OPTIMIST: State Minimization for Optimal 2-Level Logic Implementation

Robert M. Fuhrer   Steven M. Nowick*
Department of Computer Science
Columbia University
New York, NY 10027

Abstract

We present a novel method for state minimization of incompletely-specified finite state machines. Where classic methods simply minimize the number of states, ours directly addresses the implementation's logic complexity, and produces an exactly optimal implementation under input encoding. The method incorporates optimal "state mapping", i.e., the process of reducing the symbolic next-state relation which results from state splitting to an optimal conforming symbolic function. Further, it offers a number of convenient sites for applying heuristics to reduce time and space complexity, and is amenable to implementation based on implicit representations. Although our method currently makes use of an input encoding model, we believe it can be extended smoothly to encompass output encoding as well.

1 Introduction

State minimization is the problem of finding a machine realizing the input/output behaviour of a given FSM, with fewer internal states [11, 17, 12]. This is an important step in sequential synthesis: implementing unminimized FSM's often leads to considerably larger and/or slower implementations. However, it is well known that the classic formulation for state minimization expresses a *heuristic* – reducing the number of states only *tends* to decrease logic complexity. Early on, Hartmanis observed [9] that this heuristic sometimes fails; realizations having more states may be simpler to implement. Moreover, there may be many minimum-state realizations of a given FSM, and their logic complexity can vary significantly [18, 16]. Hence, simply targeting any minimum-state solution is insufficient.

The major contribution of this paper is a state minimization method which, in contrast to existing ones, directly targets logic complexity. In particular, we define and solve the *optimal state minimization problem*, that of finding for a given FSM a realization having minimum 2-level logic complexity over all realizations.

Classic sequential synthesis comprises several steps: state minimization, state encoding, 2-level logic minimization, multi-level optimization and technology mapping. Each step has traditionally been treated as an isolated problem, which limits early steps most severely. In [7], a key insight into optimal state encoding was presented: *symbolic* logic minimization can be performed *concurrently* with state encoding. More recent methods for optimal encoding have been developed [20, 8] based on the same insight, but yielding even better results.

We borrow this insight, and take it one step further: symbolic logic minimization is performed concurrently with *both* state minimization and state encoding. Our method is cast as a unique form of generalized prime implicant minimization [8]. Specifically, symbolic prime implicants are generated, and a binate covering problem is formed and solved, yielding a reduced machine and logic cover.

Our paper offers a novel theoretical framework for formulating and solving the optimal state minimization problem. We demonstrate that our method identifies optimal solutions which are inaccessible to existing tools, due to their focus on minimum cardinality state covers. In addition, we provide initial results of a CAD tool implementation.

The structure of the paper is as follows. Section 2, provides background on state minimization, state assignment and related work. Section 3 then gives a general overview of our method, with a focus on the major issues. Sections 4, 5 and 6, provide greater detail on the three major components of our method: symbolic prime generation, binate constraint generation, and symbolic instantiation, respectively. Two examples in Section 7 demonstrate the procedure and show results unattainable by existing methods. Finally, Section 8 provides some experimental results, and Section 9 presents conclusions and future work.

2 Background and Related Work

A *completely-specified* Finite State Machine (FSM) $\mathcal{M}$ is defined [16, 6] by the tuple $\langle \mathcal{I}, \mathcal{O}, \mathcal{S}, \mathcal{S}_0, \mathcal{T}, \mathcal{F} \rangle$, where $\mathcal{I}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, $\mathcal{S}$ is the set of states, $\mathcal{S}_0 \subset \mathcal{S}$ is the (set of) initial state(s), $\mathcal{T} = \mathcal{T}(i, s) \in \mathcal{S}, i \in \mathcal{I}, s \in \mathcal{S}$ is the transition function, and $\mathcal{F} = \mathcal{F}(i, s) \in \mathcal{O}, i \in \mathcal{I}, s \in \mathcal{S}$ is the output function.

An *incompletely-specified* FSM (ISFSM) is defined similarly, except that $\mathcal{T} \subset \mathcal{I} \times \mathcal{S} \times \mathcal{S}$ and $\mathcal{F} \subset \mathcal{I} \times \mathcal{S} \times \mathcal{O}$ are relations, rather than functions.[1]

*This research was funded in part by NSF CAREER Award MIP-9501880 and by an Alfred P. Sloan Research Fellowship.

[1]Where clear, a functional notation will be used for ISFSM's.

For the unreduced machine, we restrict attention to the common subclass of ISFSM's where, $\forall i \in \mathcal{I}, \forall s \in \mathcal{S}$, the next-state $\mathcal{T}(i,s)$ is either a singleton state or else is completely unspecified (denoted $\mathcal{T}(i,s) = \mathcal{S}$). Likewise, it is assumed that output $\mathcal{O}(i,s)$ is either a single value or unspecified.

2.1 State Minimization

We now review basic definitions given in [11].

A pair of states (s_a, s_b) *implies* states (s_c, s_d) *under input* $\mathcal{I}_k$ iff $\{\mathcal{T}(\mathcal{I}_k, s_a), \mathcal{T}(\mathcal{I}_k, s_b)\} = \{s_c, s_d\}$.

A pair of states is *output-compatible* iff corresponding output values agree wherever both are specified.

A pair of states is *compatible* iff they are output-compatible and imply no incompatible pair of states.

A set of states is a *compatible* iff it consists of pairwise-compatible states.

Compatible c_a *implies* compatible c_b *under input* I_k iff $\{\mathcal{T}(\mathcal{I}_k, s), \forall s \in c_a\} = c_b$.

Compatible c is a *maximal compatible* iff it is a proper subset of no other compatible.

The *implied set* $P(c)$ of a compatible c is the set of compatibles implied by c over all inputs, excluding singleton states, subsets of c, and proper subsets of compatibles in $P(c)$.

Compatible c_a *excludes* c_b iff $c_b \subset c_a$ and $P(c_a) \subseteq P(c_b)$.

Compatible c is a *prime compatible* iff it is excluded by no other compatible.

A set C of compatibles is *closed* iff every element of the implied set of each compatible $c \in C$ is contained by some compatible in C.

A set C of compatibles is a *cover* for $\mathcal{M}$ iff for every state $s \in \mathcal{S}$ there exists a compatible $c \in C$ such that $s \in c$.

