A New Approach for Checking the Unique State Coding Property of Signal Transition Graphs

Meng-Lin Yu
P. A. Subrahmanyan

AT&T Bell Laboratories
Holmdel, New Jersey 07733

Abstract

A Signal Transition Graph (STG) embodies the causal relationships among signal transitions in a system, and provides a useful starting point for the synthesis of asynchronous circuits. The prevalent synthesis techniques require the input STG to possess the unique state coding (USC) property. This paper describes an algorithm to ascertain whether a given STG has the USC property. The algorithm is path-oriented, and has the advantage of operating directly on the STG rather than a state graph. This approach has the advantage of being easier to visually and intuitively correlate with the STG specification, and therefore suggests ways in which a designer (or tool) may modify the input STG if it does not satisfy the USC property. The technique can also be used to compute the input set of a signal.

1. Introduction

A circuit is said to be asynchronous if it is sensitive to all signal transitions, and not just those that bear a fixed relation to clock signals. Asynchronous circuits arise in digital systems that interface to external signals which are not synchronized with a clock. Asynchronous designs have been argued to have some advantages over synchronous designs, such as lower average power consumption, and latencies that are governed by average rather than worst case delays; further, they alleviate difficulties related to clock skews in high performance systems. However, there continues to be a debate on the relative merits of these design styles. While synchronous design styles and CAD tools are currently more mature than their asynchronous counterparts, there has recently been renewed interest in asynchronous design techniques and tools [2], [4], [7], [9], [11].

A promising synthesis technique for asynchronous circuits proceeds from a circuit specification in the form of a Signal Transition Graph (STG) [2]. An STG is a graph in which vertices represent signal transitions and arcs represent causal dependencies between signal transitions. (Figure 1 contains an example STG.)

In order for the prevalent synthesis techniques to yield properly functioning hazard-free circuits, the input STG is required to possess certain properties. It is therefore desirable to have a way of checking whether a given STG satisfies these properties. Specifically, an STG is required to be live, safe, and have the unique-state-coding (USC) property. Intuitively, liveness implies the absence of deadlocks in a circuit or system, safety ensures that the rising and falling transitions of a signal strictly alternate, and the USC property is needed in order to guarantee that the synthesis procedure maintains enough state information to be able to disambiguate distinct states.

The focus of this paper is on an algorithm for checking whether a given STG has the USC property. In the next section we recall some of the relevant concepts and background. Section 3 then details a procedure for ascertaining whether a given STG has the USC property. An example to illustrate the procedure is contained in Section 4, while Section 5 delineates a related procedure for determining the input set of a signal.

2. Preliminaries and Background

This section summarizes some of the relevant concepts relating to Petri nets and STGs.

2.1 Petri Nets

A Petri net is a triple consisting of a finite set of transitions $T$, a finite set of places $P$ and a binary relation between transitions and places called a flow relation $F \subseteq T \times P \cup P \times T$. 
A Petri net thus defines a bipartite graph, and is commonly pictorially depicted as such. The nodes are either places or transitions, and arcs between the nodes in $P$ and $T$ represent the relation $F$. If $(p, t) \in F$, an arc is drawn from place $p$ to transition $t$. If $(t, p) \in F$, an arc is drawn from transition $t$ to place $p$. If $(v_1, v_2) \in F$ then $v_1$ is called a predecessor of $v_2$, and $v_2$ is called a successor of $v_1$. A common restriction is that the graph underlying a Petri net be strongly connected.

A free-choice net (FC net) is a Petri net, where, if a place $p$ has more than one successor, say $t_1, \ldots, t_n$, then $p$ is restricted to be the only predecessor of $t_1, \ldots, t_n$.

A marked graph is a Petri net where each place $p$ has precisely one predecessor and successor transition. The dual notion of a marked graph is a state machine where each transition $t$ has at most one input and one output place. Marked graphs represent the structure of deterministic concurrent systems, while state machines represent non-deterministic sequential systems.

2.2 Signal Transition Graphs

A signal transition graph is an interpreted free-choice Petri net. Transitions of the petri net are interpreted as signal transitions, i.e., changes of values on input, output or internal signals. (Input and output signals are sometimes called external signals.) Places are interpreted as conditions on the signal states.

