Identifying State Coding Conflicts in Asynchronous System Specifications Using Petri Net Unfoldings [*]

Alex Kondratyev
The University of Aizu, Japan

Jordi Cortadella
Universitat Politècnica de Catalunya, Spain

Michael Kishinevsky
The University of Aizu, Japan

Luciano Lavagno
Politecnico di Torino, Italy
Cadence Berkeley Labs, USA

Alexander Taubin
The University of Aizu, Japan

Alex Yakovlev
University of Newcastle upon Tyne, UK

Abstract

State coding conflict detection is a fundamental part of synthesis of asynchronous concurrent systems from their specifications as Signal Transition Graphs (STGs), which are a special kind of labelled Petri nets. The paper develops a method for identifying state coding conflicts in STGs that is intended to work within a new synthesis framework based on Petri net unfolding. The latter offers potential advantages due to a partial order representation of highly concurrent behaviour as opposed to the more traditional construction of a state graph, known to suffer from combinatorial explosion. We develop a necessary condition for coding conflicts to exist, by using an approximate state covering approach. Being computationally easy, yet conservative, such a solution may produce fake conflicts. A technique for refining the latter, with extra computational cost, is provided.

1 Introduction

There exists a variety of approaches to synthesis of speed independent circuits from their formal behavioural specifications. One of the most popular specification languages is Signal Transition Graphs (STGs) that are Petri nets (PNs) whose transitions are labelled with the names of rising and falling edges of circuit signals [1, 17]. Circuit synthesis methods based on STGs can be classified into two major groups. The first group includes those based on a State Graph (SG), which is the Reachability Graph (RG) of an STG (strictly speaking of the PN underlying the STG) encoded with binary vectors corresponding to the states of signals in every reachable marking. This approach is used in existing software tools for asynchronous circuit synthesis such as SIS [19] and Petrify [2]. The actual process of circuit implementation involves direct construction of the full reachable state space, which then provides logic minimisation routines with the information about On, Off and Don't care sets for each non-input signal. An obvious practical limitation of this approach is a potential combinatorial growth in the number of reachable states. The use of symbolic techniques, such as Binary Decision Diagrams (BDDs), sometimes yields a more efficient representation of the binary encoded states [2] but does not remove the root of the complexity issue.

The second approach avoids construction of the full reachable state space; it includes techniques either based on structural analysis of STGs [15] or use of PN unfoldings [10, 18]. The structural method of [15] has given rise to the idea of an approximation-based synthesis of the logic implementation of an STG. Albeit efficient in many practical cases, it is restricted to only handling a sub-class of PNs – free-choice nets [3]. The attempt to generalise it within the framework of unfolding presented in [18] has proved to be quite promising in dealing with large STG models.

In particular, unfoldings exploit the nature of practical asynchronous specifications, that suffer much more from state explosion due to concurrency than due to conflict. STGs generally also exhibit a "regular" interaction between the two, thus avoiding the pathological cases in which the unfolding performs as poorly as traditional state exploration (or even worse that state exploration, due to the larger constant factors in the complexity of the algorithmic implementations).

The main shortcoming of the method of [18], however, was that its approximation and refinement strategy was fairly straightforward and could not work well with the Don't care state sets, i.e. sets of states which would have been unreachable if the exact reachability analysis was applied. In particular, if two approximation cubes were intersecting on the unreachable states, the only way to confront this problem was to construct the corresponding states to see whether this intersection was dangerous or not. The construction (or refinement) procedure suggested in [18] was inefficient and caused an explosion in the number of cubes obtained during the refinement.

With the increasing popularity of STGs and associated synthesis tools, there is a clear need for further development of the partial order approach to asynchronous circuit synthesis. We do not attempt to tackle at once all the issues involved, since this subject requires developing a considerable amount of new theory. This paper therefore aims at improving the synthesis method based on unfoldings in its particularly critical part: to find a more accurate way of determining actual coding conflicts in the STG unfolding. A state coding conflict occurs when a pair of different

[*]This work has been partially supported by ACiD-WG (ESPRIT 21949), CICYT TIC 95-0419, EPSRC GR/L24038/K70175 (projects ASTI and HADES)

states in a specification has the same binary encoding (this is called Complete State Coding (CSC) conflict [1]). Such conflicts are tentatively identified by means of a conservative estimation of the state space, via place cover cubes. Some of these conflicts may not be actual CSC conflicts, thus leading to the two *main contributions* of the paper:

1. Conditions to determine whether a particular state coding conflict is fake (Section 3). From the computational point of view, these conditions are relatively easy to check, but they are necessary and not sufficient, which may require further refinement if the designer is prepared to use a more complex procedure.

2. An algorithmic method for the partial construction of the state space when the "fast" techniques from Section 3 fail (Section 4). This method is based on solving the problem of calculating the part of the STG unfolding whose states (unfolding cuts) evaluate a given boolean cover cube to true. This problem has its own specific value in the list of issues that need to be tackled for a more thorough understanding of the "boolean properties" of partial order behavioural specifications.

The position of this paper amongst asynchronous design techniques is illustrated by the "roadmaps" shown in Figure 1.

2 Background

This section introduces the basic concepts required for describing the new method. These include: (i) models, such as Signal Transition Graph, State Graph, Unfolding; (ii) target properties, such as Complete State Coding, CSC conflicts; (iii) important notions supporting the method, such as unfolding cuts, slices, marked regions, approximation cubes.

2.1 Signal Transition Graph and State Coding Problems

A *Petri net* (PN) is a quadruple $PN = \langle P, T, F, m_o \rangle$, 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) [17, 1] is a triple $N = \langle PN, A, \lambda \rangle$, where PN is a PN, $A = I \cup O$ is a set of signals partitioned into input and output signals, and $\lambda : T \rightarrow A \times \{+, -\}$ is a labelling function that assigns a signal edge name to each transition in T. An STG is thus a labelled PN, specialised to describing the behaviour of asynchronous circuits at the logic level. The set of transitions represents signal changes, i.e. their rising (a_i+) and falling (a_i-) edges. Notation a_i* is used to indicate a signal transition regardless of the direction of the change. Given a Petri net element $x \in T \cup P$, its predecessors and successors sets are denoted $\bullet x$ and $x \bullet$ respectively. We further assume that for any transition $t \in T : \bullet t \neq \emptyset$ and $t \bullet \neq \emptyset$. A PN in which every transition has at most one predecessor and one successor is called a *State Machine*.

An STG is called *k-bounded* iff the number of tokens in any place $p \in P$ 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 signal persistent* [8] iff no output signal transition a_i* excited at any reachable marking can be disabled by transition of another signal a_j*. If an STG is output signal persistent, then it 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 require building a *State Graph* (SG). The SG is derived from the graph of reachable markings (RG), constructed for the STG using either explicit [16] or symbolic traversal [14] methods, and then assigning a binary code $v \in \{0, 1\}^n, n = |A|$, to each reachable marking m [1]. Thus an SG is a triple $SG = \langle S, E, \gamma \rangle$, where S is a set of binary encoded states $s = (m, v)$, E is a set of arcs between the states, and $\gamma : E \rightarrow A \times \{+, -\}$ is a function that labels the arcs with signal transitions. In order to allow a meaningful interpretation of the SG model as the behaviour of an asynchronous circuit, the binary codes v must be assigned to their markings m *consistently*, i.e.

- every arc between two states $s_1 = (m_1, v_1)$ and $s_2 = (m_2, v_2)$ 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.

Whilst at the STG level the states are pairs, consisting of a marking and a state code, at the circuit level, only their binary codes will be represented. Thus it may be possible that two states of an STG that have different markings and are semantically different (they generate different behaviour in terms of firing transition sequences) but having equal binary codes will be indistinguishable at the circuit level. This situation will be called a *coding* or *CSC conflict*. The *Complete State Coding* (CSC) condition introduced in [1] 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.

