belonging to $\mathcal{S}(+a') = \langle (p_1'), \{(p_7', p_8', p_9'), (p_6', p_8', p_{10}'), (p_5', p_9', p_{10}')\}\rangle$in the example shown in Figure 8(b). The approximation cover for p_4' is found from $\xi_{\lceil +a' \rceil}$ as $\mathcal{C}_m^*(p_4') = \{1 - -0 - -0\} = ad\bar{g}$. Place p_{10}', on the other hand, belongs to at least one max-cut. Thus its approximation cover is found using two approximations corresponding to two different sets of concurrent transitions:

$$\begin{aligned} \mathcal{C}_m^*(p_{10}') &= \mathcal{C}_{f'}^*(p_{10}') \ \ (\text{for } \{b', c', e'\}) \\ &+ \mathcal{C}_{e'}^*(p_{10}') \ \ (\text{for } \{b', c', f'\}) \\ &= \{1 - -1 - 01\} \cup \{1 - -10 - 1\} = ad\bar{f}g + ad\bar{e}g. \end{aligned}$$

■

Property 5.2 The cover approximation $\mathcal{C}_m^*(p')$ obtained from an on-set (off-set) slice $\mathcal{S}_{On}(+a_i')$ ($\mathcal{S}_{Off}(+a_i')$) in the STG-unfolding segment of an STG satisfying the general correctness criteria always has the value of the element corresponding to a_i set to "1" ("0").

Proof: Follows from the fact that the on-set (off-set) slice contains no instances of a_i except for $+a_i'$ ($-a_i'$), which is used to define it. Thus there are no instances of a_i in the slice which are concurrent to p'. Since p' is sequential to $+a_i'$ ($-a_i'$), the value of the corresponding element will remain the same as $\xi_{\lceil +a_i' \rceil}[i] = 1$ ($\xi_{\lceil -a_i' \rceil}[i] = 0$) corresponding to the minimal stable cut of $+a_i'$ ($-a_i'$). □

5.2.3 Finding $\mathcal{P}$-set and $\mathcal{N}$-set approximations

Since our strategy requires approximation covers for both $\mathcal{P}$-set and $\mathcal{N}$-set, let us consider finding those for each implementation architecture.

ACGpS implementation After the On- and Off-set partitioning w.r.t signal a_i was found, the slices $\mathcal{S}_{On}(+a_i')$ and $\mathcal{S}_{On}(-a_i')$ represent the cuts starting from those enabling each instance $*a_i'$ until (but not including) the cuts enabling instances from $next(*a_i')$. To approximate the cuts enabling $*a_i'$ we use the transition instance approximation cover $\mathcal{C}_e(*a_i')$. To find the approximation for the rest of cuts in $\mathcal{S}_{On}(+a_i')$ and $\mathcal{S}_{On}(-a_i')$ we use a set of place instances which belong to $\mathcal{S}_{On}(+a_i')$ and $\mathcal{S}_{On}(-a_i')$, respectively.

For simplicity we only consider the on-set cover. The off-set cover approximation is found by using the instances of $-a_i$.

A cover approximation for a particular place p_l' will cover all states at which p_l' is marked with any other concurrent place p_j'. Therefore, only mutually non-concurrent subset of places belonging to $\mathcal{S}_{On}(+a_i')$ can be considered. A set of such places is found in the STG-unfolding segment.

Definition 5.1 A set of places P_a' belonging to the slice $\mathcal{S}_{On}(+a_i')$ is called an *approximation set of the slice* $\mathcal{S}_{On}(+a_i')$ iff $\forall p_k' \in P_a' : +a_i' \prec p_k'$ and $\forall p_k', p_l' \in P_a' : p_k' \not\parallel p_l'$. □

The approximation set for a slice is a "skeleton" of places for this slice which are either sequential or in conflict with each other. Any cut encapsulated by $\mathcal{S}_{On}(+a_i')$ will contain a place from its approximation set.

