A General State Graph Transformation Framework for Asynchronous Synthesis

Bill Lin[a] Chantal Ykman-Couvreur[a] Peter Vanbekbergen[b]

[a]IMEC Laboratory, Kapeldreef 75, B-3001 Leuven, Belgium
[b]Synopsys Inc., 700 E. Middlefield Rd., Mountain View, CA 94043-4033

Abstract — A general framework for synthesis of asynchronous control circuits at the state graph level is proposed. The framework can consider both concurrency reduction as well as new state signal insertion during the transformation process. Considering concurrency reduction in the exploration space is crucial because more area efficient and higher performance circuits may be possible. This is partly because often fewer new state signals are required and the resulting logic is typically more highly unspecified, thus leaving more room for optimizations at the logic level. Considering new state signal insertion is also crucial as new signals are usually required for disambiguating state coding conflicts. A larger solution space can be searched when both classes of transformations are considered. The new framework has been implemented and verified on a large set of realistic design examples.

1 Introduction

Automated synthesis procedures for asynchronous control circuits from formal specifications are becoming essential for system-level design. In this paper, a general framework for synthesis of asynchronous control circuits at the state graph level is proposed. Formulating the problem at the state graph level has several notable advantages. First, the developed synthesis methods are generally applicable to different well-formed high-level specification models since it is possible to translate them to the state graph level. For example, very general forms of signal transition graphs (STGs) [2, 9] can be translated to equivalent state graphs. Second, it is possible to perform important transformations at the state graph level that are difficult or impossible to formulate on a high-level model (e.g. STGs).

Consider a simple state graph shown in Figure 1(a). This specification cannot be directly implemented because the initial state graph contains two states with the same code, namely it violates the so-called *complete state coding* (CSC) requirement. This means that a *race* condition can occur. If each non-input signal is to be implemented with a single complex gate and this complex gate contains no *internal* hazards, then it has been shown that the necessary and sufficient condition for a hazard-free implementation is precisely characterized by the CSC requirement. A state graph satisfying the CSC requirement may also be implemented in a hazard-free manner using only basic gates by employing the methods described in [5, 1].

In [8], a generalized state assignment method was proposed for transformations at the state graph level. This method has the capability of adding new state signal transitions in a concurrent way. For example in Figure 1(a), the CSC violation can be corrected by adding a new state signal C, as shown in Figure 1(c) and (d). This process is referred to as *state graph expansion*. While the transformations that can be performed by this process are quite powerful, they cannot arrive at solutions that require the *reduction of concurrency*.

Referring again to Figure 1(a), the CSC violation can instead be corrected by forcing $A+$ to always proceed $B+$. This effect can be achieved at the state graph level by removing the first 01 state from the state graph, as shown in Figure 1(b). The solution shown in Figure 1(b) reduces to simply an inverter with only *1 literal*. On the other hand, the solution shown in Figure 1(d) requires *8 literals* to implement. The solution here with concurrency reduction is clearly faster as well. Therefore, the common assertion that maintaining concurrency is crucial for performance is false. Let us examine briefly why the consideration of concurrency reduction in the exploration space is important for both area efficiency and performance: (1) In a highly concurrent specification, the number of unreachable states is limited. Consequently, the Boolean functions can be highly specified so that logic optimization is limited. Concurrency reduction can increase the number of unreachable states in the transformed state graph, thus providing more *don't care conditions* for logic optimization. (2) With concurrency reduction, fewer state signals may be required to disambiguate the states. Thus, fewer next-state functions are required, and the required Boolean functions may depend on fewer support variables. In addition, requiring extra state signal transitions may also delay the output signal transitions, and hence can also increase the delay of the final circuit.

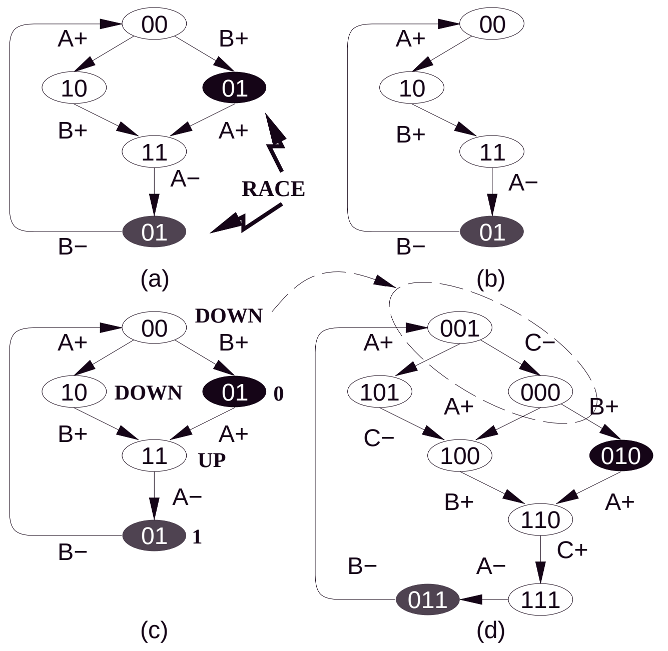

Figure 1: (a) Initial state graph with CSC violation. (b) Correction via concurrency reduction. (c) Correction by generalized state assignment. (d) Expanded state graph.

