A Fast State Reduction Algorithm for Incompletely Specified Finite State Machines

Hiroyuki Higuchi and Yusuke Matsunaga
FUJITSU LABORATORIES LTD, Kawasaki 211-88, Japan

Abstract— This paper proposes a state reduction algorithm for incompletely specified FSMs. The algorithm is based on iterative improvements. When the number of compatibles is likely to be too large to handle explicitly, they are represented by a BDD. Experimental results are given to demonstrate that the algorithm described here is faster and obtains better solutions than conventional methods.

1 Introduction

Reducing the number of states in incompletely specified finite state machines(ISFSMs) is an important step in FSM synthesis. It may result in a fewer number of state variables needed to encode the states. Even if the number of state variables is not reduced, many unused states are introduced. They can be used as don't-cares in combinational logic synthesis. State reduction also reduces the size of the machine for subsequent steps in the synthesis. As a result, algorithms for state assignment and test generation etc. can perform much better.

The problem of minimizing ISFSMs has been studied by a number of researchers [1, 2]. However these are not suitable for large problems because the problem is NP-hard. In order to handle practical problems, heuristic algorithms for obtaining near-minimal solutions have been proposed [3, 4, 5, 2]. Though they can reduce medium-sized machines in practical time, they require considerable time for larger machines.

In this paper we present a fast state reduction algorithm. It depends upon ESPRESSO-style iterative improvement[6]. The algorithm consists of generating the set of all the maximal compatibles as an initial solution and attempting to reduce each compatible in the set by iterative improvements in order to reduce the size of the solution. The number of the maximal compatibles is, however, exponential to the number of the original states in the worst case. When the number of the maximal compatibles is likely to explode, we utilize Binary Decision Diagrams(BDDs) to avoid such combinational explosion. Experimental results show that the proposed method is faster and obtains better solutions in many cases than conventional ones.

	NS, z			
PS	I_1	I_2	I_3	I_4
A	$B, 0$	$-, -$	$-, -$	$E, 1$
B	$A, 0$	$C, -$	$-, -$	$-, -$
C	$C, -$	$A, 1$	$D, 1$	$E, -$
D	$-, -$	$B, -$	$-, -$	$A, -$
E	$C, -$	$B, 1$	$F, -$	$-, 0$
F	$F, 1$	$A, -$	$E, -$	$B, -$

Figure 1: Machine M_1

2 Preliminaries

A *Finite State Machine* M is a 5-tuple $(I, O, S, \delta, \lambda)$, where $I, O,$ and S are finite nonempty sets of inputs, outputs, and states, respectively; $\delta : I \times S \rightarrow S$ is the state transition function; $\lambda : I \times S \rightarrow O$ is the output function. An FSM is *incompletely specified*, if either the next state or the output is not specified for at least one (input,state) pair.

Two output sequences of machine M are *compatible* if and only if their corresponding outputs are not conflicting, i.e., identical whenever both outputs are specified. An input sequence is *applicable* to state s_i of M if no unspecified next state is encountered. Two states s_i, s_j of M are *compatible* if and only if s_i, s_j yield compatible output sequences for every input sequence applicable to both s_i and s_j. Two states are *incompatible* if they are not compatible.

Theorem 1 *Two states of machine M are compatible if and only if, for every input, (i)their outputs are not conflicting and (ii) their next states are compatible.*

Let us introduce an example machine M_1. The state table is shown in Fig.1. An unspecified (or don't-care) entry is denoted by a "–". In the table, state pair (AB) is compatible. Therefore (DF) is also compatible. (AE) is incompatible, because the outputs of A and E for input I_4 are conflicting.

Let the next states of s_i and s_j be s'_i and s'_j, respectively; then $(s'_i s'_j)$ is said to be *implied* by $(s_i s_j)$, or an implied pair of $(s_i s_j)$.