The on-set cover approximation is found from $\mathcal{S}_{On}(+a_i')$ as

$$\mathcal{C}_{On}^*(+a_i') = \mathcal{C}_e^*(+a_i') + \sum \mathcal{C}_m^*(p_l'),\ p_l' \in P_a'$$

and the whole on-set cover is found as:

$$\mathcal{C}_{On}^*(a_i) = \sum \mathcal{C}_{On}^*(+a_i').$$

Proposition 5.1 For any STG satisfying the general correctness criteria, the on-set cover approximation $\mathcal{C}_{On}^*(a_i)$, obtained from its STG-unfolding segment, covers all reachable states in $On(a_i)$.

Proof: Since there are two types of states in $On(a_i)$, the proof is split into two parts: for those states where $+a_i$ is excited and for those states where a_i is stable. The proof for the former (represented by $\mathcal{S}_e(+a_i')$ which is included into $\mathcal{S}_{On}(+a_i')$) is given later in Proposition 5.2. Hence, we give here only the proof that a cut corresponding to any state from $On(a_i)$ at which a_i is stable is covered by its $\mathcal{C}_{On}^*(a_i)$.

Any reachable state of an STG is represented in the STG-unfolding segment. Therefore, for any state in $On(a_i)$ there exists a corresponding cut in the segment. Any state in $On(a_i)$ with stable a_i is sequential to some transition of $+a_i$. Thus any such a state is represented in the STG-unfolding segment as a cut sequential to minimal stable cut of some $+a_i'$. Consider the set of places belonging to this cut and sequential to $+a_i'$. One of these places, say p', will be chosen in to the approximation set P_a'. The cuts containing p' are formed by different orders of firing of the concurrent transitions. The p' cover approximation ignores any ordering and, therefore, will cover cuts obtained by any ordering existing in the segment. Thus p' cover approximation will cover all cuts of $\mathcal{S}_{On}(+a_i')$ which contain p' and represent states from $On(a_i)$. □

ACGpEF and ACGpER implementations Let us consider only calculation of the set function. For the reset function the instances of $-a_i$ are used. The $\mathcal{P}$-set and $\mathcal{N}$-set in these architectures are $\mathsf{GER}(+a_i)$ and the rest of reachable states, respectively. In the discussion above, for the ACGpS implementation, we used two separate approximations for states in which $+a_i$ is excited and in which a_i is stable. Thus the $\mathcal{P}$-set and $\mathcal{N}$-set cover approximations can be found by obtaining the On- and Off-set partitioning of the STG-unfolding segment.

The set cover approximation, covering the $\mathcal{P}$-set, is then found from the approximations of the excitation slices as:

$$\mathcal{C}_S^*(a_i) = \sum \mathcal{C}_e^*(+a_i')$$

Proposition 5.2 For any STG satisfying the general correctness criteria, the set cover approximation $\mathcal{C}_S^*(a_i)$, obtained from its STG-unfolding segment, covers all reachable states where $+a_i$ is excited.

Proof: First, we show that each state is represented as some cuts encapsulated by excitation slices found for $+a_i$. The STG-unfolding segment represents all reachable states and all instances of signal transitions. Therefore, if there exists a reachable state enabling $+a_i$, it is represented in the STG-unfolding segment. Furthermore, there will exist a corresponding instance $+a_i'$. Hence there will exist a slice $\mathcal{S}_e(+a_i')$ defined by the minimal excitation and minimal stable cuts of $+a_i'$. Thus for any state in which $+a_i$ is excited there exists a cut encapsulated by $\mathcal{S}_e(+a_i')$.