In general, both concurrency reduction and the insertion of new state signals are required to search a larger solution space. Considering only the solution space without concurrency reduction may leave out many potentially important solutions, solutions that may be more optimal with respect to area and performance. Although the importance of concurrency reduction has been recognized, the existing methods have rather severe restrictions. For example, a technique has been proposed for STGs [7], but it is restricted only to *marked graphs*.

In this paper, we describe a general framework that considers both concurrency reduction and new state signal insertion transformations in a unified solution framework that works purely at the state graph level. Specifically, we describe two new methods of concurrency reduction and show how each method can be combined with the generalized state assignment method. The first method is based on a pairwise reduction between a pair of *concurrent transitions*. This pairwise reduction method can be combined with generalized state assignment in an *iterative* manner. The second method is based on formulating the concurrency reduction step as a *Boolean satisfiability* problem by defining a *reduction function* that transforms the state graph by removal of states. Correctness conditions are formulated as Boolean constraints. This approach can be combined with the generalized state assignment method in a *"compositional"* way by solving both the concurrency reduction and state assignment constraints [8] together in a satisfiability framework.

2 State Graph Specification

2.1 State graphs

A state graph is a finite automaton given by $G = \langle A, S, T, \delta, s_o \rangle$, where the components are defined as follows.

$A = I \cup O$ is the set of *signals*, I is the set of *input* signals, and O is the set of *non-input* signals, such that $I \cap O = \emptyset$.

$T = A \times \{+, -\}$ is the set of *signal transitions*, where $a+$ denotes the $0 \rightarrow 1$ transition of signal a and $a-$ denotes the $1 \rightarrow 0$ transition of signal a. $a*$ is used to depict either $a+$ transition or a $a-$ transition.

$\delta : S \times T \mapsto S$ is a partial function representing the *transition function* such that if $\delta(s, t) = s'$ is defined then t is said to be *enabled* in state s and the *firing* of t takes the system from s to s'. This is denoted as $s \xrightarrow{t} s'$. It can also be denoted as $s[t\rangle s'$. We will use $s[t\rangle$ to denote that t is enabled in state s. To depict the *firing* of a *sequence* of transitions $\sigma = t_1 t_2 \ldots t_m$, the notations $s \xrightarrow{\sigma} s'$ and $s[\sigma\rangle s'$ are used. We will use $s[\sigma\rangle$ to denote that σ is a possible firing sequence in state s.

$s_0 \in S$ is the *initial state*. Each state in the state graph is labeled with a *binary vector* $\langle s(1), s(2), \ldots, s(n) \rangle$ according to the signals $A = \{a_1, a_2, \ldots, a_n\}$ of the system. The labeling is given by a *state assignment function* $\lambda_A : S \times A \mapsto \{0, 1\}$. For a given state $s \in S$, $s(i)$ denotes the i-th component of s corresponding to the value of signal $a_i \in A$. The state assignment function is defined as follows: (1) if $s(i) = 0 \wedge t = a_i+$ then $s'(i) = 1$; (2) else if $s(i) = 1 \wedge t = a_i-$ then $s'(i) = 0$; (3) otherwise, $s'(i) = s(i)$. If the states of a state graph can be encoded according to the above rules, then the state graph is said to have a *consistent state assignment*.

In a state $s \in S$, if a transition t is enabled, the corresponding signal a_i is said to be *excited* (which can be denoted pictorially with an asterisk $*$).

Since a state graph is represented as a directed graph, it is often useful to refer to its arcs, which we can denote as $E \subseteq S \times S$. $(s, s') \in E$ if and only if $\exists t \in T : s \xrightarrow{t} s'$ is defined. That can be denoted by the relation $sE(t)s'$.

A state in the state graph captures the state of all signals in a circuit, while a transition between states is a transition of exactly one signal. There may be many signals enabled in a state, but exactly one signal transition is fired at a time. This corresponds to the *interleaved concurrency* model where enabled transitions can fire in any order, but only one is fired at a time.

2.2 Objects and properties

In this section, we formally define some basic objects and properties that we will be using to formulate our framework. We first define some properties of state graphs.

Definition 2.1 (Semi-modularity) *Let G be a state graph. A* **transition** *t is said to be* **semi-modular** *with respect to $u \xrightarrow{t} v$ in G if and only if t remains enabled in all states w' where $\exists t' \neq t : u \xrightarrow{t'} w'$. A* **signal** *$a$ is said to be semi-modular in G if and only if all transitions $a*$ in G are semi-modular. A state graph G is said to be semi-modular if and only if all of its signals $a \in A$ are semi-modular. A state graph G is said to be* **output semi-modular** *if and only if all of its non-input signals $a \in O$ are semi-modular.*

In a state graph specification, concurrency is represented indirectly using *interleaving*. In order to derive a correspondence between *actual* signal transitions and states in the state graphs, different regions are defined as follows:

Definition 2.2 (Excitation region) *An* **excitation region** *of signal a in state graph G is a maximal connected set of states in which a has the same value and is excited.*

The excitation region corresponding to transition a_i* will be denoted as $ER(a_i*)$. Note that there can be several excitation regions for a corresponding to multiple transitions of a.

Definition 2.3 (Switching region) *A* **switching region** *of transition a_i*, denoted as $SR(a_i*)$, is the set of all states that are reachable from $ER(a_i*)$ by a firing of transition a_i*.*

More formally, state $u \in SR(a_i*)$ if and only if state v exists in state graph G, $v \in ER(a_i*)$, and $v \xrightarrow{a_i*} u$.