The classic state minimization problem can now be defined as finding a minimum cardinality closed cover of prime compatibles of $\mathcal{M}$. In the sequel, we designate the resulting reduced machine as $\mathcal{M}'$.

2.2 The State Mapping Problem

Given an incompletely-specified FSM, a state reduction often defines a *set* of compatible realizations [17, 11]. In this case, the next-state behaviour of the resulting ISFSM forms a *relation*, so that different next-state bindings are possible. This flexibility gives rise to the *state-mapping problem* [15], in which the symbolic relation is reduced to a conforming symbolic function. The function is obtained by choosing a specific next-state wherever a choice exists. This choice has a direct impact on logic quality, and is a pivotal issue for optimal state minimization.

$\mathcal{M}$	0	1
s_0	s_0,0	s_2,0
s_1	s_1,0	s_1,-
s_2	s_1,-	s_0,1

$\mathcal{M}'$	0	1
$s'_0 = \{s_0, s_1\}$	s'_0,0	s'_1,0
$s'_1 = \{s_1, s_2\}$	$\{s_1\}$,0	s'_0,1

Figure 1: State table before and after minimization

The state mapping problem is illustrated in Figure 1, taken from [15]. A choice of next-state in $\mathcal{M}'$ at total state $\langle\, \texttt{0}\ s'_1 \,\rangle$ exists; it can be assigned to either s'_0 or s'_1. The best cover achievable[2] when state mapping s_1 to s'_1 in that total state has 4 terms. In contrast, state mapping to s'_0 yields a 3-term cover.

2.3 Optimal State Assignment as Input Encoding

As our strategy for optimal state minimization is the *concurrent* reduction and assignment of states, we now review basic concepts in optimal state assignment.

The problem of *optimal state assignment* is that of finding an assignment of N-bit binary values to the internal states of an (IS)FSM so as to minimize the logic complexity of the resulting realization. One useful approximation to this problem is known as *input encoding.* As we use an input encoding formulation in our approach to optimal state minimization, we briefly give an overview. For further details, see [7], or, for recent extensions for hazard-free encoding of asynchronous machines, see [10].

Input encoding, like several approaches to optimal encoding, has two steps:

1. *Symbolic 2-level logic minimization*
2. *Constrained encoding*

In input encoding, symbolic logic minimization is performed by temporarily "1-hot encoding" the states. The present state is then treated as a multi-valued input [19], while each next state is treated as a *disjoint* binary output function. Multi-valued minimization [19] produces the desired 2-level symbolic cover. The structure of this symbolic cover reflects that of the target realization.

Using this cover, encoding constraints are generated. The constraints are then solved, yielding an encoding with which the symbolic cover can be "instantiated" to produce a correct binary implementation. The resulting 2-level realization has *identical* cardinality to the original symbolic cover.[3]

2.4 Previous Work

The topic of state minimization has been researched extensively over several decades. Hartmanis observed in [9] that the minimum cardinality solution does not always yield the best implementation. The problem was later approximated as a search for a closed cover *of minimum cardinality* and solved exactly in [11]. In [13], the relationship of state reduction to implementation complexity was explored via an elegant theoretical framework, which unfortunately did not provide a solution to the problem.

Several recent methods have been focused on producing minimum cardinality covers. Efficient algorithms have been produced for solving the problem exactly ([12], [14]), and heuristics ([12]) have been developed for inexact solutions. These two fronts have seen considerable progress.

Only a few recent attempts have been made to address the more general problem of *optimal* state minimization. In STAMINA [12], some attention is paid

[2]assuming an input-encoded implementation

[3]Modulo the well-known "all-0 code" issue; see [6].

to implementation complexity, but no attempt at direct nor exact solutions was made. A somewhat more direct approach was taken by Avedillo et al. [1], but results were less than encouraging (they were not even compared with a state reduction tool, but rather with NOVA, a state assignment tool, which did better on already-minimized FSM's), and no theoretical results are given. Finally, Calazans [3] offers a framework within which both optimal encoding and state minimization can be expressed. In it, state reduction is modeled as the assignment of two or more states to the same code. This insight constitutes one of the cornerstones of our method. Unfortunately, the only solution method given is a simple, greedy method which fares relatively poorly. Worse, it is not evident how to express a high-quality solution method within that formulation.

Two recently developed CAD optimization techniques are also relevant to our work, though created for other purposes. First, Devadas and Newton [8] address the problem of exact state encoding via *GPI minimization.* In this method, *generalized* symbolic primes (*GPI's*) are formed, and a constrained binate covering problem is then solved which both selects a set of GPI's and produces a compatible state encoding. The basic flow – symbolic prime generation followed by constrained covering – is used in our present method, but with considerably different symbolic primes and constraints.

Second, Lin and Somenzi [15] introduce a technique for the exact minimization of symbolic relations. Of particular interest is its application to exact state encoding, incorporating an elegant method for state mapping. The symbolic relation they minimize, however, is the *result* of state minimization; state minimization itself is not addressed.

3 Optimal State Minimization: Overview

This section provides a general overview of our optimal state minimization method.

Optimal state minimization is defined as finding, for a machine $\mathcal{M}$, a reduced machine $\mathcal{M}'$ with compatible behaviour and minimum logic complexity. Our method is cast as a form of symbolic GPI (generalized prime implicant) minimization which encapsulates both state encoding *and* state minimization. The method has 5 steps:

1. Generate state compatibles
2. Generate symbolic primes
3. Generate binate constraints
4. Solve constraints
5. Instantiate symbolic cover

First, state compatibles are formed by any of various standard methods (e.g., STAMINA [12]). Next, a novel form of symbolic prime called *RGPI's*, based on GPI's [8], is generated. A set of binate constraints which identify the valid realizations is then formed. These constraints are solved so as to minimize logic cardinality using a binate solver (e.g., Scherzo [5]). The solution is a set of selected compatibles and a set of selected RGPI's. These are trivially combined during cover instantiation, to produce the reduced machine $\mathcal{M}'$ and its symbolic realization. From there, input encoding constraints can be immediately generated and solved to produce an optimal encoding.

We now give an overview of the 3 steps unique to our method: symbolic prime generation, binate constraint generation, and cover instantiation. Further details are provided in Sections 4 through 6.