Now we show that the approximation for any slice $\mathcal{S}_e(+a_i')$ does not miss out any states from corresponding $ER(+a_i)$. The set of states in which $+a_i$ is enabled is formed by particular orderings of firings of concurrent transition instances belonging to $\mathcal{S}_e(+a_i')$. The approximation uses all concurrent instances in $\mathcal{S}_e(+a_i')$ and ignores the orderings. Therefore, all real orderings are included and all states in $ER(+a_i)$ are covered. □

The cover approximation for the $\mathcal{N}$-set is found as:

$$\mathcal{C}_{\mathcal{N}}^* = \sum \mathcal{C}_m^*(p_l') + \sum \mathcal{C}_e^*(-a_i') + \sum \mathcal{C}_m^*(p_k')$$

where p_l' and p_k' belong to the approximation sets found for On- and Off-set partitioning, respectively, as in the previous architecture.

Correctness of the approximation strategy Here we demonstrate that the Negative set approximation strategy produces a correct implementation for any STG satisfying CSC.

Proposition 5.3 Let the $\mathcal{P}$-set and the $\mathcal{N}$-set be chosen as on- and off-set of a_i, respectively. Let the cover approximations $\mathcal{C}_{On}^*(a_i)$ and $\mathcal{C}_{Off}^*(a_i)$ be calculated from the segment for a

CSC-compliant STG in the above way. Then the Negative set approximation procedure given in Figure 7(a) produces correct covers for ACGpS architecture iff the refinement procedure in the worst case restores the exact covers for On- and Off-set partitioning after a finite number of iterations.

Proof: (1) According to Proposition 5.1 the above approximation method produces covers which cover the on- and off-set of a_i. (2) The procedure in Figure 7(a) exits only if the covers are disjoint or terminates if the STG does not have CSC. For a CSC-compliant STG the on- and off-sets of a_i are disjoint. Therefore, if the refinement procedure in the worst case restores exact covers for each on- and off-slice of the partitioning, then the states covered by these restored covers will belong only to on- and off-set, respectively. Thus the refined covers will satisfy the correctness criterion for ACGpS architecture (Definition 2.1). □

The next proposition illustrates that covers found for ACGpEF and ACGpER using our strategy are also correct.

Proposition 5.4 Let the $\mathcal{P}$-set and the $\mathcal{N}$-set be chosen as $GER(+a_i)$ and the rest of reachable states, respectively. Let the cover approximations $\mathcal{C}^*_e(+a_i)$ and $\mathcal{C}^*_\mathcal{N}$ be calculated from the segment for a CSC-compliant STG as above for ACGpEF and ACGpER architectures. Then the Negative set approximation procedure given in Figure 7(a) produces correct covers for ACGpEF and ACGpER architectures iff the refinement procedure in the worst case restores the exact covers for transition and place instances used to find $\mathcal{C}^*_\mathcal{P}(a_i)$ and $\mathcal{C}^*_\mathcal{N}(a_i)$ in a finite number of iterations.

Proof: (1) According to Proposition 5.2 the above approximation method produces covers which cover $GER(+a_i)$. (2) There are three components in the $\mathcal{N}$-set cover approximation $\mathcal{C}^*_\mathcal{N}$. From the properties of transition and place instance cover approximations it follows that the cover approximations $\sum \mathcal{C}^*_m(p'_l)$ and $\sum \mathcal{C}^*_e(-a'_i)$ will have an element corresponding to a_i set to "1". This element will always be set to "0" in the $\mathcal{P}$-set cover $\mathcal{C}^*_e(+a_i)$. Therefore, the only intersection between $\mathcal{P}$-set and the $\mathcal{N}$-set cover approximations may happen if some $\mathcal{C}^*_e(+a'_i)$ intersects with $\sum \mathcal{C}^*_m(p'_k)$. The former corresponds to the states where signal transition $+a_i$ is excited and the latter to those where a_i is stable at "0". In a CSC-compliant STG intersection between these two sets of states is empty. Thus, if the refinement procedure in the worst case produces exact covers, then these covers satisfy the correctness criteria for these architectures (Definitions 2.2 and 2.3, respectively). □

Note that the above proposition also showed that only intersection between the cover approximations for the excitation slices and the places of the Off-set partitioning needs to be checked when the set function is found for the implementation in the ACGpEF or ACGpER architectures. This reduces the number of cover intersections to be checked and, hence, reduces the synthesis time.