Definition 2.4 (Ordered and concurrent set) *A signal b is said to be* **ordered** *with respect to transition a_i* if no transition of signal b is excited within $ER(a_i*)$. Otherwise b is said to be* **concurrent** *with a_i*.*

A condition called the *Complete State Coding* requirement was proposed and proved as a necessary and sufficient condition by Chu [2] for a race-free implementation.

Definition 2.5 (Complete State Coding (CSC)) *A state graph is said to satisfy the* Complete State Coding *requirement if and only if (1) each pair of states have unique binary codes, or (2) for pairs of states having identical binary codes, the set of excited non-input signal transitions must be identical.*

If all states in a state graph have unique binary codes, then the state graph is said to satisfy the *Unique State Coding* (USC) requirement.

2.3 Conformance and the solution space

In this section, the solution space being considered in this work is defined. Our first correctness condition is related to Dill's notion of *conformance* [3]. Here, we give a variant definition for conventional state graphs. Let $G = \langle A, S, T, \delta, s_0 \rangle$ be a state graph. A *trace* is any $r \in T^*$. The set of *success traces* accepted by G, denoted by $\mathcal{S} \subseteq T^*$, is the set $\{r \in T^* \mid s_0[\sigma\rangle\}$. For every trace in $\mathcal{S}$, including an empty string, $\mathcal{S}$ describes the next possible *input* and *non-input* event that is *accepted*. Since a real circuit *cannot* control its environment, it must be *receptive* to any input event at all times. When an input event, $\mathcal{I} = I \times \{+, -\}$, is not accepted following a partial trace r, meaning $r\mathcal{I} \notin \mathcal{S}$, then $r\mathcal{I}$ and all subsequent traces $r\mathcal{I}T^*$ are said to result in a *failure*. All such *failure traces, $\mathcal{F} \subseteq T^*$*, are formally defined by $\mathcal{F} = ((\mathcal{S} \cup \{\epsilon\})\mathcal{I} - \mathcal{S})T^*$.

Given a specification state graph G, we wish to define when a transformed state graph G' still *implements* G. Informally, G' implements G if the set of possible traces of G' over the set of signals that the environment can observe is a subset of the possible traces of G, and the possible *failures* of G' is a subset of possible failures of G.

Assuming the set of *input* and *non-input* signals for the specification state graph G and the transformed state graph G' are identical, then *conformance* can be defined as follows.

Definition 2.6 (Conformance I) *Let G' and G be two state graphs such that $I' = I$ and $O' = O$, and let $\mathcal{P}' = \mathcal{S}' \cup \mathcal{F}'$ and let $\mathcal{P} = \mathcal{S} \cup \mathcal{F}$. Then G' is said to conform to G, denoted by $G' \subseteq G$, if and only if $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{F}' \subseteq \mathcal{F}$.*

With state assignment, it is possible that the transformed state graph G' has more signals than the specification state graph G, meaning $O' \supseteq O$; however, $I' = I$ must remain the same. The set of (state) signals $O' - O$ are said to be *non-observable*.

We can define a **hide** operation on a state graph G with respect to a subset of signal $A_D \subseteq A$, denoted with **hide**$(A_D)(G)$, as follows. Replace all transitions $T_D = A_D \times \{+, -\}$ in G with an ϵ transition and then *determinize* the resulting state graph by means of *epsilon closure* [4]. The state assignment function λ_A is modified accordingly. This is equivalent to the **hide** operation in [3]. Then the definition of *conformance* is modified as follows.

Definition 2.7 (Conformance II) *Let G' and G be two state graphs such that $I' = I$ and $O' \supseteq O$. Then G' is said to* **conform** *to G, denoted by $G' \preceq G$, if and only if* **hide**$(O' - O)(G') \subseteq G$.

Note that conformance is defined as a property on traces rather than on state graphs. Any concurrency reduction transformations must ensure that the conformance property is preserved. This implies all input concurrency and input choices must be preserved; i.e., if an input event is possible in the specification state graph, it must remain possible in the transformed state graph.

In addition to ensuring conformance, the transformed state graph G' should also satisfy the following requirements in order for it to be implementable with respect to a specification state graph G. (1) *Complete state coding*: G' must satisfy the CSC requirement. (2) *Semi-modularity*: Every signal transition that is semi-modular in G should also be semi-modular in G'. Every new state signal transitions introduced should also be semi-modular in G'. Preserving semi-modularity ensures that an output semi-modular state graph remains output semi-modular. Otherwise, arbitration circuitry must be synthesized to handle the output non-determinism. Though this is theoretically possible, we exclude it from our solution space to ensure existing synthesis methods can be used to build hazard-free circuits.

3 Heuristic Pairwise Reduction

3.1 Reduction operator

In this section, we formulate how concurrency reduction can be solved at the state graph level. Intuitively, concurrency reduction involves *removing* some possible traces. Recall that state graphs use an *interleaved concurrency* model. This means that in a given state there may be many *concurrent* transitions *enabled* (cf. Definition 2.4), but exactly one may *fire* at a time. The state graph represents different firing orders of concurrent transitions. We can reduce the concurrency with respect to a *pair* of concurrent transitions $\langle t_0, t_1 \rangle$ in a state graph G by imposing that the transition t_0 must first fire *before* t_1 (or vice versa). This essentially corresponds to the *removal* all traces $r \in \mathcal{S}$ where t_1 is the first to appear.