3.1 Symbolic Primes

A *symbolic product* is a product of literals over an mvi domain [6]. Each value taken by a symbolic output is called a *symbolic part.* A *symbolic implicant* is a symbolic product which satisfies the following two properties: (i) it contains no OFF-set minterm of any *binary output* to which it contributes; (ii) for each *symbolic output* to which it contributes, it asserts all symbolic parts specified in the minterms which it contains. A symbolic implicant of an FSM is expressed as a 4-tuple $p : \langle\, \texttt{in ps ns out}\, \rangle$, denoting the input, present-state, next-state and output fields.

Our procedure forms a set of symbolic implicants on the unreduced machine, which can be used to cover portions of various reduced machines, after a suitable transformation. These implicants are maximal over $\mathcal{M}$ in an intuitive sense. Specifically, we define a novel type of symbolic prime implicant, called a *restricted generalized (prime) implicant*, or *RG(P)I*, a variant of a GPI, formed on the unreduced machine.

GPI's were introduced in [8] as a kind of symbolic prime implicant, each "tagged with" (i.e., having a field comprising) a *set* of next-states. The tag contains *all* next-states which are specified in all total states the GPI contains. For example, in Figure 1, the symbolic implicant $g_a : \langle - \ s_0 \ s_0, s_2 \ \texttt{0} \rangle$ is a GPI, where $\texttt{ns}= \{s_0, s_2\}$.

We now formally define RGI's, and an easily-calculated subset of RGPI's called *RGPI seeds.*

Definition 3.1 *An* **RGI** *is a symbolic implicant whose next-state field consists of compatible states.*

Intuitively, the selection of an RGI corresponds to a state-mapping choice. Note that the next-state field of an RGI (if non-empty) is a set of states, i.e. a *compatible*, in the unreduced machine. This compatible, if selected, will appear as a *single state* in the reduced machine. Therefore, the RGI will be bound to (at most) one symbolic next-state in the reduced machine, as required by the input encoding formulation.

Definition 3.2 *An RGI p_1* **contains** *RGI p_2 iff each field of p_1 contains the corresponding field of p_2.*

Definition 3.3 *An* **RGPI seed** *is an RGI which is contained* only *by RGI's with unequal next-state fields.*

Example: Figure 1 has the following RGPI seeds:

$p_0 : \langle \texttt{0} \ s_0 \ s_0 \ \texttt{0} \rangle$ $\quad p_1 : \langle \texttt{0} \ s_0, s_1, s_2 \ s_0, s_1 \ \texttt{0} \rangle$
$p_2 : \langle \texttt{0} \ s_1, s_2 \ s_1 \ \texttt{0} \rangle$ $\quad p_3 : \langle \texttt{0} \ s_2 \ s_1 \ \texttt{1} \rangle$
$p_4 : \langle \texttt{1} \ s_0 \ s_2 \ \texttt{0} \rangle$ $\quad p_5 : \langle \texttt{1} \ s_1 \ s_1 \ \texttt{1} \rangle$
$p_6 : \langle - \ s_1 \ s_1 \ \texttt{0} \rangle$ $\quad p_7 : \langle \texttt{1} \ s_0, s_1 \ s_1, s_2 \ \texttt{0} \rangle$

$p_8 : \langle \texttt{1}\ \ s_1,s_2\ \ s_0,s_1\ \ \texttt{1} \rangle$ $\quad$ $p_9 : \langle \texttt{0}\ \ s_1,s_2\ \ s_1,s_2\ \ \texttt{0} \rangle$
$p_{10} : \langle -\ \ s_1,s_2\ \ s_0,s_1\ \ \texttt{0} \rangle$ $\quad$ $p_{11} : \langle -\ \ s_2\ \ s_0,s_1\ \ \texttt{1} \rangle$

Note that $\{s_0, s_1\}$ is a compatible, but $\{s_0, s_2\}$ is not. Therefore, p_1 is an RGI, but GPI $g_a : \langle -\ \ s_0\ \ s_0,s_2\ \ \texttt{0} \rangle$ is not. Furthermore, $p_b : \langle \texttt{0}\ \ s_0,s_1\ \ s_0,s_1\ \ \texttt{0} \rangle$ is an RGI, but is not an RGPI, since p_1 contains p_b and has the same next-state field. On the other hand, although p_1 contains p_0, its next-state field is different, and so p_0 is a distinct RGPI. $\Box$

We generate RGPI seeds by a slightly modified version of a GPI generation algorithm [8], which will be described in Section 4.

While RGPI seeds are the basic covering objects which will be used, it will be shown in Section 4 that a more general class, RGPI's, is in fact needed.[4]

Definition 3.4 *An* **RGPI** *is an RGI which is contained* only *by RGI's with unequal input or next-state fields.*

The class of RGPI's includes RGPI seeds, as well as smaller RGPI's which result from reducing seeds in the input dimension, to allow finer-grained covering.

3.2 Constraint Generation

Once RGPI's are generated, constraints are formulated to insure a valid and optimal implementation. The solution is a set of selected compatibles and RGPI's. There are 3 main objectives. First, reduced machine $\mathcal{M}'$ must be a realization of $\mathcal{M}$. Second, the selected set of RGPI's must constitute a cover for $\mathcal{M}'$. Third, the resulting cover must have minimum cardinality. We now outline the constraints; details are provided in Section 5.

The first objective is ensured by two sets of constraints. Each corresponds to a classic state minimization constraint: that the state compatibles form a cover, and that the cover be closed. State covering constraints are precisely as described in [11].

The second objective, covering $\mathcal{M}'$, is met by a novel set of constraints. Each selected compatible identifies a unique state in the reduced machine. Our constraints ensure that each ON-set minterm of this reduced state is covered by some (instantiated) RGPI.

The third objective, minimum logic cardinality, is met by the binate solver, which finds a minimum-cost solution. To make cost a straightforward calculation, we introduce one extra column variable per RGPI. Only these column variables have non-zero cost.

3.3 Symbolic Instantiation

Symbolic instantiation is the process by which selected RGPI's are transformed one-for-one into a symbolic realization of $\mathcal{M}'$. To describe the process, we consider each of the 4 fields of an RGPI.

Formally, for $p = \langle I,\ PS,\ NS,\ O\rangle$ we define

$$p' = \text{Instantiate}(p) = \langle I,\ PS',\ NS',\ O\rangle$$

[4]Note that RGPI seeds are GPI's, while RGPI's in general are not.