5.3 Cover refinement

The purpose of the refining procedure is to restore some of the relations between concurrent transitions belonging to a slice $\mathcal{S}$ which were ignored during the approximation. The refinement is only performed when an intersection of two covers for some instances x' and x'' is non-empty. An intersection can be done on the "per cube" basis, which allows us to narrow the intersection to a particular pair of cubes. If the intersection of two cubes is non-empty, the set of the offending signals Sig can be found. These are the signals whose corresponding literals are missing from a particular cube of the cover.

The refinement of cover for element x' is based on finding the refinement set for x' w.r.t. an offending signal a_j.

Definition 5.2 A set of place P'_r places belonging to the slice $\mathcal{S}$ is called the *refining set of* x' *w.r.t.* a_j iff it satisfies the following: $\forall p'_k \in P'_r: \; x' \| p'_k$, $\forall p'_k, p'_l \in P'_r: \; p'_k \not\| p'_l$ and $\forall *a'_j \in \mathcal{S}: \; (*a'_j) \bullet \cap P'_r \neq \emptyset$ □

In other words, the refining set is a set of mutually non-concurrent places belonging to $\mathcal{S}$ which are pair-wise concurrent with x'. The condition on the *inclusion of the successors* of each $*a'_j$ into P'_r is constituted by the progress requirement of the refinement procedure. Thus at least one signal will be refined at each iteration.

A refined cover $\mathcal{C}^*_{new}(x')$ is obtained from the old approximation as: $\mathcal{C}^*_{new}(x') = \mathcal{C}^*(x') \cdot \sum \mathcal{C}^{*r}_m(p'_k), p'_k \in P'_r$. The cover $\mathcal{C}^{*r}_m(p'_k)$ is *a restricted cover approximation* for p'_k, where only those literals are set to "–" whose instances, concurrent to p'_k, belong to $\mathcal{S}$ and are successors of x'. Unlike the ordinary place instance cover, $\mathcal{C}^{*r}_m(p'_k)$ uses the restricted set of transitions to exclude those who have already fired while reaching x'.

The pseudo-code of the algorithm for cover refinement is shown in Figure 9.

proc *Refine covers*(x', x'')
  if $\mathcal{C}^*(x') \cdot \mathcal{C}^*(x'') \neq \emptyset$ **then do**
    Find set of offending signals Sig
    if $|Sig| = 0$ **then** Report CSC problem and TERMINATE
    Choose an offending signal a_j *from* Sig
    Find refining set P'_r *for* x' *w.r.t.* a_j
    $\mathcal{C}^*_{new}(x') = \mathcal{C}^*_m(x') \cdot \sum \mathcal{C}^{*r}_m(p'_k) : \; p'_k \in P'_r$
    Find refining set P'_r *for* x'' *w.r.t.* a_j
    $\mathcal{C}^*_{new}(x'') = \mathcal{C}^*(x'') \cdot \sum \mathcal{C}^{*r}_m(p'_k) : \; p'_k \in P'_r$
  end do
end do

Figure 9: Procedure for refining cover approximations

Proposition 5.5 The synthesis procedure given in Figure 7(a) is finite for any implementation architecture.

Proof: [*Sketch*] Since each step refines the value of at least one variable and the set of signals is finite, the refinement procedure will terminate after at most $|A| \times |P'_a|$ iterations. If at the end of refinement any two covers intersect, then the STG has CSC problem which will be reported. □

Proposition 5.6 The fully refined (in the worst case) cover of x' in the STG-unfolding segment built for a CSC-compliant STG covers only states corresponding to the cuts which are encapsulated by $\mathcal{S}$ and covered by the exact cover for x'.