An STG that is bounded, consistent, output signal persistent, and producing a SG with CSC is known to be implementable [8] as a *speed-independent* logic circuit [2]. An implementable STG gives rise to truth tables, which can be derived from the SG state codes for each output signal. The implementation is obtained from the truth tables by building cover functions, which are then directly associated with the circuit elements. This is the so-called *complex gate implementation*. In this paper we assume such an implementation to be the target of synthesis, and thus consider only coding conflicts related to this basic form.

A boolean function *covers* a state $s = (m, v)$ if the function evaluates to true when the variables have their values equal to the signals at the binary code v. A function

[1] In general one marking of an STG can correspond to a few binary codes. It can happen for example if two-phase signal transitions are allowed or due to a few different initial paths leading to same place of an STG. However, any STG can be converted to an equivalent STG with single binary code for each marking. Therefore, in this paper we consider only such STGs.

[2] Circuits whose behaviour is independent of the delays in logic gates; such circuits are known to be free from hazards under the unbounded delay model.

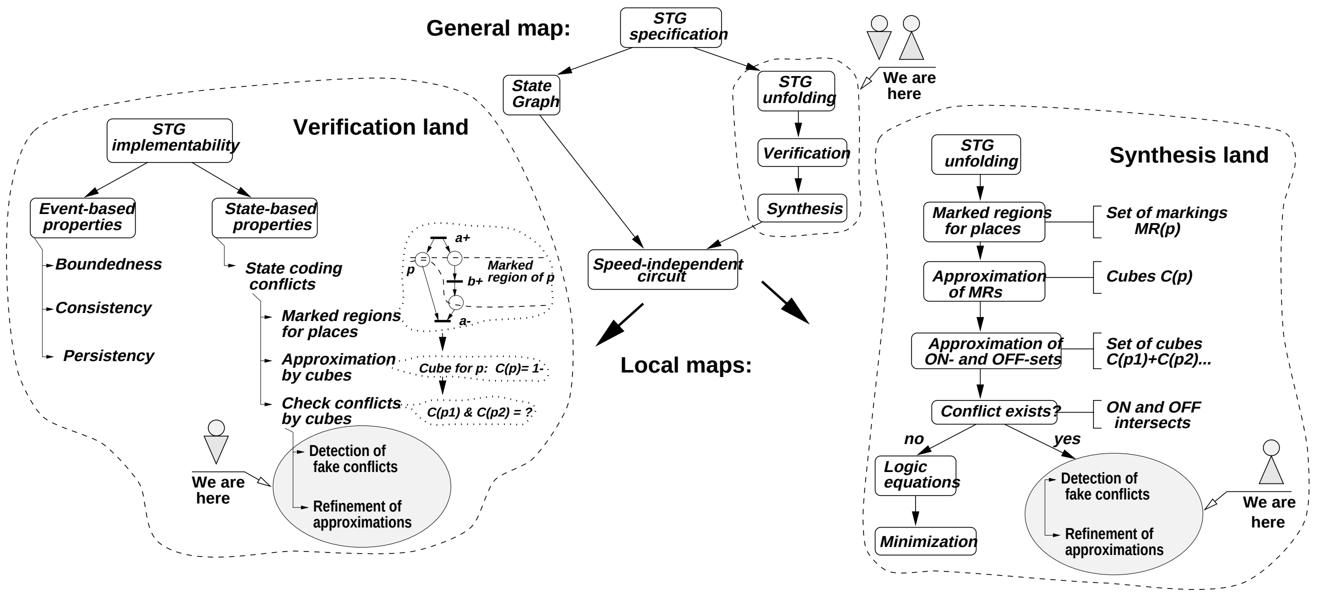

Figure 1: Where are we?

covering a set of states is called a *cover function* (or simply *cover*). Each product term of the cover is associated with a *cube* which may cover several states (commonly associated with min-terms) in the state space. In the sequel we will use $\cup$ (union) and $\cap$ (intersection) for covers (or cubes), assuming these set-theoretic operations to be applied to the sets of states covered by these covers (cubes).

Example 2.1 *(The "xyz" example.) Consider the* STG *and its* SG *shown in Figure 2,a,b. This* STG *is bounded, consistent and output-persistent (assuming that all signals x, y and z are outputs); it satisfies the* CSC *property since each reachable state has a unique state encoding (shown next to the marking). An example of a cover function is: $(x+z)\overline{y}$ (we will often use an alternative Boolean vector notation $10-\cup-01$, assuming signal ordering xyz), which covers the set of states: $\{(p2p3;100),(p4p3,101),(p6p3;001)\}$. Note that $10-$ and -01 are the cubes associated with products terms $x\overline{y}$ and $\overline{y}z$, respectively.*

2.2 STG unfoldings and their role in synthesis

Checking whether a particular STG is implementable in complex gates is a crucial step in speed-independent circuit synthesis. To be able to synthesise circuits from large STGs we would like to avoid using explicit state enumeration techniques. A compact representation of STG state space is provided by Petri net unfolding [12]. It is known that its finite fragment, a *truncated unfolding* [12], completely represents the entire reachability graph of the PN. Techniques for analysis of boundedness, consistency and output-persistency of STGs using unfoldings have been developed elsewhere, e.g. [10]. Those conditions could be easily interpreted in terms of ordering relations (concurrency, conflict and precedence) between the unfolding elements. The situation with the CSC condition, which is related to the problem of binary state encoding, is different. To be able to check this condition, one needs a way to capture state encoding information from the STG unfolding.

One such possible way was suggested in [18], within a general framework for synthesis of speed-independent circuits from unfoldings. It was based on the idea of finding *approximated* boolean covers for instances of places and transitions [15].

An *exact* cover for a given set of states S' can be obtained directly from the set of binary codes S', 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 be contained in 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 [15, 18]. However, implementations created by using approximated covers require additional checking for their correctness. One such condition for complex gate implementation is that the cover for the part of the state space where the function is on (ON-set cover) must not intersect with the cover where the function is off (OFF-set cover). If such intersection is non-empty, the synthesis process must refine the covers, until they become exact in the worst case. As a matter of fact, it was pointed out in [18] that the situation when the exact covers for ON-set and OFF-set have a nonempty intersection precisely corresponds to the case of a CSC problem.

The technique for generating and refining approximated covers proposed in [18] was quite straightforward. It did not take into account that the intersection of the ON-set and OFF-set covers for a signal could be on the set of unreachable states, corresponding to the DC(Don't Care)-set. Therefore, the fact that the ON-set and OFF-set covers have nonempty intersection cannot say precisely whether the STG has a CSC conflict or not. In the latter case we shall say that the CSC conflict is fake.

In order to tackle the problem of checking the CSC condition in the STG unfolding, we apply some of the concepts used in the unfolding theory. First, the concept of an STG-unfolding is outlined. Then, we introduce the notions of cuts [4] and slices [18], which allow us to capture the corresponding notions in the SG, namely states and connected subsets (regions) of states. Cuts and slices will thus provide us with an important link with the state coding information. The latter is represented in the form of boolean cubes (and covers) associated with the unfolding elements.

2.2.1 STG unfolding

An STG *unfolding*[3] built for an STG N, is an occurrence net $N' = \langle P', T', F', \Lambda \rangle$ where P', T' and F' are sets of places, transitions and the flow relation, respectively; and Λ is a labelling function which labels each element of N' as an instance of elements of N. N' is a partial order obtained from an STG N by the process of its unfolding [12, 5, 10]. We tacitly assume that unfolding N' inherits the signal transition labelling (function λ) of its STG origin N.

Note. To distinguish the elements of the PN (or STG) unfolding from those of the original PN (STG) we will always refer to the former by adding one or several primes (p', t'',...) while the objects of the latter are denoted simply by p, t, etc.

