Fast Heuristic Algorithms for Finite State Machine Minimization

L. N. Kannan
Cadence Design Systems
3393 Octavius Dr., Santa Clara, CA 95054

D. Sarma
Dept. of Computer Science
Rutgers University, 311 N. 5th St., Camden, NJ 08102

Abstract

In this paper, a technique for the minimization of completely and incompletely specified sequential machines is described. By employing fast heuristic algorithms, it has been shown that it is possible to effectively reduce large (121 states) finite state machines in reasonable computing time when compared to other methods. It has been shown that it is possible to achieve area/literal reductions in the range of 30-100% over unreduced machines using this technique.

1 Introduction

The state table of a finite state machine (FSM) often contains redundant states which may be introduced into the state table either by the designer or by the synthesis program that generates the state table from a high-level description. Elimination of the redundant states in an FSM offers a number of significant advantages. It may make the operation of the machine easier for the designer to understand and at the same time may result in a fewer number of state variables needed to encode the table. Even if the number of state variables is not reduced, the introduction of additional "don't care" rows in the table can be used to advantage by a good state assignment program. Either way, this results in savings in the logic needed to implement the FSM.

State reduction also reduces the size of the machine for the subsequent steps in the synthesis process. As a result of this, algorithms for state assignment etc. can perform much better.

A number of programs have been developed in recent years for FSM synthesis. Almost all of these use heuristic techniques for state assignment and decomposition to achieve a reduction in the logic complexity of the final realization. However, none of these make effective use of the equivalent or compatible states present in the FSM to obtain reduced number of states - resulting in non-minimal realizations.

In this paper we describe efficient heuristic algorithms to minimize large FSMs in reasonable time. The effectiveness of the approach is shown with the help of a number of benchmark examples.

2 Previous Work

Given a *completely specified* FSM, there is a technique which operates in polynomial time to determine if there are two or more equivalent states [1, 2].

The problem of minimizing *incompletely specified FSMs* has been studied by a number of researchers. Paul and Unger [3] developed a general theory and provided methods for generating the maximal compatibles and for obtaining the minimal closed cover. However, their tabular method for generating the maximal compatibles is lengthy and the closure test for the cover involves large amounts of enumeration and inspection and their method is not readily implementable on a computer.

Grasselli and Luccio [4, 5] developed the concept of prime classes and formulated the minimization problem in the form of a linear integer program. However, their method, while suitable for computer implementation is computationally quite complex.

DeSarkar *et al.* [6, 7] simplified the procedure of Grasselli and Luccio by first tackling the problem of closure and then that of cover. They generate prime closed sets for all prime compatibles and then select a union of prime closed sets representing the minimal cover. Both of these are computationally expensive procedures.

Kella [8] proposed a method that avoided the generation of all maximal compatibles. Yang [9] proposed yet another method in which "superseded" compati-

2130/91/0000/0192$01.00 © 1991 IEEE

bles are eliminated from the search thus substantially reducing the number of compatibles to be considered for a minimal closed cover. Biswas [10] used primary and secondary "implication trees" associated with each maximal compatible to obtain the minimized state machine cover the incompletely specified machine. Rao and Biswas [11] start with the set of all subsets of the maximal compatibles as possible candidates and then use various theorems to eliminate some of these from further consideration in their search for the minimal closed cover.

All the papers mentioned above attempt to arrive at the exact solution using a number of rules etc. to reduce the search. Many of them are tailored towards hand calculation. Even those that are implementable on the computer are not suitable for really large problems because of the nature of the minimization problem which is NP-complete [12].

The first attempt using heuristic algorithms to obtain near-minimal solutions was by Bennetts *et al.* [13]. In their method they start with the prime compatible set (PCS) or the maximal compatible set (MCS) and using heuristic techniques, identify essential and quasi-essential MCSs and generate a closed set of MCSs and then continue till all the states are covered. However their technique is still slow (251 seconds for a 22 state machine) because they start with the PCS/MCS as the starting point which could be very large for some machines (like *ex2*). More recently, Perkowsky and Nguyen [14] have proposed the use of rules and evaluation functions for cutting off and ordering the branches of the search tree but their method is also slow (60-100 seconds for a 10 state machine).

3 The Minimization Procedure

The minimization procedure for an incompletely (completely) specified machine in the general case is as follows:

1. From the state table of the FSM obtain a list of all the pairwise compatible states of the machine.
2. Generate the set of all maximal compatibles.
3. Get the set of maximal compatibles that cover all states (minimal cover). In case of completely specified machines, we are done at this point.
4. Expand on the minimal cover to obtain an optimal closed cover (in case of incompletely specified machines).
5. Generate the state table of the reduced machine from the optimal closed cover.

We now describe algorithms for effectively accomplishing each of these steps.

3.1 Generating the Maximum Compatibles

To generate the maximum compatibles, a pairwise incompatibility table for the states of the machine [15] is first constructed. This is done in two phases. First, the outputs are checked for incompatibility. If the state pairs are incompatible, they are flagged as such, otherwise a list of implied pairs is constructed which reflect all the dependencies of the compatible pair. Next, all the pairs of states that are conditionally compatible are checked for the presence of incompatible implied pairs. If any are found, that conditionally compatible pair is now flagged incompatible. This process is repeated till no more incompatibles are found.

From this list of pairwise incompatibles the maximal compatibles are obtained using the technique [15] given below:

1. Form a Boolean product-of-sums expression with each term composed of the members of an incompatible pair in which every pair occurs exactly once.
2. Find the prime implicants (PIs) of this expression.
3. Generate a maximum compatible from each PI by taking the set of states *not* represented in the PI.