Consider the example shown in Figure 2. Transition $a+$ is concurrent with $b+$, $c+$, and $d+$. We can reduce the concurrency of this state graph with respect to $\langle c+, a+ \rangle$ by transforming it so that $c+$ always proceed $a+$. At the state graph level, removing traces where $a+$ fires first can be achieved by removing states where $a+$ has fired before $c+$. However, the removal of states cannot be performed arbitrarily since we must ensure that the transformed state graph still *conforms* to the specification state graph and that *semi-modularity is preserved*.

Definition 3.1 (Pairwise reduction) *Let $G = \langle A, S, T, \delta, s_0 \rangle$ be a finite connected state graph with a consistent state assignment. Let $\langle t_0, t_1 \rangle$ be two concurrent transitions by Definition 2.4, such that t_1*

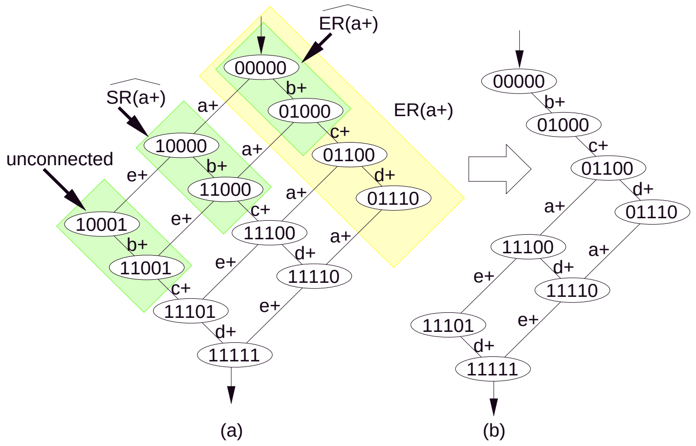

Figure 2: (a) In this state graph, transitions $a+$ and $c+$ are *concurrent*. We can reduce the concurrency by forcing $c+$ to proceed $a+$, resulting in the state graph shown in (b).

is not an input transition. (i.e. $t_1 \notin I \times \{+, -\}$.) Then the **pairwise reduction** *of G with respect to $\langle t_0, t_1 \rangle$, denoted by $G_{|t_0 \mapsto t_1}$, is a new state graph $\langle A, S_{|t_0 \mapsto t_1}, T, \delta_{|t_0 \mapsto t_1}, s_0 \rangle$ derived as follows: (1) Let $ER(t_1)$ be the excitation region t_1, and $SR(t_1)$ be its corresponding switching region. (2) Let $\widehat{ER(t_1)} \subset ER(t_1)$ be the maximally connected subset of states in $ER(t_1)$ before encounting a t_0 transition, and $\widehat{SR(t_1)}$ be the corresponding switching region. (3) Remove all states in $\widehat{SR(t_1)}$ and all associated fanin and fanout arcs. (4) Remove unreachable (dead) states. $S_{|t_0 \mapsto t_1} \subset S$ is the set of remaining states and $\delta_{|t_0 \mapsto t_1}$ is δ restricted to $S_{|t_0 \mapsto t_1}$.*

Note that the operator is only defined when t_1 is *not* an input transition. We illustrate the reduction operator on the example shown in Figure 2 by reducing the state graph with respect to the concurrent transition pair $\langle c+, a+ \rangle$, meaning we wish to impose that $c+$ is *ordered* before $a+$. By definition of concurrent transitions, $c+$ can fire inside $ER(a+)$. We first identify $\widehat{ER(a+)} \subset ER(a+)$ where $a+$ is enabled to fire, but $c+$ has not yet fired – namely the states 00000 and 01000. To prevent $a+$ from firing first, we remove all states in $\widehat{SR(a+)}$ – namely 10000 and 11000. This is according to rule (3). We also remove states 10001 and 11001 because they are now no longer reachable, according to rule (4).

Theorem 3.1 (Conformance preservation) *Let G be a finite connected state graph with a consistent state assignment, $\langle t_0, t_1 \rangle$ be two concurrent transitions, and $G_{|t_0 \mapsto t_1}$ be a reduced state graph derived according to Definition 3.1. Then $G_{|t_0 \mapsto t_1} \preceq G$.*

Theorem 3.2 (Semi-modularity preservation) *Let G be a finite connected state graph with a consistent state assignment, $\langle t_0, t_1 \rangle$ be two concurrent transitions, and $G_{|t_0 \mapsto t_1}$ be a reduced state graph derived according to Definition 3.1. Then every transition t semi-modular in G remains semi-modular in $G_{|t_0 \mapsto t_1}$.*

3.2 Combining with state assignment

Let G be an initial state graph. The reduction operator described in the previous section can be combined with the generalized state assignment method [8] to *iteratively* search the solution space. G can be first reduced to minimize the number of CSC violations. The remaining CSC violations can be solved by applying the generalized state assignment method to insert new state signals and transitions. By iterating these two types of transformations, a very large solution space can be explored.