A set of states is compatible if and only if each pair of states in the set is *compatible*. A set of states is *incompatible* if it is not compatible. In this paper, a compatible set is simply called a *compatible*. For example, set of states (ABC) is compatible in M_1.

The *implied set* of compatible c for input i is the set of next states of states in c for input i. A compatible containing the implied set of c for i is said to be an *implied compatible* of c for i. In the example of Fig.1, the implied set of compatible (DEF) for input I_1 is (CF) and an implied compatible is

(CEF). The *class set* $CS(c)$ of compatible c is the set of all the implied sets d such that (i) $|d| > 1$, (ii) $d \not\subseteq c$, and (iii) $\forall d' \in CS(c), d \not\subset d'$. The class set of a compatible expresses the closure conditions imposed by the compatible. The class set of compatible (DEF) in Fig.1 is $\{(AB), (CF)\}$.

A set of compatibles for a machine is said to be *closed* if, for every compatible in the set, all its implied compatibles are also contained in the set. A closed set of compatibles which contains all the states of the machine is called a *closed cover*. The *state reduction problem* of an ISFSM is to find a small closed cover of the original machine. For M_1, $\{(ABC), (DE), (F)\}$ is a closed cover. Therefore the reduced machine has 3 states. Each compatible corresponds to a state of the reduced machine.

A compatible is said to be *maximal* if it is not a proper subset of another compatible. For M_1 maximal compatibles are (ABC), (BCE), (BDE), (CEF), (DEF).

3 A fast state reduction procedure

In this section we propose a fast state reduction algorithm.

3.1 Outline

Our method attempts to reduce each compatible in the solution iteratively, that is, to delete unnecessary states in the compatible. By reducing a compatible, we have two possibilities for decreasing the size of the closed cover:

- Some compatibles may be contained by another one. Therefore they can be deleted.
- Implied sets of reduced compatibles become smaller. As a result, another compatibles in the solution are also likely to be reduced.

The state reduction procedure is as follows:

1. Check if each state pair is compatible.
2. Generate all the maximal compatibles.
3. Solution $S \leftarrow$ "all the maximal compatibles".
4. $S \leftarrow$ REDUCE-I(S).
5. $S \leftarrow$ MERGE(S).
6. $S \leftarrow$ EXPAND(S).
7. $S \leftarrow$ REDUCE-II(S).
8. $S \leftarrow$ MERGE(S).

We use the set of all the maximal compatibles as an initial solution, according to the following theorem.

Theorem 2 *The set of all the maximal compatibles is a closed cover.*

3.2 Reduce-I

The initial solution is the set of all the maximal compatibles. For the covering condition we only have to consider maximal compatibles. However a subset of a maximal compatible may be better for the closure condition, because its implied sets may be smaller. REDUCE-I attempts to reduce each compatibles iteratively, keeping the solution closed.

For example, consider the machine M_1 again. The initial set of compatibles and their class sets are shown in Fig.2.

compatible	class set	core
ABC	$\emptyset$	ABC
BCE	ABC, DF	$\emptyset$
BDE	AC, BC	BE
CEF	AB, DEF, BE	CF
DEF	AB, CF	DEF

Figure 2: Initial solution and its core subsets

```
REDUCE-I($CompSet\ S$) {
1.  while ($S$ is changed) {
2.      $QC \leftarrow$ the set of essential compatibles for $S$;
3.      if ($QC = \emptyset$) $QC \leftarrow$ an compatible in $S$;
4.      for (each compatible $c$ in $S$) $core(c) \leftarrow \emptyset$;
5.      while ($QC \neq \emptyset \vee$ there exists uncovered states) {
6.          if ($QC = \emptyset$) $QC \leftarrow$ an compatible in $S$;
7.          Take a compatible $c \in QC$ and delete it from $QC$;
8.          for (each implied set $d$ in the class set of $c$) {
9.              Select a compatible $c'$ such that $d \subseteq c' (\in S)$;
10.             if ($core(c') = \emptyset$) $QC \leftarrow QC \cup c'$;
11.             $core(c') \leftarrow core(c') \cup d$;
12.         }
13.     }
14.     $S \leftarrow$ the set of the core subsets;
15.     Check and satisfy the covering condition;
16.     Delete core subsets contained by another core subset;
17.     Check and satisfy the closure condition;
18. }
19. return $S$;
}
```