In the STG 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 of the original STG. For any pair of instances $x'_1, x'_2 \in P' \cup T'$ the following three relations are defined:

- *Precedence* , denoted as $x'_1 \Rightarrow x'_2$, iff (x'_1, x'_2) belongs to the reflexive transitive closure of F', i.e., there is a path in the graph of an unfolding between x'_1 and x'_2.
- *Conflict*, denoted as $x'_1 \# x'_2$, iff there exist two distinct transitions t'_1 and t'_2 such that $\bullet t'_1 \cap \bullet t'_2 \neq \emptyset$, and $t'_1 \Rightarrow x'_1$, and $t'_2 \Rightarrow x'_2$.
- *Concurrency*, denoted as $x'_1 || x'_2$, iff x'_1 and x'_2 are neither in precedence, nor in conflict.

In contrast to PN unfolding [12, 5], the STG unfolding preserves the signal interpretation of the PN transitions and keeps track of the binary codes reached by transition firing. However, it explicitly represents only a subset of all reachable states of N (called *basic states* in [11]) and thus is typically more compact than the SG. The set of predecessor transitions of t' of the STG unfolding is called the *local configuration* of t' and is denoted as $\Rightarrow t'$.

The set of place instances reached by firing all transitions in $\Rightarrow t'$ is called the *postset of* $\Rightarrow t'$ and is denoted by $(\Rightarrow t')\bullet$. Mapping a postset onto places of the original STG produces a marking of the original STG, called a *basic marking* (unlike the reachability graph, the unfolding represents only basic markings) and denoted as $m(\Rightarrow t')$. Any non-conflicting and transitively closed (w.r.t. the precedence relation) subset of transitions $T1' \subseteq T'$ is called a *configuration*. It is clear that a configuration is a union of local configurations of the transitions that are maximal (w.r.t. the precedence relation) in the configuration.

Each instance t' of the STG unfolding has a binary code $v(\Rightarrow t')$ which is reached by firing transitions in $\Rightarrow t'$. The postset $(\Rightarrow T1')\bullet$ and binary code $v(\Rightarrow T1')$ corresponding to a configuration $T1'$ are calculated from $(\Rightarrow t')\bullet$ and $v(\Rightarrow t')$ of the max-transitions t' of this configuration. The pair $(m(\Rightarrow t'), v(\Rightarrow t'))$ is called the *final state* of the local configuration $\Rightarrow t'$. Similarly, we can denote the final state of a configuration $(m(\Rightarrow T1'), v(\Rightarrow T1'))$, which always corresponds to one of the reachable markings. It has been known that all reachable markings of an STG are represented in the STG unfolding as post-sets of some configuration [12], and this is easily generalized for all states of the SG [10].

The process of constructing the STG unfolding (which is a finite object for a bounded PN) is terminated at the transition instances called *cut-off points*, whose final state is equal to the final state of some other instance already put into the unfolding. There exist several definitions of the cut-off condition [12, 5, 10], different in their attempts to minimize the size of the truncated PN (or STG) unfolding necessary to fully represent the SG.

The initial state of the STG is associated with an imaginary *initial transition* in the unfolding, whose postset is the set of place instances of the places involved in the initial marking.

2.2.2 Cuts and slices of STG unfolding

To represent a state of the SG we define a cut in the unfolding [4].

Definition 2.1 A cut of an STG unfolding *is a maximal set of mutually concurrent places* $p' \in P'$.

Each cut $m' \subset P'$ thus represents a reachable marking $m = \Lambda(m')$ of the original STG. Due to the acyclic nature of the PN unfolding (recall that we are talking about the fragment of the unfolding truncated at its cutoff transitions) it may cover some markings more than once, i.e., several cuts may map to the same marking. Due to the main property of the STG unfolding to be representative of all reachable states, for every reachable state in an STG there is a cut in the STG unfolding. Thus, similar to markings, each cut $m' \subset P'$ is also associated with a binary code $v(m')$ of the marking $m = \Lambda(m')$.

The order relations can be defined between cuts in the following way:

- *Precedence*, $m1' \Rightarrow m2'$ iff $\forall p1' \in m1' \ \exists p2' \in m2'$, $p1' \Rightarrow p2'$. Note that relation $\Rightarrow$ for cuts is reflexive due to reflexivity of $\Rightarrow$ for places of an unfolding.
- *Conflict*, $m1' \# m2'$ iff $\exists p1', p2'$, $p1' \in m1'$, $p2' \in m2'$ and $p1' \# p2'$.

[3]We apply term unfolding to the notion of the "truncated unfolding" for simplicity, under the assumption that the STG is bounded and such a truncation is possible [12].

- *Coexistence*, $m1' || m2'$ iff neither $m1' \Rightarrow m2'$ nor $m1' \# m2'$

Since a cut m' represents a reachable state $s = (\Lambda(m'), v(m'))$, there exists a configuration $T1'$ such that $s = (m(\Rightarrow T1'), v(\Rightarrow T1'))$. We shall call such $T1'$ the *configuration of cut* m', and denote it by $\Rightarrow m'$. In particular, the empty configuration corresponds to the initial cut of the unfolding. Conversely, for configuration $T1' = (\Rightarrow m')$ the cut m' will be called the *final cut of configuration* $T1'$. The precedence and coexistence relations involve cuts whose configurations do not contain conflict transitions. The conflict relation is between cuts whose configurations include at least a pair of transitions, one from each configuration, which are in conflict.

We need also to rephrase the notion of CSC in terms of cuts.

Definition 2.2 *Two* cuts $m1'$ *and* $m2'$ *are said to be in* CSC conflict *iff* $v(m1') = v(m2')$ *and they enable transitions labeled with different output signals.*

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

Definition 2.3 *A slice* $\mathcal{S} = \langle \bullet\mathcal{S}, \{\mathcal{S}\bullet\}\rangle$ *is a set of unfolding cuts defined by a cut,* $\bullet\mathcal{S}$*, called min-cut and a set of cuts* $\{\mathcal{S}\bullet\}$ *called max-cuts, which satisfy the following conditions:*

- *(1)* **Min-max correspondence.** *For any max-cut* $\mathcal{S}\bullet$ *:* $\bullet\mathcal{S} \Rightarrow \mathcal{S}\bullet$ *(the min-cut is backward reachable from any max-cut).*
- *(2)* **Conflict of max-cuts.** *All max-cuts in* $\{\mathcal{S}\bullet\}$ *are in conflict*[4].
- *(3)* **Containment.** *If cut* $m' \in \mathcal{S}$*, then there is a max-cut* $\mathcal{S}\bullet$ *such that:*
  $\bullet\mathcal{S} \Rightarrow m' \Rightarrow \mathcal{S}\bullet$ *(any cut of a slice is squeezed between a min-cut and some max-cuts).*
- *(4)* **Closure.** *If cut* m' *is such that* $\bullet\mathcal{S} \Rightarrow m' \Rightarrow \mathcal{S}\bullet \in \{\mathcal{S}\bullet\}$*, then* $m' \in \mathcal{S}$ *(there are no 'gaps' in a slice).*

Conditions 1 and 2 guarantee well-formedness of the slice borders; conditions 3 and 4 guarantee containment and contiguity of a slice. Note that due to the reflexivity of the $\Rightarrow$ relation on cuts, conditions (4) and (1) imply that the min and the max cuts are part of a slice.

It is easy to see that the entire (truncated) STG unfolding is a special case of a slice. Other special kinds of slices can be defined in the STG unfolding as follows.

The *marked region* for a place instance $p' \in P'$ is the set of cuts to which p' belongs. It is easy to see that a marked region for a finite unfolding is a slice. Therefore, for the place p' we denote it as $\mathcal{S}(p')$ (an alternative name is a *place slice*). This definition can be extended to a set of mutually concurrent place instances $P1' \subset m'$, where m' is a cut; the marked region of $P1'$ is also a slice, denoted by $\mathcal{S}(P1')$.