A positive or rising transition, denoted $s^+$, denotes a change in the value of $s$ from 0 to 1. Correspondingly, a negative or falling transition, denoted $s^-$, denotes a change in the value of $s$ from 1 to 0. Henceforth, we use $s^+$ to denote any transition (either $s^+$ or $s^-$) of the signal $s$ and $s^*$ denotes the transition complementary to $s$.

2.3 Dynamic Behavior of a Petri Net/STG: Marking and Firing

A token marking of a petri net is an assignment of a non-negative integer to each place in the net.

A transition is enabled, implying that the corresponding event can actually occur in the system, when there is at least one token on each place leading into the transition.

Once a transition is enabled, it must eventually fire. When a transition fires, one token is removed from each predecessor of the transition, and one token is placed in each successor of the transition.

A set of transitions is said to be concurrent if all the transitions in the set simultaneously enabled in some marking reachable from the initial marking. If $t_2$ is constrained to occur after $t_1$, we write $t_1 \rightarrow t_2$; $t_1$ and $t_2$ are said to be (sequentially) ordered. Transitions that are neither concurrent nor sequential are said to be in conflict.

When a place has multiple fanout, (i.e., there are multiple successor transitions), and the net is presumed to be a free-choice net, only one of the successor transitions is nondeterministically enabled. In practice, this may be interpreted to mean that an external condition determines which transition will fire next. If this interpretation is adopted, then all such multiple output places must be constrained to be input signals.

2.4 Feasible and Complementary Sets of Transitions

A set of transitions is said to be feasible if the transitions in the set can fire without any transitions not in the set having to fire.

A set $T$ of transitions is said to be complementary if it is a proper subset of the transitions in the STG, (i.e., it does not contain all of the transitions of the STG), and $t' \in T \rightarrow t' \notin T$, i.e., if a transition is in the set, then so is its complementary transition. For example, in the STG in Figure 1, the set $\{y^+, y^-, z^+, z^-\}$ is one of several possible complementary sets. However, $\{x^+, x^-, y^+, y^-, z^+, z^-\}$ contains all of the STG...
transitions, and is not a complementary set.

2.5 Live and Safe Nets

A free-choice net is said to be live if it is always the case that each of its transitions is enabled in some marking that is reachable from its initial marking, i.e., no transition is permanently disabled in a live net.

Definition 1 (Live STG) An STG is live iff:

1. it is strongly connected.
2. every simple loop contains exactly one token
3. transitions \( t \) and \( t' \) are ordered.

A free-choice net is said to be safe if no place is ever assigned more than one token after any sequence of firings from the initial marking.

An FC net can be decomposed to yield a set of state-machine (SM) components or a set of marked graph (MG) components[9]. A set of SM or MG components is said to cover the original net iff each transition and place of the net appears in at least one of the components. Hack[9] proved that a decomposition into state-machine or marked graph components covers the original net iff the net is live and safe. This provides a syntactic check for the liveness and safeness of free-choice nets.

2.6 State Graphs

A state graph is an alternative representation of the behavior of an asynchronous circuit, where all of the concurrency in the transitions is resolved by representing it as a set of interleaved sequential transitions. The state graph corresponding to an STG can be deterministically by starting with the initial marking of the STG and exhaustively simulating the dynamic behavior of the STG[2].

Definition 2 (State Graph of an STG) The state graph \( SG \) of an STG is a directed graph with the set of nodes bearing a one-to-one correspondence with the set of STG markings reachable from the initial marking. An arc exists from state \( s_1 \) to \( s_2 \) if the marking corresponding to state \( s_2 \) is reachable from the marking corresponding to \( s_1 \) by firing a single transition; the arc \( s_1 \rightarrow s_2 \) is labeled with this transition.

2.7 Unique State Coding

The state graph \( SG \) corresponding to an STG is used in the synthesis technique described in [2], and in most related techniques. Further, the signals in the STG are used as state variables in the synthesized circuit. Clearly, since the nodes in \( SG \) represent distinct states, enough "state information" must be contained in the state encoding in the synthesized circuit to be able to disambiguate distinct markings (states in \( SG \)). This motivates the notion of an STG having the unique state coding property, and is more formally defined below.