Proof: [*Sketch*] The approximated cover for x' represents partial markings: those where input places of x', if x' is a transition, and those where x' is marked, if x' is a place. At each step the refinement procedure restores the marking component of reachable states represented by $\mathcal{S}$. It finds a set of places which can be marked together with each already partially restored marking. The cover function is changed to reflect the fact that partial markings now include the places found. Thus in the end, when the procedure terminates, the covers correspond to fully restored markings and cover only states with these marking components. □

Corollary 5.1 The fully refined cover for an excitation slice is equal to the exact cover of this slice. □

Corollary 5.2 The fully refined cover for an on- (off-) slice is equal to the exact cover of this slice. □

From the above corollaries it follows that Propositions 5.1 and 5.2 hold. Therefore, our chosen strategy produces correct covers for any CSC-compliant STG. Moreover the refinement procedure preserves covering of the $\mathcal{P}$-set. Therefore, after each iteration step a newly obtained cover also covers $\mathcal{P}$-set. Hence, if after some iteration the new $\mathcal{P}$-set and $\mathcal{N}$-set covers have empty intersection, then they will satisfy the implementation specific correctness criterion, set out in Definitions 2.1-2.3 for each implementation architecture.

Example. Consider a fragment of STG-unfolding segment shown in Figure 8(c). Suppose that on-set cover approximation $\mathcal{C}^*_{On}$, found with approximation set $P'_a = \{p'_1, p'_3, p'_5, p'_8\}$, intersects with $\mathcal{C}^*_{Off}$ for some signal. Suppose also that a cube $B = d\bar{e}$ which is an cover approximation of place p'_5 causes this non-empty intersection. The set of offending signals is found as $Sig = \{a, b, c\}$. Let a be the signal chosen for refinement. Its only instance which should be used in refinement is $-a'$. A refinement set is chosen as $P'_r = \{p'_2, p'_4, p'_7, p'_9\}$. Consider calculation of the restricted cover approximation for p'_2. The only instances which can be used in approximation is $+e'$ as other concurrent instances, $+a'$ and $+d'$, precede p'_5. Thus $\mathcal{C}^{*r}_m(p'_2) = \{1001-\}$ (the order of signals is $abcde$). Similar, cover approximations are found for other places in P'_r. The refined cover approximation is thus found as: $\mathcal{C}^*_{new}(p'_5) = \{---10\} \cap [\{1001-\} \cup \{1101-\} \cup \{1111-\} \cup \{0111-\}] = a\bar{c}d\bar{e} + bcd\bar{e}$. The resulting cover is an exact cover of for place p'_5. Note that if simply cover approximation $\mathcal{C}^*_m(p'_2) = \bar{b}\bar{c}$ were chosen for p'_2 (or any other place from P'_r), then refinement would not refine a. ■

6 Experimental results

We implemented the method introduced in this paper on the basis of already existing STG verification tool "PUNT" which constructs an STG-unfolding segment. The Negative set approximation strategy requires On- and Off-set partitioning for each of our implementation architectures. Therefore, we chose only one of them, ACGpS, as an indicator of the method's performance. While testing the new method we pursued two objectives:

- To demonstrate the practicality of our approach on a set of moderate sized examples.
- To illustrate the increased feasibility of synthesis process using our approach.

Let us discuss the results for each objective separately.

6.1 Practicality

To demonstrate the practicality of our approach we chose a set of publicly available benchmarks. All STGs in this set of benchmarks have low number of signals (max. 25). Thus the size of the state space is moderate and it is possible to synthesise these STGs with other existing tools. The synthesis results are presented in Table 1. The table presents total time spent (in