The *input* and *output* fields of p are unchanged by instantiation.

The *next-state* field of p is a *compatible* of $\mathcal{M}$. Note that this compatible corresponds to a *single row* in the reduced machine. Hence, an RGPI's next-state field identifies a unique state of $\mathcal{M}'$, and is mapped trivially:

$$NS' = \text{ the unique state of } \mathcal{M}' \text{ corresponding to } NS$$

The role of the RGPI in the reduced machine is thus to contribute to (at most) a *single* symbolic next-state. This role reflects our use of an input encoding formulation, where each next-state in the reduced machine is treated as a distinct function. Therefore, each RGPI embodies a *uniform state mapping* over some cube of $\mathcal{M}'$ to a *specific* reduced state of $\mathcal{M}'$.

Finally, the *present-state* field of an RGPI contains one or more symbolic states. To a first approximation, an RGPI will be mapped so as to cover *all selected* compatibles c which contain the present state field of that RGPI. That is, the RGPI is regarded as covering the class of compatibles which are contained within its present state field.

Example: RGPI $p_8 : \langle \texttt{1}\ \ s_1,s_2\ \ s_0,s_1\ \ \texttt{1} \rangle$ in Figure 1 contributes to (compatible) next-states $\{s_0, s_1\}$ of $\mathcal{M}$, and has present states $\{s_1, s_2\}$. Therefore, in the reduced table $\mathcal{M}'$, p_8 maps to the product $p_8' : \langle \texttt{1}\ \ s_1'\ \ s_0'\ \ \texttt{1} \rangle$. The resulting present state field contains $s_1' = \{s_1, s_2\}$; the next-state field consists of the single reduced next-state s_0' (which corresponds to the original compatible set $\{s_0, s_1\}$). $\Box$

As indicated, for the present state, we can include in PS' all selected compatibles contained by PS. This scheme works in some cases, but fails to capture the full flexibility of state mapping in others. In Section 6, we define the precise mapping for PS which circumvents this problem.

Example: One solution to the constrained covering problem, for Figure 1, consists of (i) compatibles $s_0' \equiv \{s_0, s_1\}$ and $s_1' \equiv \{s_1, s_2\}$, and (ii) RGPI's $\{p_1, p_7, p_8\}$ shown below.

$p_1 : \langle \texttt{0}\ \ s_0,s_1,s_2\ \ s_0,s_1\ \ \texttt{0} \rangle$	$p_1' : \langle \texttt{0}\ \ s_0',s_1'\ \ s_0'\ \ \texttt{0} \rangle$
$p_7 : \langle \texttt{1}\ \ s_0,s_1\ \ s_1,s_2\ \ \texttt{0} \rangle$	$p_7' : \langle \texttt{1}\ \ s_0'\ \ s_1'\ \ \texttt{0} \rangle$
$p_8 : \langle \texttt{1}\ \ s_1,s_2\ \ s_0,s_1\ \ \texttt{1} \rangle$	$p_8' : \langle \texttt{1}\ \ s_1'\ \ s_0'\ \ \texttt{1} \rangle$

The selected RGPI's, $\{p_1, p_7, p_8\}$, can be instantiated as symbolic implicants $\{p_1', p_7', p_8'\}$ of reduced machine $\mathcal{M}'$. It is easy to verify that the result is a cover for $\mathcal{M}'$. Observe that the input and output fields are unchanged; only the present-state and next-state fields are transformed. Further, each mapped implicant contributes to at most 1 state. Note that the state mapping choice at $\langle \texttt{0}\ s_1' \rangle$ is resolved to s_0' by the binding effected by p_1'. $\Box$

4 Symbolic Primes: RGPI's

The structure of this section is as follows. First, we present an RGPI seed generation algorithm. Next, we highlight the problem arising from restricting solutions to RGPI seeds. Finally, we describe the solution to this problem – use of a larger set of symbolic primes, the complete set of RGPI's.

4.1 Generating RGPI Seeds

RGPI seeds are generated by a modified version of an existing "k-cube" algorithm [8]. First, "0-cubes" are generated, which essentially record the next-state and output in each total state. Then, an iterative "merge-and-dominate" step is performed. The algorithm produces seeds with "tight-fitting" next-state fields, which exactly equal the set of next-states specified within each cube. Finally, a simple post-processing step (not shown) expands the next-state fields to larger compatibles, yielding the remaining seeds.

```
**merge**$(s_a, s_b)$ :=
        $s_a \cup s_b$ if compatible$(s_a, s_b)$ and neither $= \phi$,
        $s_a$        if $s_b =$ DC,
        $s_b$        if $s_a =$ DC,
        $\phi$        otherwise
**generate-0-cubes**() {
  cubes$_0 := \phi$;
  **foreach** total state $\tau = \langle IN\ PS \rangle$ **do**
    OUT := $\{o_i \mid$ output $i \neq 0$ in $\tau\ \}$;
    NS := $\{s\}$ iff $\mathcal{T}(IN, PS) = s$, else DC;
    cubes$_0$ := cubes$_0 \cup \{\ \langle IN\ PS\ OUT\ NS \rangle\ \}$;   }
**generate-k-cubes**() {
  **generate-0-cubes**();
  **for** $k := 1$ **to** kMax **do**
    cubes$_k := \phi$;
    **for** each pair $p_i, p_j$ in cubes$_{k-1}$ **do**
      **if** distance$(p_i, p_j) = 1$ **then**
        $\mathrm{IN}' := \mathrm{IN}_i \cup \mathrm{IN}_j$;
        $\mathrm{PS}' := \mathrm{PS}_i \cup \mathrm{PS}_j$;
        $\mathrm{NS}' :=$ merge$(\mathrm{NS}_i, \mathrm{NS}_j)$;
        $\mathrm{OUT}' := \mathrm{OUT}_i \cap \mathrm{OUT}_j$;
        **if** $(\neg$empty$(\mathrm{NS}') \wedge \neg$empty$(\mathrm{OUT}'))$ **then**
          cubes$_k$ := cubes$_k \cup \{\langle \mathrm{IN}', \mathrm{PS}', \mathrm{NS}', \mathrm{OUT}' \rangle\}$;
          **if** $(\mathrm{OUT}' = \mathrm{OUT}_i) \wedge (\mathrm{NS}' = \mathrm{NS}_i)$
            mark $p_i$ dominated;
          **if** $(\mathrm{OUT}' = \mathrm{OUT}_j) \wedge (\mathrm{NS}' = \mathrm{NS}_j)$
            mark $p_j$ dominated;   }
```

4.2 Non-Seed RGPI's

It is not always possible to construct an optimal solution using only RGPI seeds. The reason is the incompatibility of RGPI's which intersect and implement next-state, but disagree. Their incompatibility results from their commitment to bind the next-state entries of contained total states to *conflicting* states. Lacking finer-grained cubes, this interference results in cases where only sub-optimal solutions are produced, or where no cover exists.