Definition 3 (Consistent State Vector Assignment) An assignment of a boolean vector \( v_i \) (binary code) to a state \( s_i \) in the state graph \( SG \) is called a consistent state assignment if for each edge \( s_1 \rightarrow s_2 \):

1. if the edge is labeled \( t^+ \) then the value of the signal \( t \) in \( v_1 \) must be 0, and 1 in \( v_2 \);
2. if the edge is labeled \( t^- \) then the value of the signal \( t \) in \( v_1 \) must be 1, and 0 in \( v_2 \);
3. otherwise, the value of the signal \( t \) must be identical in \( v_1 \) and \( v_2 \).

Definition 4 (Unique State Coding) An STG has the unique state coding (USC) property if two distinct states in the state graph of the STG do not have identical binary codes.

An example of the state graph corresponding to the STG in Figure 1 is shown in Figure 2. The nodes are labeled with a consistent state assignment; note however, that this STG does not possess the USC property.

It was shown by Chu[2] that:

Theorem 1: An STG \( S \) has the USC property iff \( S \) is live, and no complementary set of transitions is feasible in \( S \).

2.8 Lock Graphs

As remarked earlier, if the external signals of an STG are to suffice in encoding the state of an STG, then the STG must have the USC property in order for the synthesis technique to proceed. It is thus desirable to have an algorithm that can be used to check whether a given STG has the USC property.
It is clearly possible to have an algorithm that first goes through the initial phase of the synthesis process to generate a state graph corresponding to an STG, and then checks if the state graph has the USC property. However, there are some advantages in working directly at the STG level rather than at the state graph level. First, we obtain a more intuitive “higher” level understanding of why an STG does not satisfy the USC property. Since this explanation is directly at the level of the STG, the designer can more easily suggest ways to fix the problem. Second, the complexity of the checking loop is reduced. It also bolsters the belief that USC is the appropriate property needed for the synthesis of hazard-free circuits proceeding from an STG.

One of the first high level techniques for determining whether an STG satisfies the USC property was proposed by [11], and was based on the notion of “locked signals” in the STG and an associated concept of a lock graph.

**Definition 5 (Locked Signals)** Two signals \( a \) and \( b \) are said to be locked, denoted by the lock relation \( a \overset{L}{\rightarrow} b \) iff either the sequence of transitions \( a^+ \rightarrow b^+ \rightarrow a^- \rightarrow b^- \) or \( a^+ \rightarrow b^- \rightarrow a^- \rightarrow b^+ \) lies in a simple cycle.

**Definition 6 (Lock Graph of an STG)** The lock graph (LG) of an STG is an undirected graph. The vertices are the signals of the STG, and there is an edge between \( s_1 \) and \( s_2 \) iff \( s_1 \overset{L}{\rightarrow} s_2 \) in the STG.

The following theorem may then be proved [10].

**Theorem 2 (Unique State Coding)** If the lock graph of an STG is connected, then the STG has the USC property.

While the lock graph of an STG being connected is a sufficient property for the STG to have the USC property, it is not a necessary criterion.

### 3. Algorithm

We next describe a procedure to determine whether a live, safe STG has the USC property. Our assumption in what follows is that the STG is a marked graph.

Assume a consistent state vector assignment for a state graph corresponding to an STG. We first observe the following.

**Proposition 1** For any two distinct states having identical state vectors in the state graph, each path connecting these two states corresponds to a sequence of signal transitions that comprise a complementary set, i.e., if a transition \( z' \) belongs to this set, then the transition \( z' \) also belongs to this set.

**Proof:** In the cases where a signal has multiple rising and falling transitions in the STG, we can suffix the transitions in each pair, since the rising and falling transitions are required to alternate. The extended names are unique for every transition and the definition of a complementary set of transitions can then apply.

If the path involved is not a complementary set, then there is at least one signal transition \( z' \) that is unpaired. Since the input STG is assumed to be live and safe, the state graph is guaranteed to have a consistent state assignment. This implies that the values of the bit corresponding to signal \( X \) in the state vectors in those two states are different, contradicting the assumption of the proposition. \( \square \)

**Definition 7 (Complementary Path)** A complementary path (CP) \( P \) is a path through the state graph whose labels form a complementary set of transitions, i.e., \( t' \in P \rightarrow t' \in P \) i.e., if a transition occurs along the path, then so does the complementary transition.
A complementary path at the state graph level that contains transitions from a complementary set implies that at STG level, all transitions in the set can occur without any other "unpairing" transition firing. Our task is therefore to identify such all-paired transitions in the STG graph.