[4]A more general definition of a slice, requiring max-cuts not to be in precedence, has been used in [18].

Since every cut in an STG unfolding has a binary encoding, each slice can be assigned a boolean cover obtained as the sum of minterms corresponding to the cuts contained in the slice. Further in Section 4 we shall define the notion of a cube slice, a slice which can be obtained for a given cube in such a way that the cube evaluates to true in all cuts of that slice and in false in all cuts outside the slice.

Our discussion of coding conflicts in an STG unfolding will require the concept of a boolean cover approximation for individual places.

Consider an arbitrary place instance $p' \in P'$. Let $t' = \bullet p'$, i.e. let t' be the *unique* (due to the non-reconvergent nature of unfoldings with respect to places) predecessor transition, and let $v(\Rightarrow t')$ denote the binary code of the final state of the local configuration of t'.

Definition 2.4 *The* cover approximation *of place* p' *is the cube* $C(p') = c[1]c[2]\ldots c[n]$*, where* $n = |A|$ *is the number of signals in the* STG*, and* $\forall i : c[i] \in \{0, 1, -\}$*, computed as follows:*

- $c[i] = $ *"$-$" if* $\exists a_i*$ *such that* $a_i * || p'$*, and*
- $c[i] = v(\Rightarrow t')[i]$*, otherwise.*

The approximate cover is a cube derived only from the local configurations of the unfolding transitions and the concurrency relation between places and transitions; all this information that can be obtained in polynomial time from the unfolding. On the other hand, the exact cover of a place p' is the boolean cover of the set of cuts in place slice $\mathcal{S}(p')$. It should be obvious that the exact cover is a subset of the approximate cover, since the approximate cover assumes that transitions concurrent to p' are all mutually concurrent, and hence that all their immediate predecessor and successor place instances can be marked in any combination. The containment is strict, except for the case wherein no pair of transitions concurrent to a place is ordered or in conflict[5].

We are now ready to consider the problem of detecting CSC conflicts using information available from an STG unfolding. The key point to avoid the complete state traversal is that the information about the state codes in the unfolding will be obtained only from place cube approximations. The next section develops a necessary condition for CSC by using this compact representation.

Example 2.2 *(The "xyz" example.) Consider the STG and its unfolding shown in Figures 2,a and c, respectively. Transition* $y-'$ *is the only cut-off transition. An example of a local configuration, for* $x-'$ *is the set* $\{x+', z+', x-'\}$*, whose final cut is* $p6'p3'$*. while an example of a non-local configuration is the set* $\{x+', z+', y+'\}$ *Its final cut is* $p4'p5'$*. An example of a slice is defined by the min-cut* $p2'p3'$ *and a max-cut set consisting of cut* $p6'p5'$*. This slice has the exact cover:* $1{-}{-} \cup 0{-}1$ *(again with signal order* xyz*). The approximate place covers are shown in the unfolding next to their place instances. Place* $p3'$ *is concurrent to transitions* $z'+$ *and* $x'-$ *and is ordered with the transitions of* y*, hence the cover approximation for this place is* $-0-$*. The exact cover of the place slice* $\mathcal{S}(p3')$ *is* $\{10-, -01\}$.

[5]This case is relatively rare in practice, except in the special case of so-called *burst-mode* specifications [13].

3 Detection of CSC conflicts by unfolding

A conservative check for CSC conflicts can be done using place cover approximations.

Definition 3.1 *Places $p1'$ and $p2'$ are said to be in* collision *in an* STG *unfolding if their cover approximations intersect, i.e. $C(p1') \cap C(p2') \neq \emptyset$.*

There are three sources of collisions between places $p1'$ and $p2'$ in an unfolding:

Case 1. The marked regions of places $p1'$ and $p2'$ contain only cuts that map to the same marking of the original STG (i.e., there is no CSC conflict).

Case 2. In the marked regions of places $p1'$ and $p2'$ there are two cuts that albeit mapped to two different markings have the same binary encoding. This may or may not be a CSC conflict, depending on whether these markings enable different or identical sets of output signals.

Case 3. The exact boolean covers of the marked regions of $p1'$ and $p2'$ do not contain the same binary codes but the place cover approximations $C(p1')$ and $C(p2')$ intersect due to an overestimation. This is called a *fake collision* and does not correspond to a CSC conflict.

The idea of approximate techniques in detecting CSC conflicts is to consider collisions (which can be easily analyzed) instead of actual CSC conflicts. However such a consideration can be overly conservative because actually we are interested only in collisions for Case 2 above, while Cases 1 and 3 must be excluded.

Definition 3.2 *A collision between places $p1'$ and $p2'$ is called* fake *if no cut in the marked region of $p1'$ is in* CSC *conflict with cuts from the marked region of $p2'$.*

To make analysis of coding conflicts by collisions between places less conservative, we need to identify as many fake conflicts as possible.

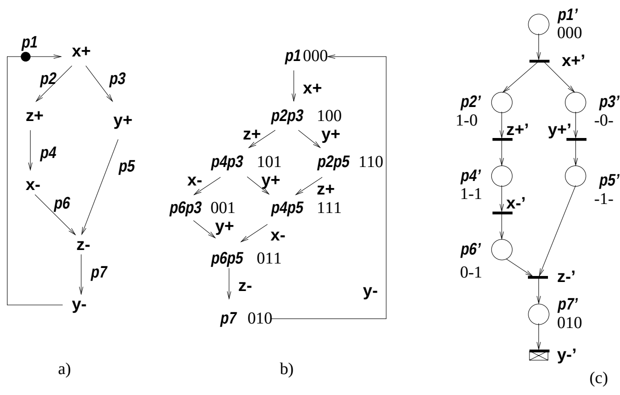

Figure 2: Approximation technique for *xyz* example

Example 3.1 *(The "xyz" example.) Consider again the* STG *and its unfolding shown in Figure 2,a,c. The cover approximation for place $p3'$ is $-0-$ (signal order is xyz). This cube intersects the corresponding cubes for places $p1, p2, p4, p6$ and thus has collisions with $p1, p2, p4, p6$. The* SG *for the xyz example is known to be free of* CSC *conflicts, therefore all these collisions are fake.*

Definition 3.3 *A directed path $e'_1, \ldots, e'_n$ over unfolding nodes (places or transitions) is called maximal if there is no node e' in the unfolding such that either $e'F'e'_1$ or $e'_nF'e'$.*

Informally a maximal path is a path that cannot be extended in the unfolding, it starts at one of its initial places and ends either at a cutoff transition or at a place without output arcs.

Definition 3.4 *A directed tree$L' = \{e'_1, \ldots, e'_n\}$ over unfolding nodes is called maximal iff:*

1. *every e'_i belongs to a maximal path formed by some of the tree nodes,*
2. *for any place $p' \in L'$ every $t' \in p\bullet'$ belongs to L',*
3. *for any transition $t' \in L'$ only one place $p' \in t\bullet'$ belongs to L'.*

Informally, maximal trees play the same role in unfoldings as State Machine components do in Free-Choice PNs [6, 3]. Specifically, they identify sets of place instances that can never be marked together (because they are ordered or in conflict), and whose marked regions contain all reachable cuts of an unfolding.

Proposition 3.1 *[9] A maximal tree contains no concurrent places.*

A maximal tree represents a maximal fragment of an unfolding without concurrency. Figure 3,c shows an example of a maximal tree in the STG unfolding of Figure 3,b. There is one more maximal tree in this unfolding given by the set of nodes: $\{p0', p8', p7'\}$ [6].