Example: The following table, when reduced using compatibles $\{s_1, s_2\}$ and $\{s_2, s_3\}$, cannot be covered by RGPI seeds alone:

$\mathcal{M}$	00	01	11	10
s_0	s_1,-	s_2,-	s_3,-	s_4,1
s_1	s_1,1	s_1,1	s_4,-	s_4,-
s_2	s_2,1	s_2,-	s_4,0	s_4,0
s_3	s_2,-	s_3,0	s_4,0	s_4,0
s_4	s_4,0	s_4,1	s_4,1	s_4,1

Consider the RGPI seeds $p_1 : \langle\ \texttt{0-}\ \ s_0, s_1, s_2\ \ s_1, s_2\ \ \texttt{1}\ \rangle$ and $p_2 : \langle\ \texttt{-1}\ \ s_0\ \ s_2, s_3\ \ \texttt{1}\ \rangle$. These seeds are incompatible: they intersect in total state $\langle\ \texttt{01}\ s_0\ \rangle$, but have different next-state fields ($\{s_1, s_2\}$ vs. $\{s_2, s_3\}$), representing conflicting state mappings. Yet, p_1 and p_2 are both essential for covering minterms at $\langle\ \texttt{00}\ s_0\ \rangle$ and $\langle\ \texttt{11}\ s_0\ \rangle$, respectively. Hence no solution exists, using only RGPI seeds. □

Clearly, RGPI seeds are not fine-grained enough to express the full flexibility of state mapping. In some cases, no solution consisting solely of RGPI seeds exists; in others, no optimum solution exists. In general, an optimum solution requires a combination of RGPI seeds and finer-grained symbolic cubes. To see the kind of cubes that are needed, consider following reduction of the above machine.

Example:

$\mathcal{M}'$	00	01	11	10
$s'_0 = \{s_0\}$	s'_1,-	$\{s_2\}$,-	s'_2,-	s'_3,1
$s'_1 = \{s_1, s_2\}$	s'_1,1	s'_1,1	s'_3,0	s'_3,0
$s'_2 = \{s_2, s_3\}$	$\{s_2\}$,1	s'_2,0	s'_3,0	s'_3,0
$s'_3 = \{s_4\}$	s'_3,0	s'_3,1	s'_3,1	s'_3,1

There are two state mapping choices in $\langle\ \texttt{01}\ s'_0\ \rangle$: s'_1 and s'_2. If we choose mapping s'_1, we require an implicant which covers the isolated s'_2 minterm at $\langle\ \texttt{11}\ s'_0\ \rangle$; however, no RGPI seed maps onto such an implicant. If we choose s'_2, two distinct implicants are required to cover the minterms s'_1 at $\langle\ \texttt{00}\ s'_0\ \rangle$ and $\langle\ \texttt{01}\ s'_1\ \rangle$; no RGPI seed maps onto either implicant. □

We need smaller RGPI's to gain finer control over state-mapping. Specifically, we require a set of implicants with smaller input and/or present-state fields.

Example: An optimal cover can be constructed, if the set of RGPI seeds is augmented with the following RGPI's: $p_{1a} : \langle\ \texttt{00}\ \ s_0, s_1, s_2\ \ s_1, s_2\ \ \texttt{1}\ \rangle$, $p_{1b} : \langle\ \texttt{0-}\ \ s_1, s_2\ \ s_1, s_2\ \ \texttt{1}\ \rangle$, and $p_{2a} : \langle\ \texttt{11}\ \ s_0\ \ s_2, s_3\ \ \texttt{1}\ \rangle$. □

We now show how such non-seed cubes as p_{1a}, p_{1b}, and p_{2a} above can be derived from RGPI seeds by restricting their input and/or present-state fields. Two distinct approaches are used. For cubes restricted in the present-state field, we introduce decision variables into constrained covering which determine the selected subset of reduced states. For cubes restricted in the input field, we simply refine RGPI seeds in the input dimension. In the following sections, we describe both approaches.

4.3 Present State Field

Non-seed RGPI's which span fewer reduced states are obtained, by associating a set of Boolean decision variables $\{\gamma_{p,c}\}$ with each RGPI p. Each $\gamma_{p,c}$ is set to true iff RGPI p is to be mapped so as to span the reduced state corresponding to c. Thus, the present-state mapping of p is incorporated into the constrained covering step. The $\gamma_{p,c}$ assignments chosen during covering are later used in symbolic instantiation.

$\gamma_{p,c}$ is well-defined only for compatibles c which are contained in $PS(p)$, since including any other c would not result in an implicant of $\mathcal{M}'$. Therefore, we define $\Gamma_p = \{c \mid c \subseteq PS(p), c \in C\}$, and reserve a $\gamma_{p,c}$ for each c in Γ_p.

4.4 Input Field

Non-seed RGPI's which are smaller in the input dimension are obtained, by simply "refining" RGPI seeds, reducing them in all possible inputs, and adding the resulting sub-cubes to the RGPI set.

5 Constraint Generation

Once RGPI generation is completed, constraints must be generated. There are five distinct sets of constraints, each addressing a specific requirement on valid realizations. Two (sets 1 and 3 below) correspond directly to classic state minimization constraints, which ensure a closed cover of state compatibles. One set (set 2) ensures that the ON-set of the reduced machine is covered by the selected RGPI's. An additional set (set 4) ensures that the machine is state-mapped consistently by the selected RGPI's. The final set of constraints (set 5) provides the solver with a trivial means of determining the cover's cost.

The covering problem can be expressed as a constraint matrix, where each row is a constraint and each column is a decision variable. The covering matrix contains three kinds of columns: state compatibles, RGPI's, and "γ" variables, which modulate RGPI instantiation. Note that many constraints are binate, as in state reduction [11] and relation minimization [15, 2].