**Theorem 3** For an MDG with no free-choice nodes, there exists a complementary path in the state graph iff the STG does not have the USC property.

*Proof.* (If part). If the STG does not have the USC property, then this implies that there are two distinct states with state identical vectors as labels. Since the two states are connected, a path from one to the other must exist; further, by Proposition 1, this path must be a complementary path.

(Only if part). Here, we have to prove that the existence of a complementary path implies that the STG does not satisfy the USC property.

Consider the identically labeled states at the beginning and end of such a complementary path. Two cases can arise:

- If the two states indeed correspond to distinct markings, then the STG clearly does not satisfy the USC property.

- It is possible that the beginning and end states are identical i.e., that they correspond to the same marking. In such a case, it can either be the case that all transitions have fired, or that only a subset of the transitions have fired. If all of the transitions have fired, thus causing the net to return to the original marking, then the path by definition is not a complementary path.

If only a subset of the transitions have fired, then it must be the case that there is a set of disjoint or overlapping loops at the node corresponding to this state (Figure 3). This is because the assumption that the STG is live implies that all of the transitions much be reachable from this state; thus there must be some path from s containing the transitions not covered by a give path P. However, in either case, this entails the existence of a state from which there is a fork, which corresponds to a choice point in the initial STG. However, this contradicts our assumption that the initial STG is an MDG without any free choice nodes. This completes the proof.

![Figure 3: Complementary Paths from the state S.](image)

The implication of this theorem is that in order to determine whether an STG violates the USC property, we have only to check for the presence of all-paired transitions in the STG corresponding to the complementary paths in the SG (provided that the STG is a marked graph with no FC nodes). Note that a CP in SG corresponds to a path or a group of paths in STG.

To find all-paired transitions starting with transition $x^*$, we first need to find all transitions that must fire between $x^*$ and $z^*$.

**Proposition 2** Consider the set of paths between transitions x and y with an initial marking (tokens) in an STG. Transitions that must fire after x and before y are the transitions along a path that contains the minimum number of tokens.

*Proof:* Consider the scenario after x is fired. Each fanout branch of x, including all paths leading to y, will have a token. Note that the firing of y requires each of the fanin branches of y to have a token. Because the STG is an MDG, a token from x is needed for each path between x and y before y can fire. (Non-MDG can have or-join nodes and a join node can have token from either input branch. y may thus fire without waiting for x's token propagation if the join node received tokens from other branches.) Because the STG is live, between two x firings, transitions along all branches having the minimum number of tokens have to fire to let the tokens from x to propagate down to y's input places before y can be fired. The other paths (branches) that have more than the minimum number of tokens do not have to fire all transitions to supply y the needed token, because of the extra token(s) they possess. The transitions along those branches can actually
happen concurrently with x and y. □

Note that these paths with minimum number of tokens are the true paths or executable paths in an STG, which has a corresponding path in the state graph. From now on, whenever we say path in an STG, we mean an executable path in the STG.

**Lemma 1** The minimum token strategy in Proposition 2 yields all executable paths starting with $x^*$. Consider transitions $t_1$ and $t_2$ with a set of concurrent chains $\{C_1, ..., C_n\}$, $C_i = \lt t_1, ..., t_n, t\gt$, that begin with $t_1$ and end with $t_2$. In the SG, these concurrent chains lead to a set of interleaving paths between the state after $t_1$ and the state before $t_2$ since the transitions in $C_i$ can occur concurrently with transitions in $C_j$, $j \neq i$. This set of interleaving paths forms an n-dimensional "lattice structure" as shown in Figure 4.

Since there can be many CPs in an SG, we would like to reduce the solution space and the search space we have to compute and report. Observe that non-minimal CPs are the CPs which start with $t$ but also contain concurrent transitions with $t$. A minimal CP starting with $t$ is thus representative of all CPs starting with $t$ and indicative of where the USC violations are. Minimal CPs are thus good candidates for USC violation detection and for complexity reduction. We will look only for the minimal CPs.