Proposition 3.2 *[9] Let P' be the set of places of a maximal tree in an unfolding N' and let M' be the union of all cuts in the marked regions of places from P'. Then M' contains all reachable cuts of unfolding N'.*

Corollary 3.1 *[9] Let an* SG *G correspond to an* STG *N with an unfolding N'. The set of cover approximations for places of a maximal tree in N' covers all states of G.*

Definition 3.5 *A place p' of an unfolding N' is called* collision stable *if every maximal tree passing through p' contains another place $p1'$ which is in collision with p'.*

Proposition 3.3 *[9] If an original* STG *N has a* CSC *conflict then its unfolding N' contains a pair $(p1', p2')$ of collision stable places.*

Proposition 3.3 states that if an STG does not satisfy CSC, then there are places (at least two) in the STG unfolding that are in collision with other places in every maximal tree. This fact will be used as a characteristic property of an CSC conflict in terms of cover approximations. Note that this property is *necessary but not sufficient*: the unfolding of an STG satisfying CSC may have stable collision

[6]When no ambiguity arises we will refer to maximal trees by their place nodes only.

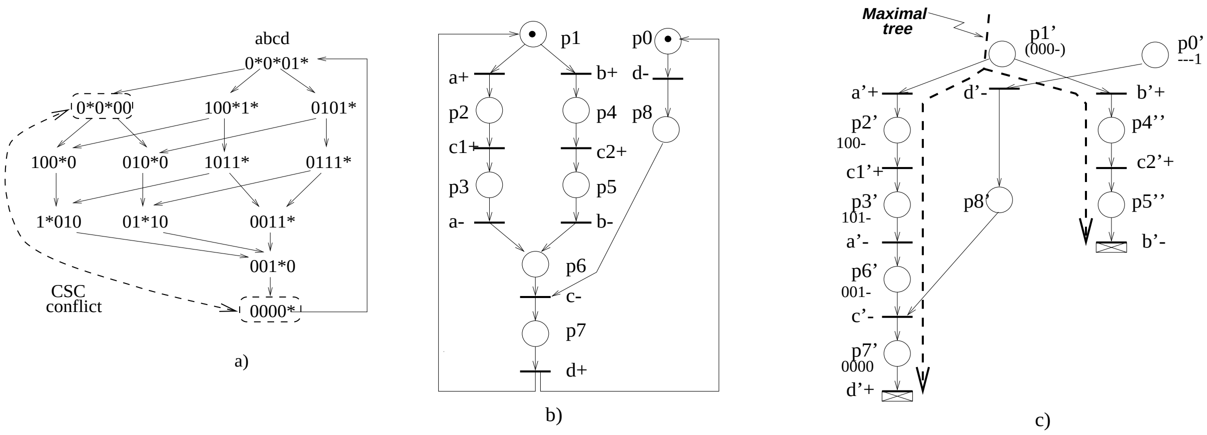

Figure 3: SG with CSC conflict a) its STG b) and unfolding c)

places. This can happen due to an overestimation of place approximation cubes and reflects the conservative nature of our approach.

Checking whether the above-mentioned situation takes place, i.e. checking for a fake collision, requires refining the collision relation between places. In Definition 3.1 this relation is defined on pairs of places $(p1', p2')$ independently from the rest of the unfolding. However, by considering the structure of collisions between $p1'$ and other places in an unfolding it is sometimes possible to conclude that the collision between $p1'$ and $p2'$ is fake.

Example 3.2 *The* SG *in Figure 3,a shows a* CSC *conflict between the pair of states 0*0*00 and 0000* (signals enabled in the state are denoted by stars, output signal d is not enabled in the first state but is enabled in the second). Let us find collision stable places in the unfolding shown in Figure 3,c (cf. Proposition 3.3).*

In the maximal tree $L1'$ (dashed line) in Figure 3,c places $p1'$ and $p7'$ are in collision. The only maximal tree that passes through $p1'$ is $L1'$ and hence $p1'$ is a stable collision place. Place $p7'$ belongs to two maximal trees: $L1'$ and $L2' = \{p0', p8', p7'\}$. In tree $L2'$, $p7'$ is in collision with $p8'$. Hence $p7'$ is a stable collision place as well. The fact that the STG *of Figure 3,b does not have* CSC *is confirmed by collision stable places in the unfolding, which illustrates Proposition 3.3.*

3.1 Refinement of collision relation between places

This subsection shows a partial (computationally easy) way to refine collisions for a given unfolding place p'. It further exploits information about maximal trees involving p'. For a particular place p' of an unfolding we can have the following cases of collisions:

(1) p' is collision free in every maximal tree;

(2) There exists a maximal tree in which p' is collision free;

(3) In any maximal tree p' has a collision, i.e. p' is collision stable.

While case (1) excludes any possibility to have CSC conflicts involving p', and case (3) is conservatively taken as a potential indication of a CSC conflict, case (2) always excludes any possibility to have CSC conflicts related to the binary states in the marking region of p'.

Proposition 3.4 *[9] If there is a maximal tree L' passing through place p' in which p' is free from collisions, then for any other maximal tree $L1'$ passing through p' any collision between p' and $p1' \in L1'$ is fake.*

Note that Proposition 3.4 does not imply that any collision between p' and other places in an unfolding are fake. It refines only the collision relations between p' and any place that can be in the same maximal tree as p'. The refinement, however, does not concern places that are concurrent with p', because these places never occur together with p' in a maximal tree. An example of such a non-fake collision between concurrent places is shown in Figure 4. In the unfolding of Figure 4,c place $p2'$ belongs to the maximal tree $\{p2', p5'\}$ and is free from collisions in this tree. The marked region of $p2'$ includes cuts $m1'$ and $m2'$ corresponding to states $0{*}0{*}$ and $00{*}$ that are in CSC conflict. Therefore a collision between $p2'$ and $p4'$ ($p4'$ is concurrent with) is not fake.

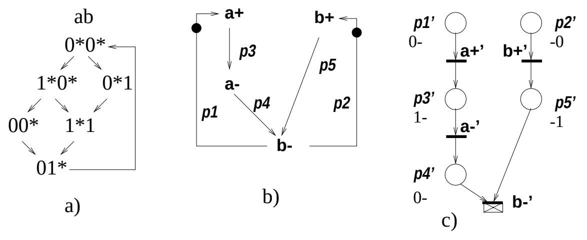

Figure 4: Non-fake collision between concurrent places

We can ignore collisions between concurrent places in an unfolding because:

1. Any CSC conflict always leads to collisions between non-concurrent places (see Proposition 3.3).
2. Insertion of new signals to distinguish CSC conflicts will be done between non-concurrent places, if we

```
Input:
    unfolding $N' = \langle P', T', F', \Lambda \rangle$,
    set $Cubes = P' \times A$ of approximation covers
    for places ($A$ -signals of STG)
    and matrix $Order = (P' \cap T') \times (P' \cap T')$ of
    ordering relations between nodes of $N'$
Output:
    matrix $Coll = P' \times P'$ of collision
    relations between places of $N'$
------------------------------------------------------------
1:foreach place $p' \in P'$ do
    construct the collision relations
    of $p'$ with all $p1' \in P'$;
    store collision relations in a matrix $Coll$
  endfor
2:do until a fixed-point in $Coll$ is reached
    foreach place $p' \in P'$ do
        if $p'$ is not a collision stable place
        then remove from $Coll$ collisions
        between $p'$ and any $p1', p1' \not\parallel p'$
    endfor
  enddo
```

Figure 5: Algorithm for the refinement of collision relations.

extend any of the known CSC resolution methods for STGs to work on unfoldings.

Thus we arrive at the procedure to refine a collision relation shown in Figure 5.

The only non-trivial step in Figure 5 is the check whether a place p' is collision stable or not. The direct analysis of this by checking the collisions with p' in every maximal tree is computationally inefficient because the number of maximal trees containing p' can be exponential. Instead we use the converse approach, and the check essentially reduces to the construction of a maximal tree in which p' is collision free. If such a tree exists, p' is clearly not collision stable (see Proposition 3.4). The procedure that finds a maximal tree (if it exists) in which p' is collision free is shown in Figure 6.