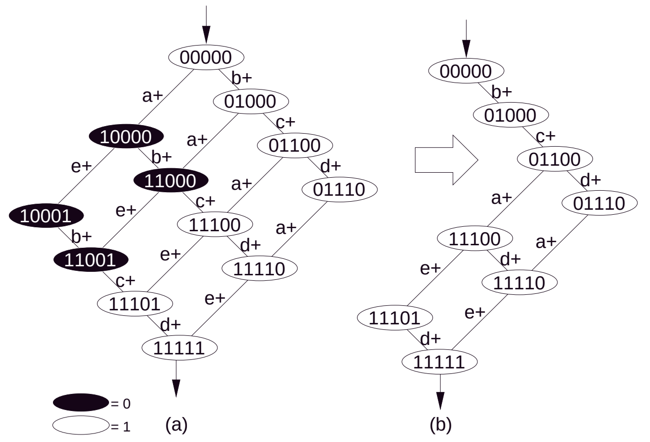

Figure 3: (a) Initial state graph. (b) Transformed state graph.

4 Global Satisfiability Approach

4.1 Formulation

In Section 3, it was shown that concurrency reduction essentially corresponds to the removal of traces, and that the removal traces can be achieved at the state graph level by the removal of states. The reduction operator described in Section 3 reduces *one pair* of concurrent transitions at a time.

In this section, we describe an alternative approach that builds a set of *Boolean constraints* to characterize all possible sets of states that can be removed without violating the conformance condition and the semi-modularity condition. These Boolean constraints can be solved using any *Boolean satisfiability solver*. Formulating the problem in this way has two advantages: first, the approach can simultaneously consider the reduction of concurrency between several sets of concurrent transitions rather than just a single pair; second, since the problem is formulated using a Boolean satisfiability approach, it can naturally be combined with the Boolean satisfiability approach used for generalized state assignment [8]. In [8], a set of Boolean constraints was prescribed for solving the state assignment problem. The set of Boolean constraints for concurrency reduction can be combined with those for the state assignment problem to solve both concurrency reduction and state assignment simultaneously using a Boolean satisfiability framework.

The basic idea is to transform a state graph G into a new state graph G' by encoding each state by either a "0" or a "1". This is formalized by defining the notion of a **reduction function**. Let $G = \langle A, S, T, \delta, s_0 \rangle$ be the initial state graph. A **reduction function** is a function $\omega : S \mapsto \{0, 1\}$ on G that maps each state in G to either 0 or 1. The assignment $\omega(u) = 1$ means that the state u is *kept*, and the assignment $\omega(u) = 0$ means that u is *removed*. Given this intuition, we can define the *global reduction operator* as follows:

Definition 4.1 (Global reduction) *Let $G = \langle A, S, T, \delta, s_0 \rangle$ be a finite connected state graph with a consistent state assignment. Let $\omega : S \mapsto \{0, 1\}$ be a reduction function defined on G. Then the* **global reduction** *of G with respect to ω, written $\omega(G)$, is a new state graph $G_\omega = \langle A, S_\omega, T, \delta_\omega, s_0 \rangle$, where A, T, and s_0 remain unchanged. S_ω is defined as*

$$S_\omega = \{s \in S \mid \omega(s) = 1\},$$

and the transformed next state function δ_ω is defined as

$$\delta_\omega(s,t) = \begin{cases} \textit{undefined} & \textit{if } \delta(s,t) \textit{ is undefined} \\ \textit{undefined} & \textit{if } \omega(s) = 0 \\ \delta(s,t) & \textit{otherwise} \end{cases}$$

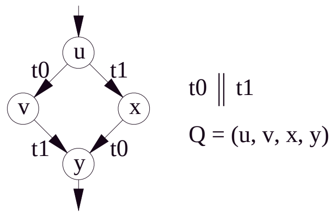

Figure 4: A quadrant configuration.

This essentially defines the meaning of $\omega : S \mapsto \{0, 1\}$ and how the new state graph can be derived. For example, in Figure 3, the states in "black" are assigned $\omega = 0$, and the states in "white" are assigned $\omega = 1$. The transformed state graph is shown in Figure 3(b). However, as with the pairwise reduction operation described in Section 3, concurrency reduction cannot be performed arbitrarily. Therefore, conditions must be imposed on the assignment function. That is, conditions must be imposed such that the transformed state graph G_ω remains a "valid" implementation of the initial state graph G.

Specifically, we need to formulate constraints to capture the following requirements: (1) *Conformance*, which corresponds to the preservation of input concurrency, input choices, and the initial state; and (2) *semi-modularity*. (3) In addition to these two types of constraints, constraints should also be imposed to ensure that the transformed state graph does not contain any unreachable (dead) states. These constraints are called *connectivity* constraints. (4) Finally, constraints are imposed to see if the transformed state graph satisfies the *complete state coding* property.

As in [8], we will formulate this as a Boolean satisfiability problem and express the constraints directly as Boolean constraints. We will use the notation ω_u to denote $\omega(u) = 1$, and the notations $\overline{\omega_u}$ or $\neg\, \omega_u$ to denote $\omega(u) = 0$. In fact, this directly corresponds to a Boolean variable ω_u and the corresponding two Boolean *literals*. The constraints are described next.

Conformance constraints: The first set of constraints are related to conformance.