Benchmark	Sigs	PUNT ACGpS		Other tools		
		TotTim	LitCnt	Petrify	SIS	LitCnt
imec-master-read.csc	18	77.00	83	125.66	630.52	69
nowick.asn	7	0.97	17	1.44	0.51	20/17
nowick	6	0.57	15	1.10	0.23	14
par_4.csc	14	3.63	36	12.31	168.55	36
sis-master-read.csc	14	5.78	48	27.09	130.66	48
tsbmSIBRK	25	42.70	72	299.90	141.51	72
pn_stg_example	6	1.77	19	4.20	6.84	19
forever_ordered	8	1.46	20	5.24	8.81	16
alloc-outbound	9	0.85	16	1.75	1.53	16
mp-forward-pkt	20	0.83	17	1.50	0.22	17
nak-pa	10	0.96	20	2.28	0.29	20
pe-send-ifc	17	2.53	68	19.50	1.16	75/72
ram-read-sbuf	11	1.08	25	3.28	0.26	22
rcv-setup	5	0.25	8	0.72	0.14	8
sbuf-ram-write	12	1.48	23	4.04	0.38	23
sbuf-read-ctl.old	8	0.86	15	1.29	0.19	15
sbuf-read-ctl	8	0.71	15	0.99	0.16	15
sbuf-send-ctl	8	0.88	19	1.95	0.21	19
sbuf-send-pkt2	9	0.99	19	2.16	0.23	19
sbuf-send-pkt2.yun	9	1.07	31	3.43	0.26	31
sendr-done	4	0.23	6	0.33	0.14	6
Total	228	146.78	592	520.16	1092.77	580/574

Table 1: Experimental results

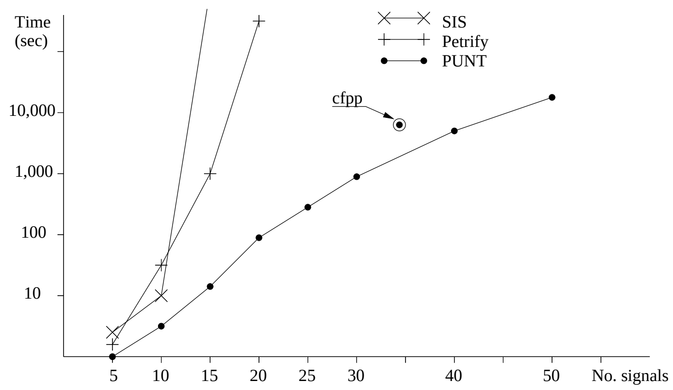

Figure 10: Experimental results for Muller pipeline.

seconds) on synthesis of speed-independent circuits from their STG specifications in the ACGpS architecture ("PUNT ACGpS"). On average, about 1% of this time was spent on building the STG-unfolding segment and about 15% was spent on Espresso minimisation. For comparison, the same set of benchmarks was synthesised using Petrify and SIS [5, 19]. Their total timings are grouped in the column "Other tools".

To illustrate the practicality of our method we used literal count (columns "LitCnt"). The literal count shows total number of literals used in all cubes of logic functions implementing all signals. As it can be observed, our synthesis technique based on the STG-unfolding segment produces implementations comparable with those produced by other tools. The timing results show that our technique compares favourably to Petrify [5]. It is also comparable with SIS on the benchmarks with low count of signals and it becomes increasingly better with the growth of the signal count. These results show that for small sized benchmarks, the overheads of constructing and traversing the STG-unfolding segment may outweigh the time spent on constructing a small reachability graph with an efficient implementation. Slightly worse literal count for some benchmarks is attributed to the fact that the DC-set is partitioned due to a stricter cover correctness condition.

6.2 Feasibility

To illustrate feasibility we chose the Muller pipeline benchmark. The graph interpretation of the results is shown in Figure 10. As can be observed, existing cannot cope well with the growing size of the specification, either running out of memory or taking prohibitively long time. Doubly exponential dependency of SIS and Petrify is attributed to (1) exponential explosion of the state space, and (2) exact cover calculation methods. In addition, we synthesised a Counterflow pipeline specification [21] which has 34 signals. From the existing tools, only Petrify was able to synthesise it but it took more than 24 hours. At the same time PUNT was able to synthesise it in under 2 hours, thus giving an order of magnitude gain in speed. This result is shown on the graph as a circled dot.