Step 1 in Figure 6 removes from the unfolding all places that are concurrent with p' (they will never occur in the same maximal tree as p') and all places with which p' is in collision (if a maximal tree in which p' is collision free exists these places cannot belong to it).

Step 2 removes from the unfolding other places and transitions that cannot be included in any maximal tree, because of the removal of places on Step 1. Indeed if all input places of some transition t' are removed, then no path from the initial places can lead to this transition. Hence no maximal tree in which p' is collision free can contain t', and t' must be removed from the unfolding together with its output places.

In turn, if all output places of some transition t' are removed, then no path from this transition can lead to the end nodes of the unfolding (cutoffs or places without output arcs). Hence no maximal tree in which p' is collision free can contain t', and t' must be removed from the unfolding together with its input places.

When in Step 2 a fixed-point in deleting the unfolding nodes is reached the rest of N' (if non-empty) contains a

```
Input:
    unfolding $N' = \langle P', T', F', \Lambda \rangle$, matrix $Order$,
    matrix $Coll$, place $p' \in P'$
Output:
    true if $p'$ is collision stable,
    false otherwise
------------------------------------------------------------
1:foreach place $p1' \in P', p1' \neq p'$ do
    if $p1' \parallel p'$ then remove $p1'$ from $P'$;
    if $p1'$ is in collision with $p'$
    then remove $p1'$ from $P'$;
  endfor
2:do until $p'$ is removed or a fixed-point
    in modifying $N'$ is reached
    /* forward traversal of $N'$ */
    if for $t' \in T'$ all places $\bullet t'$ are removed
    then remove $t'$ from $T'$
    if $t'$ is removed then remove all $p1' \in t'\bullet$
    /* backward traversal of $N'$ */
    if for $t' \in T'$ all places $t'\bullet$ are removed
    then remove $t'$ from $T'$
    if $t'$ is removed then remove all $p1' \in \bullet t'$
  enddo
3:if $p' \in P'$ then false else true
```

Figure 6: Algorithm for checking collision stable places.

maximal tree with places that are not in collision with p'. If p' has not been deleted, then it is not a collision stable place. This check is done on Step 3.

Let us evaluate the complexity of the algorithm for collision relation refinement.

The construction of the collision relations (Step 1 in Figure 5) is reduced to the analysis of pairwise intersections between approximation covers for places. This analysis is performed $O(K^2)$ times, where K is the number of places in the unfolding. The cost of each check is $O(n)$, where n is the number of STG signals. Hence the complexity of Step 1 is $O(K^2 * n)$.

The complexity of the refinement of matrix $Coll$ (Step 2 in Figure 5) is determined by checking, for each place p', whether it is collision stable or not. This check is performed by the algorithm in Figure 6, whose complexity is determined by its Step 2.

On each iteration of Step 2, at least one node of the unfolding must be removed (otherwise the fixed-point is reached). The analysis of the possibility to remove a node from an unfolding takes $O(d)$, where d is the maximum in- and out-degree of unfolding nodes. Hence the complexity of Step 2 in Figure 5 is $O((K+L)*d)$, where L is the number of transitions in the unfolding. Refinement is done for each place, and therefore it requires $O((K+L)*d*K)$ operations. Assuming that $d \ll K+L$, $n \ll K+L$ we conclude that the overall complexity of collision relations refinement is $O((K+L)^2)$, which is quadratic in the size of the unfolding. This illustrates the efficiency of the suggested method.

Example 3.3 Example *xyz* **continued.** *The application of the above algorithms to refine collision relations is illustrated by using the xyz example.*

In maximal tree $L1' = \{p1', p3', p5', p7'\}$ place $p3'$ is in collision with $p1'$. $L1'$ is the only maximal tree that contains

$p3'$ and hence $p3'$ is collision stable. To check whether $p1'$ is also collision stable let us apply the Procedure from Figure 6. At first the Procedure removes from the unfolding all places that are concurrent with $p1'$ (none in this example) and are in collision with $p1'$ (place $p3'$). By traversing the unfolding forward from the removed place $p3'$, transition $y'+$ and place $p5'$ are also removed. After this, we reach the fixed-point. The remaining part of the unfolding contains $p1'$ and hence $p1'$ cannot be collision stable (indeed, it is collision free in tree $L2' = \{p1', p2', p4', p6', p7'\}$). Hence, Proposition 3.4 implies that the collision between $p1'$ and $p3'$ is fake. From similar considerations, the collision between $p5'$ and $p7'$ is also detected as fake. After this refinement, all places in the unfolding are collision free and we can conclude that the xyz example satisfies the CSC requirement.

4 Avoiding fake collisions

Section 3.1 provided a way to detect fake collisions by refining collision relations using additional information extracted from all possible maximal trees (however, without enumerating all of them). The algorithm shown in Figure 6 actually looks for one maximal tree where a place is free from collisions. This method is more general than [15], where such a refinement was performed only by state machines that belong to the so-called State Machine-cover set, because an SM-cover set contains usually only a few SMs in comparison to the total number of SMs in which an STG can be decomposed ([6]).

However, even when refining collision relations by using *all* maximal trees, it is not always possible to avoid fake collisions. The case where the method from Section 3.1 fails can be illustrated by a modification of the *xyz* example.

Example 4.1 *(The xyz example modified.) Let us change the initial marking of* xyz *from $p1$ to $p5p6$ (see Figure 7,a).*

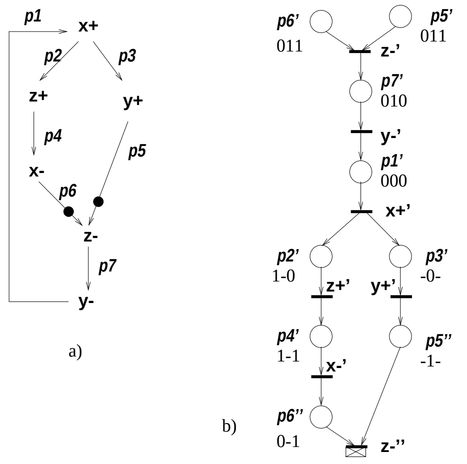

Figure 7: Unfolding of *xyz* STG with different initial marking

The unfolding for this initial marking is shown in Figure 7,b. There are four maximal trees in the unfolding: two starting from place $p6'$ ($L1' = \{p6', p7', p1', p3', p5''\}$ and $L2' = \{p6', p7', p1', p2', p4', p6''\}$) and two starting from place $p5'$ ($L3' = \{p5', p7', p1', p3', p5''\}$ and $L4' = \{p5', p7', p1', p2', p4', p6''\}$). In any of the trees there are places in collision: $p6'$ is either in collision with $p6''$ or with $p5''$, while $p5'$ is either in collision with $p6''$ or with $p5''$. Hence no refinement of the collision relation can detect them as fake ones, and it is impossible to conclude about the absence of CSC *conflicts in the* xyz *example by the unfolding in Figure 7,b.*

The failure to detect fake conflicts in the modified *xyz* example by using the unfolding shown in Figure 7,b is natural. Indeed, e.g., two collision pairs for place $p6'$ are: $\{p6', p6''\}$ and $\{p6', p5''\}$. The marked regions for $p6'$ and $p6''$ should intersect in their cover because they are instances of the same place $p6$ of the original STG, while the marked regions for $p6'$ and $p5''$ will intersect because these are instances of concurrent places $p6$ and $p5$ in the STG. Note that two instances of the same place of an STG can in general (but not in this example) be involved in a true STG conflict, if they correspond to the intersection of two reachable markings that are in CSC conflict.

There are two ways to overcome the above difficulty:

- To construct a smaller unfolding, by changing the initial marking. This method is considered elsewhere [9].
- To explore the set of states corresponding to a collision and check CSC by using this set explicitly. This method is considered below.