Requirement CR1a. *All input transitions must remain enabled.* Let E be the set of state transition pairs (u, v) in G. Let $I \subseteq E$ be the subset of state pairs (u, v), $u \xrightarrow{t} v$, such that $t \in I \times \{+, -\}$. Then the requirement **CR1a** is simply expressed as

$$\prod_{(u,v) \in I} (\omega_u \Rightarrow \omega_v) \tag{1}$$

Requirement CR1b. *All input choices must remain possible.* Let CH be the set of triplets (u, v, x) such that each has the configuration $v \longleftarrow u \longrightarrow x$, and u corresponds to an *input choice* node. Then **CR1b** can be simply expressed as

$$\prod_{(u,v,x) \in CH} \big(\omega_u \Rightarrow (\omega_v \wedge \omega_x)\big) \tag{2}$$

Requirement CR1c. *Preservation of initial state.* This is simply $\omega_{s_0} = 1$.

Semi-modularity constraints: We next need to guarantee that every transition that was semi-modular in G remains semi-modular in G_ω. Informally, an *excited* transition t is semi-modular with respect to a state u if and only if in every reachable state from u, t can become stable only through firing it. To guarantee that the transformed state graph is consistent and satisfies semi-modularity, the following requirements must be satisfied.

Requirement CR2a. *Semi-modularity must be preserved.* To explain the semi-modularity requirement, consider a 4-state configuration $Q = (u, v, x, y)$, as shown in Figure 4, such that (1) $u, v, x, y \in S$, and (2) $\exists t_0, t_1 \in T : (u \xrightarrow{t_0} v) \wedge (u \xrightarrow{t_1} x) \wedge (x \xrightarrow{t_0} y) \wedge (v \xrightarrow{t_1} y)$. This is called a *semi-modular quadrant.* If state u is kept, but state y is removed, then we require that at least one of v or x must also be removed. If only one of v or x, or both are removed, then we assume some serialization of t_0 and t_1 has occurred and we leave to the other rules to ensure correctness. However, if both v and x are kept, this means that both t_0 and t_1 remain concurrent and can fire from state u. If state y is not kept, then this quadrant is no longer semi-modular. Hence, semi-modularity has not been preserved. To prevent this problem from occurring, the following constraint is applied. Let $\mathbf{QSET}$ be the set of all quadrants. Then the requirement can be expressed as follows:

$$\prod_{Q_i \in \mathbf{QSET}} (\omega_u \wedge \overline{\omega_y}) \Rightarrow (\overline{\omega_v} \vee \overline{\omega_x}) \tag{3}$$

Requirement CR2b. *The transformed state graph must remain consistent.* Give a concurrent transition pair $\langle t_0, t_1 \rangle$, there can be several quadrants in their excitation regions where t_0 and t_1 can concurrently fire. The transformed state graph is inconsistent if concurrency is removed in some of these quadrants but not others. This corresponds to a kind of metastability problem. To prevent this, we require that every quadrant where the head state is kept must be modified the same way. For example, consider two quadrants $Q_1 = (u_1, v_1, x_1, y_1)$ and $Q_2 = (u_2, v_2, x_2, y_2)$. Suppose both u_1 and u_2 are kept. Then if v_1 is kept, v_2 must also be kept; if x_1 is removed, x_2 must also be removed, and so forth. To express this requirement, we define the following.

Let CCR be the set of all transition pairs $\langle t_0, t_1 \rangle$ such that $t_0 \| t_1$. Consider a concurrent transition pair $\langle t_0, t_1 \rangle$, let $Q_1, Q_2, \ldots, Q_n$, where $Q_i = (u_i, v_i, x_i, y_i)$, be the different quadrants in the state graph where t_0 and t_1 can fire. Let $\mathbf{QSET}(t_0, t_1) \subseteq \mathbf{QSET}$ be the set of all such Q's. For all quadrants, the following must be satisfied:

$$\prod_{\langle t_0, t_1 \rangle \in CCR} \left(\begin{array}{l} \left(\prod_{Q_i \in \mathbf{QSET}(t_0,t_1)} \omega_{u_i} \Rightarrow \omega_{v_i}\right) \vee \\ \left(\prod_{Q_i \in \mathbf{QSET}(t_0,t_1)} \omega_{u_i} \Rightarrow \neg\, \omega_{v_i}\right) \end{array} \right) \tag{4}$$

$$\prod_{\langle t_0, t_1 \rangle \in CCR} \left(\begin{array}{l} \left(\prod_{Q_i \in \mathbf{QSET}(t_0,t_1)} \omega_{u_i} \Rightarrow \omega_{x_i}\right) \vee \\ \left(\prod_{Q_i \in \mathbf{QSET}(t_0,t_1)} \omega_{u_i} \Rightarrow \neg\, \omega_{x_i}\right) \end{array} \right) \tag{5}$$

Connectivity constraints: Next we describe constraints that ensure that the transformed state graph is connected (meaning there are no dead or unreachable states). This can be expressed as two sets of constraints: *fanout connectivity* and *fanin connectivity*.

Requirement CR3a. *Fanout connectivity.* For a state u, let $FO(u) = \{v \in S \mid \exists t \in T : u \xrightarrow{t} v\}$ be the set of fanout states. If state u is kept, we must ensure that at least one fanout state is also kept. This is expressed as follows.

$$\prod_{u \in S} \left(\omega_u \Rightarrow \left(\sum_{v \in FO(u)} \omega_v \right) \right) \tag{6}$$

What this means is that at least one enabled event must be able to fire. Note that requirement **CR1a** is a stronger requirement since it requires all input events to remain enabled. Rule **CR3a** is applicable when some enabled transitions are output events.