7 Conclusions

In this paper we presented a new method for synthesis of speed independent circuits. Our approach is based on the STG-unfolding segment. It uses the segment as a model from which an implementation is obtained. As the size of the STG-unfolding segment is often smaller than that of the SG, it is possible to synthesise larger circuits. In addition, due to the smaller size of the semantic model, the implementation can be achieved faster on a number of moderate sized examples. We demonstrated applicability of our method on an existing set of benchmarks. Our method can be used with the widest class of STG specifications.

Future development of this method can be directed into exploring heuristics for the refinement procedure. Although extended covers, covering states in both ERs (GERs) and QRs (GQRs), were not considered in this work, this method can be used to obtain implementations with the covers optimised on QRs (GQRs). The authors are also planning to investigate the second cover derivation strategy based on the cover evaluation and refinement.

References

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

[2] P.A. Beerel and T.H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proceedings IEEE 1992 ICCAD Digest of Technical Papers*, pages 581–586, 1992.

[3] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[4] T.A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, 1987.

[5] J Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *Proc. of the 11th Conf. Design of Integrated Circuits and Systems*, Barcelona, Spain, November 1996. to appear.

[6] J. Esparza, S. Römer, and W. Vogler. An inprovement of mcmillan's unfolding algorithm. Technical Report TUM-I9599, Insitute Für Informatik, Technische Universität München, 1995.

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

[8] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 56–62, June 1994.

[9] A. Kondratyev, M. Kishinevsky, and A.V. Yakovlev. On hazard-free implementation of speed-independent circuits. In *Proceedings of Asia and South Pasific Design Automation Conference (ASP-DAC'95)*, pages 241–248, 1995.

[10] A. Kondratyev and A. Taubin. On verification of the speed-independent circuits by STG unfoldings. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems, Salt Lake City, Utah, USA*, November 1994.

[11] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Boston, 1993.

[12] T.H.-Y. Meng, R.W. Brodersen, and D.G. Messershmitt. Automatic synthesis of asynchronous circuits from high-level specifications. In *IEEE Transactions on Computer-Aided Design*, volume 8(11), pages 1185–1205, 1989.

[13] T. Miyamoto and S. Kumagai. An efficient algorithm for deriving logic functions of asynchronous circuits. In *Proc. of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'96)*, pages 30–35, Aizu-Wakamatsu, Fukushima, Japan, March 1996.

[14] E. Pastor, J. Cortadella, A. Kondratyev, and O. Roig. Structural methods for the synthesis of speed-independent circuits. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 340–347, Paris(France), March 1996.

[15] W. Reisig. *Petri Nets, An Introduction.* Springer-Verlag, 1985.

[16] L.Ya. Rosenblum and A.V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy, July 1985*, pages 199–207.

[17] A. Semenov and A. Yakovlev. Event-based framework for verification of high-level models of asynchronous circuits. Technical Report 487, University of Newcastle upon Tyne, 1994.

[18] A. Semenov, A. Yakovlev, E. Pastor, M.A. Peña, and J. Cortadella. Synthesis of speed independent circuits from stg-unfolding segment. Technical report, University of Newcastle upon Tyne, 1996.

[19] E.M Sentovich and et. al. SIS: A system for sequential circuit synthesis. Memorandum No. UCB/ERL M92/41, University of California, Berkeley, 1992.

[20] P. Siegel and G. De Micheli. Decomposition methods for library binding of speed-independent asynchronous designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 558–565, 1994.

[21] A. Yakovlev. Designing control logic for counterflow pipeline processor using petri nets. Technical Report 522, University of Newcastle upon Tyne, 1995.

[22] Ch. Ykman-Couvreur, B. Lin, and H. DeMan. ASSASSIN: A synthesis system for asynchronous control circuits. Reference manual, IMEC, 1995.

[23] Ch. Ykman-Couvreur, B. Lin, G. Goossens, and H. De Man. Synthesis and optimization of asynchronous controllers based on extended lock graph theory. In *Proc. European Design and Test Conference (EDAC-ETC-EuroASIC)*, pages 512–517, February 1993.