4.1 Checking CSC conflicts by partial construction of binary states

If the approximation cubes $C(p1')$ and $C(p2')$ of places $p1'$ and $p2'$ are intersecting ($c_{12} = C(p1') \cap C(p2') \neq \emptyset$), a straightforward way to check whether this intersection implies a real CSC conflict would be to construct all states corresponding to c_{12} in the marking regions of $p1'$ and $p2'$ and to compare the transitions enabled in these states. We will denote this process by the term "state restoration".

The advantage of this method is that it gives the exact information on CSC conflicts, while its difficulty is in the high cost (exponential in general) of the state construction. However, in practice, the marking region of a place often contains much less states than the entire unfolding; furthermore, only part of these states belong to the intersection of cubes.

To construct the states corresponding to some cube c we first need to identify in an unfolding all the regions (called *on-regions*) where cube c evaluates to 1. Similar to the marked regions of places, these on-regions are defined by sets of cuts (slices, as shown below) $\langle \theta'_c, \Theta'^c \rangle$, where θ'_c is the "first" cut, in which cube c evaluates to 1 and Θ'^c contains all the "last" cuts in which c still evaluates to 1.

Definition 4.1 *A cut θ'_c is called a minimal cut, or min-cut, for cube c if in θ'_c cube c evaluates to 1 and*

- *either θ'_c is the initial cut,*
- *or in any cut $\theta1'$ immediately preceding θ'_c (i.e. $\exists t', \theta1' \xrightarrow{t'} \theta'_c$) cube c evaluates to 0.*

Definition 4.2 *A cut θ'^c is called a maximal cut, or max-cut, for cube c if in θ'^c cube c evaluates to 1 and*

- *either θ'^c is a final cut of the unfolding,*
- *or in any cut $\theta 1'$ immediately succeeding θ'^c (i.e. $\exists t', \theta'^c \xrightarrow{t'} \theta 1'$) cube c evaluates to 0.*

A min-cut θ'_c points to the cut in which cube c is turned on for the "first" time after being set off. [7]

Definition 4.3 *A max-cut θ'^{ci} of cube c matches a min-cut θ'_{ci} if*
1. $\theta'_{ci} \Rightarrow \theta'^{ci}$
2. for any other min-cut θ'_c of c, if $\theta'_c \Rightarrow \theta'^{ci}$ then $\theta'_{ci} \not\Rightarrow \theta'_c$ and
3. for any other max-cut θ'^c of c, if $\theta'_{ci} \Rightarrow \theta'^c$ then $\theta'^c \not\Rightarrow \theta'^{ci}$.

Definition 4.3 associates any min-cut θ'_{ci} with the max-cuts "adjacent" to it. Adjacency is seen here in terms of partial order $\Rightarrow$ on a set of cuts: by Condition 2 if a max-cut θ'^{ci} matches the min-cut θ'_{ci} then no other minimal or maximal cuts can occur in between θ_{ci} and θ'^{ci} (otherwise θ'_{ci} and θ'^{ci} cannot be considered as adjacent ones).

After reaching θ'_{ci} cube c remains "On" until one of the max-cuts θ'^{ci} is reached. Beyond θ'^{ci} c immediately turns off. Hence a part of the unfolding between a min-cut θ'_{ci} and a max-cut θ'^{ci} corresponds to a part of the ON-set of cube c. Since there exist in general (due to conflicts, as shown below) several max-cuts matching the same min-cut θ'_{ci}, we will denote this matching set of max-cuts as Θ'^{ci} ($\Theta'^{ci} = \{\theta'^{ci}\}$).

The following property indicates that only conflict cuts can be included into a matching set of maximal cuts. This shows that the derivation of the ON-set of cube c does not depend on the degree of concurrency, and hence that highly concurrent STGs can be easily handled.

Property 4.1 *[9] Any two max-cuts for cube c $\theta 1'^{ci}$ and $\theta 2'^{ci}$ from the same matching set Θ'^{ci} are in conflict.*

The following property shows that the min-cut θ'_c and the set of max-cuts Θ'^c define a slice of the unfolding. It can be easily proved by applying Propery 4.1 and examining all the conditions of Definition 2.3.

Property 4.2 *The set of cuts $\{m'\}$ such that $\forall m'$: $\theta'_c \Rightarrow m' \Rightarrow \theta'^c$ for some $\theta'^c \in \Theta'^c$ is a slice $\mathcal{S}_c = \langle \theta'_c, \Theta'^c \rangle$.*

Definition 4.4 *A slice $\mathcal{S}_c = \langle \theta'_c, \Theta'^c \rangle$ is an ON-slice of a cube c if $\forall m' \in \mathcal{S}_c$: $\theta'_c \Rightarrow m' \Rightarrow \theta'^c$ for some $\theta'^c \in \Theta'^c$.*

The following property guarantees monotonicity, i.e. the absence of "value gaps", in an ON-slice.

Property 4.3 *[9] In any cut $m' \in \mathcal{S}_c$ cube c evaluates to 1.*

```
Input:  unfolding $N' = \langle P', T', F', \Lambda \rangle$
Output:  set of On-set slices ON of $c$
------------------------------------------------------------
main
    ON = $\emptyset$
    Find_ON-set(N', ON)

Find_ON-set(N', ON)
    $Min = \emptyset$
    Find_min-cuts(N',Min)
    /* Finds all minimal cuts of $c$ ''first''
    reachable from the initial marking */
    /*(i.e.  $\theta 1'_c \in Min \Rightarrow \not\exists \theta 2_c \Rightarrow \theta 1_c$) */
    foreach minimal cut $\theta'_{ci} \in Min$ do
        Find_match-cuts($\theta'_{ci}$, $\Theta'^{ci}$)
        ON = ON$\cup\langle \theta'_{ci}, \Theta'^{ci} \rangle$
        Calculate next cuts for slice $\langle \theta'_{ci}, \Theta'^{ci} \rangle$
        /* Cuts next to some cuts in $\langle \theta'_{ci}, \Theta'^{ci} \rangle$,
        in which c evaluates to 0
        foreach next cut $m'$ do
            Modify(N',m')
            /* Removes from $N'$ nodes that are
            in $\Rightarrow$ or # with places of $m'$ */
            Find_ON-set(N', ON)
        endfor
    endfor
```

Figure 8: Algorithm for the calculation of ON-set for cube c.

The procedure for calculating the slices corresponding to the ON-set of cube c is shown in Figure 8. This algorithm first finds all the min-cuts of cube c (set *Min*) that are reachable from the initial marking without passing through another min-cut (procedure *Find_min-cuts*). For all these cuts it constructs the corresponding ON-slices by calculating the matching sets of max-cuts (procedure *Find_match-cuts*). However the unfolding may have other minimal cuts of c that succeed cuts from *Min* (cube c can be set and reset several times). To find them we iterate the procedure starting from cuts m' that immediately succeed ("next cut m'") some ON-slice. The iteration means that we transfer the initial marking of the unfolding to m' (i.e. remove from the unfolding everything that precedes or in conflict with m') and proceed with the derivation of the ON-set from the modified unfolding. Eventually, the full ON-set will be constructed.

Procedures *Find_min-cuts* and *Find_match-cuts* are presented in detail in [9].

After the ON-set of cube c is found we can return to the original task of detecting CSC conflicts. This task, for a pair of collision places $p1'$ and $p2'$, consists of the following steps:

1. Let $C(p1')$ and $C(p2')$ be the cubes approximating $p1'$ and $p2'$ and $c = C(p1') \cap C(p2') \neq \emptyset$.
2. Find the intersection of the ON-set of c with the marked regions of $p1'$ and $p2'$ (denoted by $ON(p1')$ and $ON(p2')$).

[7] Cube c can be set and reset many times. To be more precise min-cuts must be enumerated and referred to by θ'_{ci} for the i-th setting of cube c. When no ambiguity arises we will omit this index.