Requirement CR3b. *Fanin connectivity.* For a state v, let $FI(v) = \{u \in S \mid \exists t \in T : u \xrightarrow{t} v\}$ be the set of fanin states. If all the fanin states of v have been removed, then v must also be removed. This is expressed as follows.

$$\prod_{v \in S} \left(\omega_v \Rightarrow \left(\sum_{u \in FI(v)} \omega_u \right) \right) \tag{7}$$

In addition to the connectivity rules, we must make sure that every transition in the initial state graph G must be able to fire in the transformed state graph G_ω. This means that a transition cannot *disappear*. We call this the *deadlock condition*.

Requirement CR3c. *Every transition in G must exist in the transformed state graph G_ω.* Let a_i* be a transition of signal a and $ER(a_i*)$ its excitation region. Then we can impose the following to ensure that a transition does not disappear.

$$\prod_{a \in A} \prod_{ER(a_i^*)} \sum_{u \in ER(a_i*)} \omega_u \tag{8}$$

These rules, together with the preservation of the initial state, guarantee the existence of a consistent state assignment after transformation.

Theorem 4.1 (Conformance and semi-modularity) *Let G be a finite connected state graph with a consistent state assignment, and $\omega : S \mapsto \{0, 1\}$ be a reduction function that satisfies the rules expressed in rules* **CR{1abc,2ab,3abc}**. *Then $G_\omega \preceq G$ and every transition t semi-modular in G remains semi-modular in G_ω.*

This mainly states the conditions imposed ensure that the transformed state graph still *"implements"* the initial state graph. However, we still need to ensure that G_ω satisfies the CSC requirement. In general, concurrency reduction alone is usually insufficient to remove all CSC violations. Additional state signals must be added. This will be touched upon in the next section. Nonetheless, we can formulate constraints to impose CSC only in the context of concurrency reduction.

Requirement CR4. *Checking for* CSC *property.* Let $\mathbf{CSC}$ be the set of state pairs (u, v) such that there exists a CSC violation between u and v. For a state u and a subset of transitions $T' \subseteq T$, let $FO(u)_{|T'} = \{u' \in FO(u) \mid \exists t \in T' : u \xrightarrow{t} u'\}$. For a state u, and let $T_O(u) = \{t \in T_O \mid \exists u' \in FO(u) : u \xrightarrow{t} u'\}$. Also, let $FU = FO(u)_{|T_O(u) - T_O(v)}$ and let $FV = FO(v)_{|T_O(v) - T_O(u)}$. Then for any $(u, v) \in \mathbf{CSC}$ the auxiliary function $\mathbf{csc}(u, v)$ is defined as follows:

$$\mathbf{csc}(u, v) = \overline{\omega_u} \vee \overline{\omega_v} \vee \left(\prod_{u' \in FU} \overline{\omega_{u'}} \prod_{v' \in FV} \overline{\omega_{v'}} \right) \tag{9}$$

$\mathbf{csc}(u, v)$ expresses that either u or v is removed or both $FO(u)$ and $FO(v)$ are reduced so that the CSC violation between u and v disappears in G_ω. Let $\mathbf{USC}$ be the set of state pairs (u, v) such that there exists an USC violation between u and v. Then $T_O(u) = T_O(v)$. For a state u and a transition t, let u_t denote the state if any satisfying $u[t\rangle u_t$. Then for any $(u, v) \in \mathbf{USC} - \mathbf{CSC}$ the auxiliary function $\mathbf{usc}(u, v)$ is defined as follows:

$$\mathbf{usc}(u, v) = \prod_{t \in T_O(u)} (\omega_{u_t} \oplus \omega_{v_t} \Rightarrow \overline{\omega_u} \vee \overline{\omega_v}) \tag{10}$$

$\mathbf{usc}(u, v)$ expresses that the removal of some fanout states of u and v can introduce a new CSC violation between u and v. In that case either u or v must be removed. Then checking if G_ω still contains CSC violations, we simply need to check

$$\prod_{(u,v) \in \mathbf{CSC}} \mathbf{csc}(u, v) \prod_{(u,v) \in \mathbf{USC} - \mathbf{CSC}} \mathbf{usc}(u, v) \tag{11}$$

As we are formulating the problem as a Boolean satisfiability problem, including these constraints with the rest will guarantee that if a solution exists without the addition of new signals, then the solution is in the search space of the satisfiability solver.

Circuit	Specification			Implementation		
	primary		initial			
	in	out	states	I	II	III
hp-alloc-outbound	4	3	17	2	2	2
hp-mp-forward-pkt	3	4	20	0	0	0
hp-nak-pa	4	5	34	1	1	1
hp-ram-read-sbuf	5	5	36	1	0	0
hp-rcv-setup	3	2	14	0	0	0
hp-sbuf-read-ctl	3	3	15	1	1	1
hp-sbuf-ram-write	5	5	38	2	1	0
scsi-tsend-bm	5	4	41	1	1	1
scsi-trcv-bm	5	4	44	2	1	1
scsi-isend-bm	5	4	44	1	1	1
scsi-trcv-csm	5	4	38	1	1	1
scsi-isend-csm	5	4	38	1	1	1
pscsi-isend	4	3	40	2	2	2
pscsi-ircv	4	3	25	1	1	1
pscsi-tsend	4	3	37	2	2	2
pscsi-trcv	4	3	24	1	1	1
pscsi-tsend-bm	4	4	45	2	1	1
pscsi-trcv-bm	4	4	43	2	1	1
chu-ad-opt	3	3	16	0	0	0
vanbek-ad-opt	3	3	22	0	0	0
dme-fast	3	3	26	1	1	1
dme	3	3	18	1	1	1
dram-controller	7	6	101	0	0	0
adfast	3	3	44	2	0	0
combuf	2	1	18	3	1	1
count	4	2	29	1	1	1
duplicator	2	2	20	2	1	1
mmu	4	4	174	3	0	0
sout	4	1	16	0	0	0
master-read	6	7	8932	t.o.	1	t.o.