5.1 Constraints

The columns in the covering matrix are:

c_i	–	include compatible c_i in state cover
p_i	–	include RGPI p_i in symbolic cover
γ_{p_i,c_i}	–	make RGPI p_i span c_i in $\mathcal{M}'$

The constraint clauses, which correspond to covering matrix rows, are shown below, grouped in five sections according to purpose. Each section consists of a set of similar clauses, all of which must be satisfied. The final Boolean expression is a conjunction of all rows/clauses in all five sections.

1. *State Covering*
   Each unreduced state must be covered by some state compatible (reduced state).
   $\forall$ states s of $\mathcal{M}$
   $$c_{i_1} + c_{i_2} + \cdots + c_{i_N}$$
   where $s \in c_{i_k}$.
2. *Compatible Selection $\Rightarrow$ RGPI Selection*
   ("Functional Covering")
   Each ON-set minterm of each output and next-state of $\mathcal{M}'$ must be covered. Since selecting a compatible corresponds to adding a row in the reduced table, this constraint ensures that every ON-set minterm in the reduced row is covered.

   $\forall$ compatibles c_k
   $$\prod_{\forall m' \in c_k} \{\overline{c_k} + \gamma_{p_1,c_k} + \gamma_{p_2,c_k} + \cdots + \gamma_{p_N,c_k}\}$$
   Here, m' refers to the ON-set minterms lying in c_k, corresponding to either an output or the next-state in the *reduced* machine $\mathcal{M}'$. Each clause ensures that some RGPI p_i will be mapped over c_k (i.e. γ_{p_i,c_k} will be true), and hence will cover m' in $\mathcal{M}'$.

   Although the minterms m' in row c_k of the reduced machine are not yet explicitly available, they can easily be derived. Let $i \in \mathcal{I}$ be any input value. The corresponding minterm m' is an ON-set minterm of a binary *output function* f_j in reduced machine $\mathcal{M}'$ iff, for some unreduced state $s \in c_k$, $f_j(s,i) = 1$. Minterm m' is considered an ON-set minterm of the *symbolic next-state* of the reduced machine $\mathcal{M}'$ iff for some unreduced state $s \in c_k$, $\mathcal{T}(s,i) = \tilde{s}$, for some singleton state $\tilde{s}$; that is, the next-state is specified in s. In this case, the next-state in m' will also be specified.

   For each ON-set minterm, m', those γ_{p_i,c_k} are included in the above constraints where (i) p_i contributes to the corresponding output or next-state, and (ii) also contains the minterm (i.e. intersects input column i).
3. *RGPI Selection $\Rightarrow$ Compatible Selection*
   If an RGPI which implements next-state is selected, the corresponding state compatible must also be selected.

   $\forall$ RGPI's $p_i \mid NS(p_i) \neq \phi$
   $$\overline{p_i} + c_k$$
   where compatible $c_k = NS(p_i)$.

   This constraint corresponds to a classic closure constraint [11]. An RGPI which implements next-state identifies a unique reduced state to which the next-state of all contained total states is uniformly state mapped. Hence, RGPI selection implies a commitment to select the compatible corresponding to its next-state field. Note that an RGPI implementing only outputs has no such constraint.
4. *Implicant Incompatibility*
   Two RGPI's are incompatible if they intersect in some selected compatible, but disagree on next-state.

   $\forall$ RGPI's $p_i, p_j \mid i \neq j,\ IN(p_i) \cap IN(p_j) \neq \phi,$
   $NS(p_i) \neq \phi, NS(p_j) \neq \phi, NS(p_i) \neq NS(p_j),$
   $\forall$ compatibles $c_k \subseteq PS(p_i) \cap PS(p_j)$
   and $\mathcal{T}(IN(p_i) \cap IN(p_j), c_k) \neq \mathcal{S}$
   $$\overline{c_k} + \overline{\gamma_{p_i,c_k}} + \overline{\gamma_{p_j,c_k}}$$
   If two RGPI's implement next-state and disagree, and intersect in some compatible c_k, they can both be mapped over c_k only if the next-state is unspecified ($\mathcal{T}(\mathcal{I}_a, s) = \mathcal{S}$) throughout the region of intersection. Otherwise, there is a conflict: a total state would be simultaneously mapped in two

ways. We must either not select c_k, or not map one of the RGPI's onto reduced state c_k.

5. *Implicant Cost*
   Mapping an RGPI over a reduced state implies selecting that RGPI.

   $\forall$ RGPI's $p_i, \forall$ compatibles $c_k \in \Gamma_{p_i}$

   $$\overline{\gamma_{p_i,c_k}} + p_i$$

   This is a bookkeeping device to make it easier for the solver to determine the solution cost (logic cover cardinality). The variable p_i is true when p_i is mapped over *at least one* reduced state.

5.2 Flow of Constraint Solution

The implications interconnecting the various constraints can be envisioned in a "flow" from, e.g., state covering (set 1) to functional covering (set 2) to state implication (set 3), and so on. To better illustrate the relationships, we offer the following solution scenario[5]:

1. We start by selecting a compatible c_i to cover some state, say, s_0. This satisfies the corresponding clause from set 1.
2. The selection of compatible c_i implies covering conditions on the ON-set minterms m' of $\mathcal{M}'$ lying in c_i (set 2). We choose one such minterm, and map an eligible RGPI p over c_i to cover it, by setting γ_{p,c_i} to true.
3. This now requires adding p to the cover (set 5).
4. Selecting p implies selecting the compatible c_j associated with p's next-state field, if any (set 3). If c_j was not previously selected, new covering constraints must be satisfied, and we recur on step 2.
5. If there are uncovered minterms in c_i, continue at step 2.
6. If there are uncovered states, continue at step 1.

At any point, the currently selected RGPI's may be incompatible (set 4) with RGPI's mentioned in covering clauses (set 2) for a specific compatible. If so, the latter RGPI's cannot be mapped over that compatible, and the corresponding γ's are removed from consideration in this sub-tree. There may then be no RGPI eligible to cover some minterm in step 2. If so, the sub-problem has no solution, and backtracking occurs.

6 Symbolic Instantiation

We now define symbolic instantiation precisely, taking into account the γ variables associated with each RGPI. These were introduced in section 4 to gain finer control over the shape of instantiated implicants.