3. Construct the binary states of $ON(p1')$ and $ON(p2')$.

4. Check for CSC conflicts explicitly, using the binary states of $ON(p1')$ and $ON(p2')$.

All the steps of this procedure are trivial to implement, with the exception of Step 2, which was discussed above. Let us consider apply the suggested method to the STG and its unfolding shown in Figure 9,a,b.

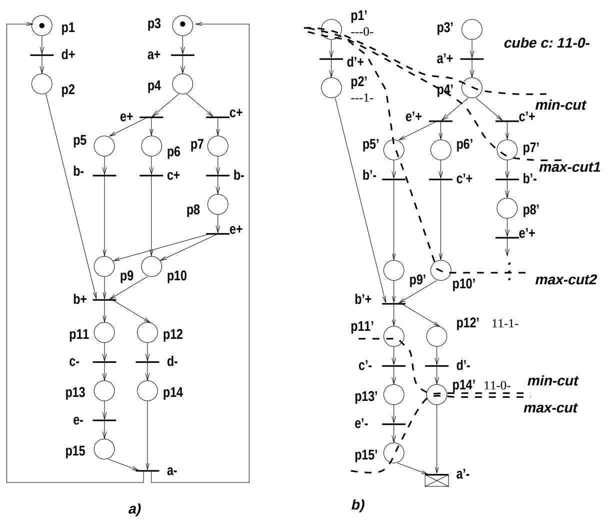

Figure 9: Derivation of On-set for a cube

Example 4.2 *Let us choose a maximal tree* $L = \{p1', p2', p12', p14'\}$ *in the unfolding of Figure 9,b. Places* $p1'$ *and* $p14'$ *in this tree are in collision. The intersection of their approximation cubes gives cube* c *=- - -0-* $\cap$ *11-0- = 11-0-.*

The marked region of $p1'$ *starts from initial marking* $p1'p3'$ *and ends in marking* $p1'p9'p10'$*. This is the first unfolding segment in which the ON-set of cube* c *is constructed. In the initial marking of this segment cube* c *evaluates to 0. Event* $a'+$ *differentiates the binary state of* m'_0 *from cube* c*. Hence we transfer the initial marking of the segment immediately after the firing of* $a'+$*: this will be the basic marking of* $a'+$*, with binary state 11000. In this binary state cube* c *evaluates to 1 and hence this is the min-cut* θ'_{c1}*. To find the matching set of max-cuts for* θ'_{c1} *let us determine the set of transitions that force* c *to reset. They are:* $a-, b-, d+$*. Only* $d+$ *and* $b-$ *have instances in the considered unfolding segment. We should remove from the segment all instances of* $d+$ *and* $b-$ *together with their successors. The remaining part has two maximal configurations: one corresponding to cut* $p1'p5'p10'$ *and another to cut* $p1'p7'$*. These cuts forms the matching set of* θ'_{c1} *and the construction the an ON-slice of* c *within the marking region of* $p1'$ *is completed:* $ON(p1') = \{p1'p4', \{p1'p5'p10', p1'p7'\}\}$*. The set of binary states corresponding to* $ON(p1')$ *is:{110*0*0*, 11*10*0, 11*0*0*1, 11*10*1}.*

In the marked region of $p14'$ *cube* c *evaluates to 1 in its initial marking* $p11'p14'$*, hence this marking is a min-cut for* c*. The marked region of* $p14'$ *does not contain any transition that resets* c*, thus the single max-cut of* c *corresponds to the single maximal configuration of the marked region,* $p15'p14'$*. The set of binary states corresponding to* $ON(p14')$ *is:{111*01, 11001*, 1*1000}.*

By checking the binary states corresponding to $ON(p1')$ *and* $ON(p14')$ *(e.g., pair of states 110*0*0* and 1*1000) it is easy to conclude that the collision between* $p1'$ *and* $p14'$ *indeed corresponds to a CSC conflict.*

5 Conclusions

We have presented a method of checking Signal Transition Graphs for state coding conflicts, in particular identifying whether an STG satisfies Complete State Coding. The latter is a key condition for an STG specification to be implementable in logic. The overall framework is based on the STG unfolding, whose potential advantage over the more traditional state graph approach is in the partial order representation of concurrent behaviour. Whilst the STG unfolding is known to help avoid the exploration of the full state space when solving some verification problems such as boundedness and consistency checks in STGs, there has been very little research in using unfoldings for circuit synthesis. In particular, the previously known method [18] for deriving logic from STG unfolding offered an important conceptual approach based on approximated boolean covers of the unfolding elements. It was however inefficient because it could not distinguish between true and fake CSC conflicts among the intersections of approximate ON and OFF covers of synthesized signals.

This paper provides an in-depth study of the coding conflict phenomenon by using the approximation-based approach. A necessary condition for CSC conflicts to exist exploits "partial" coding information (about place instances) which is made available from the computation of a maximal tree in the unfolding. Whilst this condition in many practical cases coincides with real conflicts, and is computationally efficient, it may hide the so called "fake" conflicts. This paper presents a refinement technique aimed at resolving such situations, at the expense of extra computational costs. This technique limits the search to the parts of the unfolding that *may potentially* exhibit a fake conflict. Those parts need explicit state traversing, which may be exponentially hard.

The overall efficiency of the method in practice can only be established after extensive experiments with benchmarks. It will require developing a novel set of STG benchmarks, because the existing ones (used e.g. in [8, 2]) are known to illustrate the power of state graph based techniques, rather than that of STG unfoldings, in synthesis tasks. This task, along with the software implementation of the proposed algorithms, will be addressed in the near future.

References

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

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent

specifications and synthesis of asynchronous controllers. *IEICE Trans. Inf. and Syst.*, E80-D(3):315–325, March 1997.

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

[4] J. Esparza. Model checking using net unfoldings. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT'93: Theory and Practice of Software Development. 4th Int. Joint Conference CAAP/FASE*, volume 668 of *Lecture Notes in Computer Science*, pages 613–628. Springer-Verlag, 1993.

[5] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106, Passau, Germany, March 1996. Springer-Verlag.

[6] M. Hack. Analysis of production schemata by Petri Nets. Technical Report TR 94, Project MAC, MIT, 1972.

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

[8] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig, and A. Yakovlev. Checking signal transition graph implementability by symbolic bdd traversal. In *Proc. of European Design and Test Conference*, pages 325 – 332, Paris(France), March 1995.

[9] A. Kondratyev, J. Cortadella, M.Kishinevsky, L. Lavagno, A. Taubin, and A. Yakovlev. Identifying state coding conflicts in asynchronous circuit specifications using petri net unfoldings. Technical Report TR No. 614, Computing Science, University of Newcastle upon Tyne, October 1997.

[10] A Kondratyev, Kishinevsky M., Taubin A., and Ten S. A Structural Approach for the Analysis of Petri Nets by Reduced Unfoldings. In *Applications and Theory of Petri Nets 1996. 17th International Conference. Proceedings*, volume 1091 of *Lecture Notes in Computer Science*, pages 346–365, 1996. Osaka, Japan, June.

[11] 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.

[12] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[13] S. M. Nowick and D. L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[14] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In *15th International Conference on Application and Theory of Petri Nets*, Zaragoza, Spain, June 1994.

[15] Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig. Structural methods for the synthesis of speed-independent circuits. In *Proc. of European Design and Test Conference*, pages 340 – 347, Paris(France), March 1996.

[16] J. L. Peterson. *Petri Nets*, volume 9. ACM Computing Surveys, No. 3, September 1977.

[17] L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *International Workshop on Timed Petri Nets, Torino, Italy*, 1985.

[18] A. Semenov, A. Yakovlev, E. Pastor, M. Pena, J. Cortadella, and L. Lavagno. Partial order approach to synthesis of speed-independent circuits. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, Eindhoven*, April 1997.

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