**Lemma 2** It is enough to detect minimal CPs to determine whether or not an STG has USC.

*Proof.* By the definition of minimal CP and by theorem 3; if all minimal CPs (N of them, one CP($t_i$) for each transition $t_i$) are of length $N$, then the STG has no USC violations. Otherwise, the STG does not have the USC property. □

**Lemma 3** If there is a minimal CP with length $\neq N$, the STG has no USC property.

Consider the "search space" of paths commencing with transition $t$. Note that all paths in the search space start from vertices in $L(t)$, by definition. Consider the following search strategy. Start from $s$=RHC($t$). Find all the transitions $\{y_i\}$ that have to fire before $t$' can fire. For each $y_i$, if $y_i$ and $t$ are not concurrent, find a!\] the the transitions $\{r_i\}$ that have to fire before $y_i$ can fire. If $y_i$ and $t$ are concurrent, find all transitions $\{z_i\}$ between $y_i$ and any transition that has already been examined. Let $\{z_i\}$ or $\{y_i\}$ be $\{y_j\}$. We can then repeat this search process until all $\{y_j\}$ are exhausted and no new transitions are introduced.

**Lemma 4** The above search strategy is sufficient to detect USC violations.
Proof. If none of the transitions introduced and examined are concurrent with \( t \), then clearly we have found the minimal \( CP(t) \); by construction, only the transitions have to fire to make a CP starting from \( t \) are transitively included. If any other CP is shorter, then at least a pair of transitions that are not needed are in the detected CP. This contradicts to the fact that only the transitions must fire are included.

If a transition \( x \) found in the process is concurrent with \( t \), then by definition \( x \) belongs to the lattice structure containing \( t \). Note that the search strategy says that we should search the path from \( x \) to an already examined transition, which is equivalent to moving the starting search point "backwards" from \( RHC(t) \) to a state \( m \in L(t) \), where both \( x \) and \( t \) are enabled. Now, if a CP, \( A \), starting from \( RHC(t) \) and including \( x \) exists, then some state \( s \) in the lattice has the same state assignment with \( RHC(t) \). Since transitions of the same variable cannot be concurrent, each path \( P_i \) in the SG along every axis between \( s \) and \( RHC(t) \) is either a CP or an empty set. If there is a \( P_i \neq \emptyset \) starting with a transition \( u \), then that simple CP will be detected when we examine \( CP(u) \).

Similar argument can be made after a "backward" movement. In that case, the same reasoning applies, except that the new starting corner \( s \) is used instead of \( RHC(t) \). Therefore we conclude that if there is a \( CP(t) \), it will either be detected directly or be detected when we examine some other transitions in the same lattice structure. By Lemma 3 this completes our proof. \( \square \)

Actually, if any path \( P_i \neq \emptyset \), where \( i \) is not the axis containing \( x \), then after moving backwards to \( s \), we will find \( x, CP_{\text{along_axis_containing } x}(A/x) \), which is also a CP. The above proof also illustrates that the search strategy uses the path with starting state \( RHC(t) \) to represent paths starting with any node of entire \( L(t) \). Concurrent transitions during the examination causes the starting point to move "backwards" and to represent a smaller number of nodes in the layer.

3.1 Algorithm USC-Check

The previous study and findings can be summerized as the following algorithm.

for each transition \( x^* \) in the STG {
  let \( P = \emptyset \), \( X^*/\text{Initialize} \)/
  Add \( X^* \) to an empty Queue.
  do while ((Queue not empty) and (\( P \neq N \))) {
    Take the first element out of the queue, say \( Y^* \)
    if \( Y^* \) not concurrent with \( t \) {
      for each transition \( Z^* \) along an executable path from \( X^* \) to \( Y^* \) {
        if \( Z^* \) do not belong to \( P \), \{ \}
        Add \( Z^* \) to the queue.
      }
    } else if \( Y^* \) concurrent with \( t \) {
      for each transition \( Z^* \) along an executable path from \( Y^* \) to any already examined node \( \text{in} \ L(t) \) {
        if \( Z^* \) do not belong to \( P \), \{ \}
        Add \( Z^* \) to the queue.
      }
    }
    If \( P \) is different from the set of all transitions then report the complementary path \( P \).
  }

Concurrent transitions can be determined by preprocessing. The above procedure is also quite suitable for "visual inspection," as the path and concurrency information can be easily determined. (Note that an easy visual procedure may not be computationally simple.) The number of paths between two transitions depends on the complexity of the design itself and is worst case exponential in the total number of arcs. In real designs, the worst case scenario rarely occurs. The complexity of Algorithm USC-Check is path oriented, whereas the cut-set method in [2] is state oriented.