Figure 3: Reduce-I

To satisfy the closure condition of (BCE) in the initial solution, the implied set (ABC) and (DF) must be contained by at least one compatible, respectively. Theorem2 implies that there is at least one maximal compatible containing each implied set. In this case, compatible (ABC) contains (ABC) and compatible (DEF) contains (DF). Therefore (ABC) itself out of (ABC) and (DF) out of (DEF) are necessary for the closure condition. Thus a core subset of each compatible in the solution is obtained by extracting necessary portion for satisfying closure conditions of every compatible. For example, core subsets for the initial solution is also shown in Fig.2. The next solution consists of the core subsets. The size of the solution is decreased from 5 to 4. Since each member is reduced, its implied sets are likely to be reduced. For example, compatible (CEF) is reduced to (CF). As a result, the class set is also reduced to $\{(DE), (BE)\}$. This may cause further reduction of the solution in the next iteration.

The overall procedure of REDUCE-I is shown in Fig.3. Essential compatibles on line 2 is defined as follows.

Definition 1 *A compatible c in a closed cover S is* essential *for S, if c contains at least one state which is not covered by any other compatible in S.*

The procedure consists of selecting essential compatibles for the current solution and attempting to satisfy its closure conditions by selecting a compatible in the solution and expanding core subsets. If there are no essential compatibles, a compatible with the highest gain is selected. The gain of a compatible c is calculated as follows:

$$gain = \frac{\text{number of uncovered states in } c}{(\sum_{d \in CS(c)} \text{number of implied pairs in } d\,)/|I|}.$$

When a compatible is large or its class set is small, its gain becomes large.

If the selected compatibles do not cover all the states, the procedure is repeated, this time starting with a compatible that covers most of uncovered states. A compatible is selected on line 9 in Fig.3 to cover a target implied set d. This process is assisted by noting that each core subset is as small as possible. After core subsets are generated, it is checked whether they satisfy covering and closure conditions. When they are violated, some core subsets are expanded to their corresponding compatibles. The reduction of the solution is repeated while there are some changes in the solution.

3.3 Merge

After REDUCE-I, some pair of compatibles in the solution can be still merged into one compatible. By merging these compatibles, the size of solution is reduced. Procedure MERGE attempts to merge each pair of compatibles.

3.4 Expand

Procedure EXPAND facilitates improvement over the local minimum obtained by REDUCE-I. It takes, in turn, each compatible in the solution and expands it to the maximal compatible. The benefit is that a larger compatible can generally be reduced in more directions than a smaller compatible. Thus EXPAND often allows us to move away from a locally minimal solution towards a yet smaller closed cover.

3.5 Reduce-II

We use a variation of REDUCE-I after EXPAND. We call it REDUCE-II. In REDUCE-I, core subsets are calculated by using the implied sets of compatible itself. However compatibles are to be reduced to their core subsets. Therefore some subset of implied sets are really necessary for the closure condition. REDUCE-II uses compatible pairs in each compatible instead of compatible itself in the core subset calculation. The number of pairs is generally larger than the number of compatibles. However REDUCE-II does not take much time, because the size of solution to be dealt in REDUCE-II is generally much smaller than that in REDUCE-I.

4 Implicit generation of initial solutions

Since the number of maximal compatibles is exponential in the number of the original states in the worst case, it sometimes takes a considerable time to generate and manipulate them explicitly. Therefore, if the number of compatible pairs exceeds a given threshold, compatibles are generated implicitly. In this section we discuss the implicit generation of compatibles and the extraction of initial solutions.