Symbolic instantiation is defined with respect to a selected set of compatibles (and hence a reduced machine $\mathcal{M}'$), along with the set of γ variable assignments identifying specific reduced states to span. Specifically,

$$p' = \text{Instantiate}_{\gamma_p}(p)$$

[5] which can be regarded as a crude recursive-descent algorithm

where γ_p is a set of Boolean variables associated with p. Each member of γ_p is associated with a compatible contained in $PS(p)$. We thus let

$$p' = \text{Instantiate}_{\gamma_p}(p) = \langle I,\ PS',\ NS',\ O\rangle$$

with NS' as before, but, for the present state field:

$$PS' = \{ \text{ reduced states } s' \mid \begin{array}{l}\gamma_{p,c} = 1 \text{ and} \\ s' \text{ corresponds to } c\ \}\end{array}$$

7 Examples

We now present two examples. The first illustrates our procedure by giving partial results of each step for the simple example in Figure 1. We show both the optimal cover, corresponding to the optimal state mapping, and a sub-optimal cover, resulting from sub-optimal mapping. The second example demonstrates results not obtainable by other existing methods.

Example: (from Figure 1)

$\mathcal{M}$	0	1
s_0	s_0,0	s_2,0
s_1	s_1,0	s_1,-
s_2	s_1,-	s_0,1

Maximal compatibles:
$MC = \{\{s_0, s_1\}, \{s_1, s_2\}\}$.
Prime compatibles:
$PC = \{c_0, \ldots, c_4\} \equiv \{\{s_0\}, \{s_1\}, \{s_2\}\} \cup MC$.

The RGPI seeds were given in section 3.1. There are no non-seed RGPI's. A subset of the constraints follows, in POS form, and grouped by section:

State Covering (all shown):

$$(c_0 + c_3)(c_1 + c_3 + c_4)(c_2 + c_4)$$

Functional Covering (shown for c_0, c_3, c_4):

The first clause ensures that the minterm for s_0 in $\langle$ 0 $s_0 \rangle$ is covered. p_0 satisfies the clause when γ_{p_0,c_0} is set to true, which maps p_0 over c_0.

$$(\overline{c_0} + \gamma_{p_0,c_0} + \gamma_{p_1,c_0})(\overline{c_0} + \gamma_{p_4,c_0} + \gamma_{p_7,c_0})$$
$$(\overline{c_3} + \gamma_{p_1,c_3})(\overline{c_3} + \gamma_{p_7,c_1})$$
$$(\overline{c_4} + \gamma_{p_1,c_4} + \gamma_{p_2,c_4} + \gamma_{p_{10},c_4})(\overline{c_4} + \gamma_{p_8,c_4} + \gamma_{p_{10},c_4})$$
$$(\overline{c_4} + \gamma_{p_8,c_4}) \cdots$$

Compatible Implication (shown for $p_0, \ldots, p_3$):

$$(\overline{p_0} + c_0)(\overline{p_1} + c_3)(\overline{p_2} + c_1)(\overline{p_3} + c_1) \cdots$$

Implicant Incompatibility (shown for p_0, p_1):

$$(\overline{c_0} + \overline{\gamma_{p_0,c_0}} + \overline{\gamma_{p_1,c_0}})(\overline{c_1} + \overline{\gamma_{p_1,c_1}} + \overline{\gamma_{p_2,c_1}})$$
$$(\overline{c_2} + \overline{\gamma_{p_1,c_2}} + \overline{\gamma_{p_2,c_2}})(\overline{c_4} + \overline{\gamma_{p_1,c_4}} + \overline{\gamma_{p_2,c_4}})$$
$$(\overline{c_2} + \overline{\gamma_{p_1,c_2}} + \overline{\gamma_{p_3,c_2}})(\overline{c_1} + \overline{\gamma_{p_1,c_1}} + \overline{\gamma_{p_6,c_1}}) \cdots$$

Implicant Cost (shown for p_0, p_1):

$$(\overline{\gamma_{p_0,c_0}} + p_0)(\overline{\gamma_{p_1,c_0}} + p_1)(\overline{\gamma_{p_1,c_1}} + p_1)(\overline{\gamma_{p_1,c_2}} + p_1)$$
$$(\overline{\gamma_{p_1,c_3}} + p_1)(\overline{\gamma_{p_1,c_4}} + p_1) \cdots$$

Selecting compatibles $\{s_0, s_1\}$ and $\{s_1, s_2\}$ gives two state-mapping choices in $\langle$ 00 $s_1' \rangle$, as observed earlier. The sub-optimal 4-RGPI cover $\{p_1, p_7, p_9, p_{11}\}$ corresponds to choosing s_1'. Our method instead finds the minimum cover $\{p_1, p_7, p_{11}\}$, with 3 RGPI's, which corresponds to the optimal state-mapping of s_0'. $\square$

Example: The following example demonstrates that our method, which examines *non-minimum* cardinality state covers, forms the optimal solution which other methods cannot find. In contrast, $\mathcal{M}$, when fully reduced (as other methods would do), results in a sub-optimal logic cover.

$\mathcal{M}$	00	01	11	10
s_0	s_2,1	s_1,0	s_1,-	s_0,0
s_1	s_2,0	s_1,-	s_1,-	-,-
s_2	s_2,-	s_1,1	s_1,1	s_0,1

Prime compatibles: $\{s_0\}$ and $\{s_1, s_2\}$.

$\mathcal{M}'$	00	01	11	10
s'_0	s'_1,1	s'_1,0	s'_1,-	s'_0,0
s'_1	s'_1,0	s'_1,1	s'_1,1	s'_0,1

$\mathcal{M}$, reduced via $\{s_0\}$ and $\{s_1, s_2\}$.