t.o.: timeout limit reached.

Table 1: Experimental results.

4.2 Combining with state assignment

In [8], the Boolean constraints for the generalized state assignment approach were given. To combine generalized state assignment with the concurrency reduction approach described in the previous section, the Boolean constraints can be *combined* and solved simultaneously using any Boolean satisfiability solver. Due to space limitation, we refer the interested reader to [6] for a more extensive discussion.

5 Experimental Results

All synthesis procedures described in this paper have been fully implemented in the C language. To verify these automated procedures, we have thoroughly tested them on a set of 30 realistic design examples. The results are reported in Table 1. The largest example in terms of the size of the initial state graph is *master-read*. The initial state graph for this example has nearly 9000 states. The large initial state graph is due to the high-degree of concurrency in the initial specification.

All the benchmarks reported were specified initially either as a signal transition graph specification or as a burst-mode (multiple-input-change) asynchronous finite state machine specification. They were automatically translated to equivalent state graph specifications for synthesis. The number of input signals, output signals, and initial states for each example are indicated under the columns labeled “in”, “out”, and “initial states”, respectively.

For each benchmark, we try to satisfy the CSC requirement with three ways. The first way is to apply the generalized state assignment framework method in [8], which does not perform concurrency reduction. The number of extra state signals required using this method for each benchmark is indicated under the column labeled “I”. The second way is to apply our framework with the “heuristic iteration strategy” described in Section 3. The results for this method are indicated under the column labeled “II”. The third way is to apply our framework with the “global composition strategy” described in Section 4. The results for this method are indicated under the column labeled “III”.

In both concurrency reduction strategies, input concurrency is always maintained. This means that extra state signals may still be required to disambiguate some states. As expected, in all cases, the solutions obtained with concurrency reduction methods require often less state signals, but always no more, than state assignment without concurrency reduction. This is due to the fact that a larger solution space is explored. For example, in the benchmark “mmu”, the state assignment only approach required 3 signals whereas the concurrency reduction methods did not require any extra signals. In the benchmark “master-read”, which has nearly 9000 states, a solution with only one extra state signal was found when concurrency reduction was considered. The concurrency reduction process produced a final result with only 41 states.

6 Conclusions

The principle aim of this paper was to develop a formal framework for synthesizing asynchronous control specifications purely at the state graph level that combines both concurrency reduction and introduction of new state signals in the search space. Working purely at the state graph level has two important advantages. First, it is possible to synthesize a wide variety of high-level models, thus making the synthesis framework very general. Second, it is possible perform complex transformations at the state graph level that are difficult or impossible to achieve in other frameworks. In this paper, we have presented a general framework that can explore this search space in a global way. We have shown by means of examples that faster and smaller are possible when reduction of concurrency is explored. This is partly because often less new state signals are required and the resulting logic is typically more highly unspecified, thus leaving more room for optimizations at the logic level. Thus, contrary to the common beliefs, maintaining concurrency does not always lead to higher performance realizations. By combining both concurrency reduction and insertion of new state signals, a significantly larger solution space is searched. Within the proposed theoretical framework, we have thus far developed search procedures that are targeted towards minimizing the number of introduced state signals. Currently, we are investigating alternative search procedures within the framework that are targeted towards performance.

References

[1] P. A. Beerel and T. H.-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proc. of the ICCAD*, Nov. 1992.

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

[3] D. L. Dill. *Trace Theory for Automatic Heirarchical Verification of Speed-Independent Circuits.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.

[4] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[5] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proc. of the DAC*, June 1991.

[6] B. Lin, Ch. Ykman-Couvreur, and P. Vanbekbergen, A general state graph transformation framework for asynchronous synthesis. In *IMEC Technical Report*, 1994.

[7] P. Vanbekbergen, F. Catthoor, G. Goossens, and H. De Man. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. In *Proceedings of the International Conference on Computer-Aided Design*, pages 184–187, November 1990.

[8] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man. A generalized state assignment theory for transformations on signal transition graphs. In *Proc. of the ICCAD*, pp. 112–117, Nov. 1992.

[9] P. Vanbekbergen, Ch. Ykman-Couvreur, B. Lin, and H. de Man. Generalizing signal transition graphs for modeling mixed asynchronous/synchronous and arbitration behaviors. In *Presented at the International Workshop on Logic Synthesis*, 1993.

[10] Ch. Ykman-Couvreur, P. Vanbekbergen, and B. Lin. Concurrency reduction transformations on state graphs for asynchronous circuit synthesis. In *International Workshop on Logic Synthesis*, 1993.