4. An Example

We now illustrate the operation of the algorithm by using the example shown in Figure 1.

1. Let us first consider the transition \( z^+ \). Only \( z^- \) is concurrent with \( z^+ \). According to the algorithm, the queue is initialized with \( z^- \), and \( P \) to the set \( \{ z^-, z^+ \} \).
   - Since \( z^- \) is the only element in the queue, it is the first transition considered in the next stage
of the algorithm, which examines all the paths between \( z^+ \) and \( z^- \).

- Since the path from \( z^+ \) to \( z^- \) includes \( y^- \), the transitions \( y^- \) and \( y^+ \) are added to \( P \). We also add \( y^+ \) to the queue. Thus, \( P = \{ z^+, z^-, y^+, y^- \} \), and \( Q = \{ y^+ \} \).

- Since the path from \( z^+ \) to \( y^+ \) includes \( z^+ \), the transitions \( z^+ \) and \( z^- \) are added to \( P \). At this stage, \( P = \{ z^+, z^-, x^+, x^-, z^+, y^+, y^- \} \) which is the set of all transitions. Also, \( Q = \{ z^- \} \).

The iteration is terminated, since \( P = N \) at this stage. This implies that no complementary path starts with \( z^+ \) as its first transition.

2. Let us now consider the transition \( z^- \). Since the path \( z^- \) to \( z^+ \) includes no other transitions, the procedure terminates and reports the complementary path \( P = \{ z-, z^+, y^- \} \). This complementary path will lead to a violation of the USC property, as may be seen from the state graph.

3. Similarly, considering the transition \( z^- \) shows that \( \{ z^-, z^+ \} \) is a complementary path.

4. No complementary path starts with the transition \( z^+ \).

5. Let us now consider the transition \( y^+ \). Only \( z^- \) is concurrent with \( y^+ \). The path between \( y^+ \) and \( y^- \) includes only \( z^+ \), which in turn introduces \( z^- \) into \( P \). Since the path between \( z^- \) and \( z^+ \) introduces no new transitions and the queue is empty, this yields another complementary path \( \{ y^+, z^+, y^- \} \).

Further note that since \( z^- \) can occur after \( z^+ \), and \( \{ y^+, z^- \} \) are concurrent (they are not in a simple cycle), \( \{ y^+, z^+, y^- \} \) is another possible sequence of transitions that will lead to two states having identical codes.

Similarly (observe that \( x \) and \( z \) are almost symmetrical in this graph) the paths \( \{ y^- \}, \{ x^-, y^+, y^- \} \) and \( \{ y^-, z^-, x^+, y^+ \} \) are complementary paths detected upon considering the transition \( y^- \). Figure 2 shows the state graph of this STG specification, and we can verify that the above paths indeed cause violations of the USC property.

4.1 Modifying an STG to satisfy the USC property

The complementary paths detected by the algorithm actually suggest how the STG can be modified such that it has the USC property.

One approach is to add arcs in the graph such that the offending paths no longer exist. That is, given such offending paths, we want to add arcs to make each chain to include all transitions (see Figure 5). This approach leads to persistent or semi-modular designs[9]. For example, the chain \( \{ y^+, z^+, y^-, z^- \} \) needs to "lock" in transition \( z^+ \) or \( x^- \). One way to achieve this is to add the arc \( x^- \rightarrow y^- \). This addition also resolves the chain \( \{ z^-, z^+ \} \) as a result of the newly created path \( \{ x^-, y^+, z^+ \} \). This is because in considering the transition \( x^- \), the newly introduced path causes \( y^+ \), and therefore eventually \( z^+ \) to be added to \( P \), thus yielding \( P = N \). Similarly, adding the arc \( z^- \rightarrow y^+ \) resolves the other offending path.

![Figure 5: A persistent STG obtained by adding arcs.](image)

One drawback of this approach is that the specified behavior can change erratically and not necessarily match the designer's intentions. Further, it has been shown that that semi-modularity is neither necessary nor sufficient for hazard-free circuit implementations[12], [6]. This reduces the advantage of such an approach, since the added arcs are tantamount to additional constraints and usually translate into less concurrent (slower) implementations.