4.1 Implicit representation of compatibles

In state reduction, compatibles, i.e. sets of states, need to be represented and manipulated efficiently. To represent sets of states or sets of sets of states implicitly, we use Binary Decision Diagrams(BDDs). The representation of compatibles is the same as in [7]. A set of sets of states is represented as a set S of positional-sets by a characteristic function $\chi_S : B^n \rightarrow B$, where n is the number of the states. Characteristic function $\chi_S(\vec{x}) = 1$ if and only if the set of states represented by the positional-set $\vec{x}$ is in the set S. A BDD representing $\chi_S(\vec{x})$ contains minterms, each of which corresponds to a set of states in S.

4.2 Implicit generation of compatibles

We generate all the compatibles implicitly by using the information of pair-wise compatibility checking. Since incompatibles are the sets containing at least one incompatible pair, the set of all the compatibles is calculated by iteratively excluding all the sets including each incompatible pair from a set of all the sets of states without the empty set.

4.3 Extraction of initial solutions

The procedure of extracting initial solutions from the BDD representing the set of all the compatible corresponds to the first iteration in procedure REDUCE-I. For the sake of efficiency, this procedure selects a compatible which contains the largest number of uncovered states instead of essential compatibles or compatibles with high gains.

5 Experimental results

We implemented our algorithm in a program called SLIM, a SequentiaL machIne Minimizer. We ran SLIM on all the MCNC benchmarks and several large FSMs [7]. Comparisons are made with heuristic methods in STAMINA [2]. In the experiments the threshold for implicit manipulation was set to 10000, that is, if the number of compatible pairs is more than 10000, compatibles are implicitly generated.

Table 1 summarizes the results of state reduction for MCNC FSMs. Actually all the FSMs were run. Those not reported, either have no compatible states, or have all states compatible. For all the cases not reported, the processing times were negligible. In Table 1, Column N_s show the numbers of states of the machines: Columns $init$, min, $slim$, $stam$ indicate the numbers of states in the original machines, strictly minimized machines, machines reduced by SLIM, and machines reduced by heuristic methods in STAMINA, respectively. Column $time$ shows the CPU time for state reduction. Times are referred to Sun SPARCstation10(96Mb). The results for STAMINA are the best results among three heuristics.

Table 1 shows that SLIM obtained the minimum solutions on all the MCNC benchmark FSMs within 1 second. Though

Table 1: Experimental results(MCNC benchmarks)

FSMs	N_s				time(sec.)	
	init	min	slim	stam	slim	stam
bbara	10	7	7	7	0.03	0.02
bbsse	16	13	13	13	0.08	0.03
beecount	7	4	4	4	0.03	0.00
ex1	20	18	18	18	0.15	0.04
ex2	19	5	5	5	0.14	15.47
ex3	10	4	4	4	0.06	0.15
ex5	9	3	3	3	0.03	0.05
ex7	10	3	3	3	0.03	0.07
lion9	9	4	4	4	0.04	0.01
mark1	15	12	12	12	0.10	0.02
opus	10	9	9	9	0.04	0.00
scf	121	97	97	97	0.23	0.41
sse	16	13	13	13	0.11	0.02
tbk	32	16	16	16	0.68	1.09
train11	11	4	4	4	0.02	0.02

STAMINA also obtained the minimum solutions, it required much time on $ex2$. When the exact method in STAMINA is applied to $ex2$, it required 2484 seconds.

Table 2 gives the results of state reduction for some large FSMs. Data in the parentheses in Column min show the best results so far, which are not proven to be the minimum. Table 2 shows that SLIM completed for all the tested large FSMs, while STAMINA failed on some FSMs. SLIM completed every FSM in 200 seconds. These results indicates that SLIM is robust from a practical point of view. As for the number of states, SLIM obtained smaller machines for most of the examples than conventional methods, especially for $th.55$. The implicit method described in Section4 was applied to $rubin600$, $rubin1200$ and $rubin2250$.