The minimum logic covers for these two machines are shown below, for $\mathcal{M}$ on the left (RGPI's), and for $\mathcal{M}'$ on the right (instantiated RGPI's):

$p_1 : \langle \texttt{00}\ \ s_0, s_2\ \ s_2\ \ \texttt{1} \rangle$ $\quad$ $p'_1 : \langle \texttt{0}-\ \ s'_0, s'_1\ \ s'_1\ \ \texttt{0} \rangle$

$p_2 : \langle -\texttt{1}\ \ s_0, s_1, s_2\ \ s_1\ \ \texttt{0} \rangle$ $\quad$ $p'_2 : \langle \texttt{00}\ \ s'_0\ \ s'_1\ \ \texttt{1} \rangle$

$p_3 : \langle \texttt{10}\ \ s_0, s_1, s_2\ \ s_0\ \ \texttt{0} \rangle$ $\quad$ $p'_3 : \langle -\texttt{1}\ \ s'_1\ \ s'_1\ \ \texttt{1} \rangle$

$p_4 : \langle \texttt{00}\ \ s_0, s_1, s_2\ \ s_2\ \ \texttt{0} \rangle$ $\quad$ $p'_4 : \langle \texttt{10}\ \ s'_0, s'_1\ \ s'_0\ \ \texttt{0} \rangle$

$p_5 : \langle --\ \ s_2\ \ -\ \ \texttt{1} \rangle$ $\quad$ $p'_5 : \langle \texttt{11}\ \ s'_0, s'_1\ \ s'_1\ \ \texttt{1} \rangle$

$p'_6 : \langle \texttt{10}\ \ s'_1\ \ s'_0\ \ \texttt{1} \rangle$

It is not hard to show that these two state and logic covers are two different solutions to our constraints.

Our method produces the optimal (unreduced) implementation $\mathcal{M}$, with cover $p_1, \ldots, p_5$ and 3 states. Consider p_5, which is selected in our cover. Its use enables a minimum cover, since it covers all the ON-set minterms of the output in s_2. In contrast, p_5 cannot be used in machine $\mathcal{M}'$, since it cannot be mapped over s'_1. As a result, in $\mathcal{M}'$, 2 RGPI's are needed to cover the output's ON-set minterms in s_2. Our method therefore finds the minimum cover based on the selection of compatible $\{s_2\}$, and discards the sub-optimal solution based on $\{s_1, s_2\}$.

It is important to observe that the minimum logic cover was only achievable by using *non-prime* compatibles ($\{s_1\}$ and $\{s_2\}$), which in turn necessitated a non-minimum cardinality state cover. Thus, our method finds the optimal solution, while existing tools, which only consider minimum-cardinality state covers of prime compatibles, cannot guarantee minimum logic complexity. □

8 Experimental Results

The algorithm was implemented in C++ and run under MkLinux for PowerPC. Results appear below for a small set of examples, using only RGPI seeds. For comparison, STAMINA [12] was run on the same tables. For each, the number of compatibles, seeds, covering constraints, run-time and products is given. Entries marked with a † used *all* compatibles; others used only prime compatibles.

FSM						**#products**^{a}	
	i/o/s	c	seeds	constr	time	OPT	Stam
lisom	1/2/3	5†	16	142	0.3	3	3
minst	2/1/3	4†	20	226	0.4	5	6
lion9	2/1/9	5	42	219	1006	8	8

^{a}Symbolic products after mvi minimization.

Figure 2: Experimental Results

9 Conclusions and Future Work

We have presented the first method for optimal state minimization which directly and accurately targets logic complexity, achieving a provably exactly optimal result under input encoding. The method is computationally expensive, however, and would benefit greatly from heuristic and inexact variations. Unlike some methods, ours provides several opportunities to reduce complexity, e.g. using prime or maximal compatibles, RGPI seeds only, or heuristic binate solvers, while retaining a strong minimization framework. For more efficient exact solution, recent innovations in implicit methods [4, 14] hold particular promise. With all of these choices, we believe our method offers a framework encompassing a spectrum of solutions.

A limitation to the current work is the large size of the RGPI set. Initial investigation suggests cases where the set can be pruned, while still guaranteeing an optimum solution. This is an important area for further research.

Although we have only provided limited benchmarks, we expect that future experimentation will prove our method competitive with existing methods (e.g. STAMINA, SMAS). Finally, we anticipate that an extension to output encoding will yield a definitive solution to this problem.

Acknowledgments. We greatly thank Dr. Olivier Coudert (Synopsys) for early access to Scherzo, and Dr. Ruchir Puri (IBM) for insightful discussions.

References

[1] M.J. Avedillo, J.M. Quintana, and J.L. Huertas. Smas: A program for the concurrent state reduction and state assignment of finite state machines. In *ISCAS*, pages 1781–1784, 1991.

[2] R.K. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *ICCAD-1989*, pages 316–319.

[3] N.L.V. Calazans. Boolean constrained encoding: A new formulation and a case study. In *ICCAD-1994*, pages 702–706.

[4] O. Coudert and J. Madre. Implicit and incremental computation of primes and essential primes of boolean functions. In *DAC-1992*, pages 36–44.

[5] O. Coudert and J.C. Madre. New ideas for solving covering problems. In *DAC-1995*, pages 641–646.

[6] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[7] G. De Micheli, R.K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE Trans. on CAD*, CAD-4(3):269–285, July 1985.

[8] S. Devadas and A.R. Newton. Exact algorithms for output encoding, state assignment, and four- level boolean minimization. *IEEE Trans. on CAD*, CAD-10(1):13–27, January 1991.

[9] J. Hartmanis et al. Some dangers in state reduction of sequential machines. *Information and Control*, pages 252–260, September 1962.

[10] R.M. Fuhrer, B. Lin, and S.M. Nowick. Symbolic hazard-free minimization and encoding of asynchronous finite state machines. In *ICCAD-1995*, 1995.

[11] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE TEC*, EC-14:350–359, June 1965.

[12] G. Hachtel, J.K. Rho, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Trans. on CAD*, CAD-13(2):167–177, February 1994.

[13] J. Hartmanis and R.E. Stearns. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, 1966.

[14] T. Kam, T. Villa, R.K. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *DAC-1994*.

[15] B. Lin and F. Somenzi. Minimization of symbolic relations. In *ICCAD-1990*, pages 88–91.

[16] E.J. McCluskey. *Logic Design Principles.* Prentice-Hall, 1986.

[17] M. Paull and S. Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. on Elec. Comp.*, EC-8:356–367, Sept. 1959.

[18] R. Puri, 1995. Private communication.

[19] R. Rudell and A. Sangiovanni-Vincentelli. Multiple valued minimization for PLA optimization. *IEEE Trans. on CAD*, CAD-6(5):727–750, Sept. 1987.

[20] T. Villa and A. Sangiovanni-Vincentelli. NOVA: state assignment of finite state machines for optimal two-level logic implementation. *IEEE Trans. on CAD*, 9(9):905–924, Sept. 1990.