An alternative approach (which we advocate here) is to add transitions (nodes) such that the offending paths become the ones including all transitions. The added transitions are new variable transitions, and are called internal variables in [2]. They can be considered as
memory to distinguish otherwise identical states. For example, the chain \( \{ z^-, z^+ \} \) suggests that one new transition has to be included and only two positions are possible, between \( z^- \) and \( J2 \) or between the join \( J2 \) and \( z^+ \).

- If we choose to add a new variable transition \( X1^+ \) between \( J2 \) and \( z^+ \) (Figure 6), the transition \( X1^- \) can be added between \( J1 \) and \( z^+ \). These transitions serve to eliminate all the remaining complementary paths.

- If we choose the other alternative available, i.e., introducing a transition \( X1^+ \) between \( z^- \) and \( J2 \), the complementary path \( \{ y^+, z^+, y^-, z^- \} \) still remains. In addition, \( X1^- \) cannot be placed between \( z^- \) and \( z^+ \) to resolve the chain \( \{ z^-, z^+ \} \), as \( X1^- \) and \( X1^+ \) have to be in a simple cycle. Consequently, a second variable transition has to be introduced, as shown in Figure 7. This two-variable solution is similar to the solution produced by the lock-graph based approach discussed in [11].

Since it is desirable to maximize the effect of each variable addition, i.e., to minimize the number of variables added, the first solution is preferable.

![Figure 6. Introduction of New Variables.](image)

The advantage of this approach is that the behavior change is minor and predictable. Note that it may sometimes be undesirable to add new transitions that precede input variables due to the implications on the circuit performance. This is because the circuit will be slowed down by the added delay needed to compute the new variables. In such cases, a sub-optimal number of variables may be needed to achieve the USC property. Additionally, it needs to be ensured that the new up and down transitions for each variable are ordered and not concurrent.

![Figure 7. (Suboptimal) solution with 2 variables.](image)

5. Input Set Determination

Given an STG \( <V, E> \) and a node set \( SV \), the contracted STG with respect to \( S \) is obtained by eliminating the nodes not in \( S \), and preserving the dependency between nodes in \( S \).

Informally, the input set of a signal \( s \) is the set of signals used in the logic generating the signal \( s \). It was initially incorrectly believed to be the set of immediate predecessors of \( s^+ \) and \( s^- \). Here, we define the input set to be the minimum set of signals including the immediate predecessors of \( s^+ \) and \( s^- \) whose contracted graph has the USC property.

Using the procedure to determine whether an STG has the USC property, it is possible to find out the input set of a signal \( s \). We start with a contracted STG for the given set of (input) signals. This set is initially the set of immediate predecessors of \( s^+ \) and \( s^- \). We check to see if the contracted STG has the USC property. If it does not, then we greedily choose another signal such that including it will minimize (reduce) the number of complementary paths reported by our algorithm on the new contracted graph. This is repeated until the STG satisfies the USC property. This is always feasible if the initial STG is assumed to have the USC property, since in the worst case all variables will be included in the input set.

**Proposition 3** The set computed by the above procedure includes an input set of a signal.

[10] suggested a method for determining the input set of a signal using the algorithm based on lock graphs; however, the input set computed by this technique can sometimes be pessimistic. For example, in the STG shown in Figure 8, the lock
graph indicates that Za and Lr are connected through Zr. Zr is thus a “context signal” and is a direct input to Lr. However, the logic module used to generate Lr does not need Zr, indicating that the input set computed using the lock graph is pessimistic. In contrast, using the above procedure the input set for Lr is computed to be \(\{Dr^+, Dr^-, Za^+, Za^-, x^+, x^-, z^+, z^-\}\), since the contracted graph in Figure 9 does not have any complementary paths.

![Figure 8. An STG and its Lock Graph. The lock graph indicates that Zr ∈ InputSet(Lr).](image)

![Figure 9. Contracted STG with respect to \(\{Lr^+, Lr^-, Dr^+, Dr^-, Za^+, Za^-, x^+, x^-, Zr\}\).](image)

6. Summary

We have described an algorithm to ascertain whether a given STG has the USC property. The algorithm is path-oriented and operates directly on the STG rather than a state graph that is derived from the STG. This approach has the advantage of being easier to visually and intuitively correlate with the STG specification, and therefore suggests ways in which a designer (or tool) may modify the input STG if it does not satisfy the USC property. We have also indicated how the technique can be used to compute the input set of a signal s; this set is the set of signals used by the logic to compute s.

REFERENCES