Experimental results show that SLIM can reduce the number of states efficiently. It is mainly because we do not solve the minimum closed covering problem exactly. Heuristic methods in STAMINA solve the problem exactly for restricted set of compatibles. Therefore the computational time required for these algorithms becomes considerably large in Table2, though they are efficient and obtain near-optimum solutions for small FSMs such as MCNC benchmarks.

6 Conclusions

We have described a fast state reduction algorithm for ISFSMs. The algorithm consists of generating maximal compatibles as an initial solution and attempting to reduce each compatible in the solution by iterative improvements. When the number of the maximal compatibles is likely to explode, we utilize BDDs to avoid such combinational explosion. Experimental results indicate that the proposed algorithm is efficient in terms of both computational time and reduction ability.

Acknowledgments The authors would like to thank Dr. Timothy Kam and Dr. Tiziano Villa of University of California at Berkeley for providing us with the benchmarks of the large FSMs.

Table 2: Experimental results(Large FSMs)

FSMs	N_s				time(sec.)	
	init	min	slim	stam	slim	stam
alex1	42	6	6	6	0.75	21.9
intel_edge.dummy	28	4	6	4	0.12	0.86
isend	40	4	4	4	0.21	2.44
pe-rcv-ifc.fc	46	2	2	2	0.14	0.40
pe-rcv-ifc.fc.m	27	2	2	2	0.10	0.09
pe-send-ifc.fc	70	2	3	2	0.29	1.93
pe-send-ifc.fc.m	26	2	2	2	0.09	0.05
vbe4a	58	3	3	3	4.23	331
vmebus.master.m	32	2	2	2	0.60	0.50
th.30	31	(8)	7	9	0.13	1.20
th.40	41	(12)	9	15	0.32	1.13
th.55	55	–	13	24	8.72	5.88
fo.20	21	(4)	3	3	0.03	36.0
fo.50	51	–	9	11	0.21	9.25
fo.70	71	–	10	14	0.57	23.8
ifsm0	38	3	3	3	0.15	0.10
ifsm1	74	(14)	14	15	0.15	0.49
ifsm2	48	9	9	9	0.24	862
rubin18	18	3	3	3	0.34	0.00
rubin600	600	3	3	–	12.7	fails
rubin1200	1200	3	3	–	53.9	fails
rubin2250	2250	3	3	–	199	fails
e271	19	2	2	2	0.04	0.03
e285	19	2	2	2	0.04	0.02
e304	19	2	2	2	0.05	0.02
e423	19	–	3	3	0.09	0.99
e680	19	2	2	2	0.06	0.02

References

[1] A. Grasselli and F. Luccio. "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks". *IRE Trans. on Elect. Comp.*, vol.14, pp.350–359, June 1965.

[2] J.-K. Rho, G. Hachtel, F. Somenzi, and R. Jacoby. "Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines". *IEEE Trans. on Computer-Aided Design*, vol.13, pp.67–177, Feb. 1994.

[3] R. G. Bennetts, J. L. Washington, and D. W. Lewin. "A Computer Algorithm for State Table Reduction". *Radio and Electronic Engineer*, vol.42, pp.513–520, Nov. 1972.

[4] M. A. Perkowski and N. Nguyen. "Minimization of Finite State Machine in System SuperPeg". in *28th Midwest Conf. on Circuits and Systems*, pp.139–147, Aug. 1985.

[5] L. N. Kannan and D. Sarma. "Fast Heuristic Algorithms for Finite State Machine Minimization". in *Proc. of European Design Automation Conf.*, pp.192–196, Feb. 1991.

[6] R. Btayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[7] T. Kam, T. Villa, R. Brayton, and A.Sangiovanni-Vincentelli. "A Fully Implicit Algorithm for Exact State Minimization". in *31st ACM/IEEE Design Automation Conf.*, pages 684–690, June 1994.