The computationally expensive step here is to find all the PIs (Step 2) and this is done by using ESPRESSO[1] [16, 17] with the -epos option. (The -epos option is used to minimize the OFF-set of the function)

3.2 Generating the Minimal Cover

Having obtained the set of all maximum compatibles we need to obtain a minimal cover. Since only one minimal cover is needed a very efficient heuristic algorithm was developed which gave excellent results. This is described below.

3.2.1 The Minimal Cover Algorithm

Start with C as the set of maximal compatibles. Let $\mathcal{M}$ be the minimal cover (empty to begin). Choose the state s that occurs the fewest number of times amongst all the compatible sets in C. Find the compatible set $c_{i(max)}$ in C with the most number of states that contains this state (s). Find the corresponding maximal compatible ($mc_{i(max)}$) and add this to the minimal

[1] a widely available logic minimization program developed at UC Berkeley.

cover $\mathcal{M}$. Now delete all states $s_i(\in c_{i(max)})$ from the compatible sets c_j $(j \neq i)$ and delete the compatible set $c_{i(max)}$ from C. Repeat this process with the (now) reduced set of compatibles C till there are no more states remaining in the compatible sets c_i still in C (i.e. all states of the machine are covered). $\mathcal{M}$ now contains the minimal cover.

The procedure (called min_cov) was tested with a number of examples and was found to be very effective. It produced the minimal cover in all cases that were tried and was also extremely fast.

3.3 Obtaining the Optimal Closed Cover

A set of compatibles can be used to construct the machine only if two conditions are met - every row of the original table must be covered by at least one compatible and secondly, any set of next-states implied by any compatible in the set for a particular input must be a subset of at least one compatible in the set. This latter property is called closure. For the case of completely specified machines, the minimal cover is closed - so the minimal cover generated in the earlier procedure is adequate. Another special case is when the minimal cover is the set of all maximal compatibles. Since this is always closed we need proceed no further. Aside from these two special cases, however, we need to to get a closed cover set that is as small as possible.

Since finding the closed cover is NP-complete, an efficient heuristic algorithm was developed that is easily implementable on a computer and able to perform well for large machines. The minimal cover (lower bound) gives us a good starting point for the heuristic procedure. The strategy essentially involves adding or deleting state(s) from the minimum cover in order to obtain closure of the entire set while trying to keep the size of the cover to a minimum.

Two heuristic algorithms are proposed for this.

Heuristic I (the large set approach): Start with the entire set of compatibles in the minimal cover. Then check each compatible for closure and add appropriate compatible sets of states to the cover till we have a closed set. At every step, try to add these extra states to existing compatible sets starting from the first newly created compatible set (we cannot add to the original sets we started with because they are all maximal compatibles) and create new sets only if absolutely necessary.

Heuristic II (the lean set approach): Start with the set of maximal compatibles in the minimal cover but remove all the states that occur more than once in the compatibles. If this is done starting from the first maximal compatible, all sets (with the exception of the first) may have some states less than the corresponding maximal compatible. So, excluding the first compatible, try and add states to all the other compatibles or add a new compatible set to satisfy the closure requirement. This approach gives a smaller closed cover with some of the examples.

The procedure uses either heuristic to try and grow the minimum cover until it is closed. If the size of the cover equals the number of the maximal compatibles or the number of states of the machine, the program terminates.

3.4 Generating the Reduced Machine

This is done as follows. For all the compatible sets in the closed cover, generate a list of next-states for all the present-states in the set for every input combination. Then find a compatible set from the closed cover that covers all the sets in the list and enter this compatible set as the next state for the present-state set. The output for this row would be the output that covers all the outputs for these next states.

4 Results and Discussions

The entire system consists of about 2500 lines of C and runs under UNIX on a SUN workstation.

The program (called FSMRED) was tried on a number of FSM examples. Most of these were from the MCNC Benchmarks collection. Others were from various textbooks [1, 15]. The results of these experiments are summarized in Tables 1, 2, 3, and 4. All times in CPU seconds on the SUN-3/60 About half the example machines (22 of them) did not have any redundant states. For these machines the added overhead of running FSMRED was very low (about 4%) (see Table 1). The state assignment program NOVA [18] was used to obtain optimal two-level (PLA) implementations. It is seen that using FSMRED+NOVA results in a smaller area of the final PLA in all cases (see Table 2). The reductions range from 30% to 100%. It is also seen that, since the input to NOVA is now a reduced machine, the NOVA run-time is reduced as well, and the total run-time (FSMRED+NOVA) is *less* than the run time with NOVA alone.

Experiments were also conducted for multi-level realizations. The state assignment packages NOVA and MUSTANG were used in conjunction with the multi-level optimization system MIS-II for optimal multi-level realizations of both the original and the reduced machines. The results are summarized in Tables 3, and 4. It is seen that for multi-level realizations, using FSMRED+(NOVA/MUSTANG)+MIS results in a fewer number of literals in almost all the cases as

well as reduced run-times when compared to (NOVA/MUSTANG)+MIS by themselves. The reductions in literals range from 12.5% to 100%.

In three of the MCNC examples (*donfile*, *modulo12* and *s8*), *all* states were compatible. So the machines can be reduced to a trivial machine whose output is at the same level for any input whereas both NOVA and MUSTANG (by themselves) realize these with a lot of logic! (see Tables 2, 3, and 4).

For some large examples (*scf*, *tbk*), NOVA and MUSTANG (with MIS-II) were not able to produce solutions when applied to the original machine while they could easily handle the reduced machines.

When compared to exact solutions, for small machines, FSMRED gave minimal solutions in some cases and just one state more in others. For the bigger machines, though, the exact solution could not be obtained by hand computation and so it is hard to say how effective the heuristics are. But judging by the area reductions obtained even with near minimal solutions, we can see that a preprocessor such as FSMRED is necessary before state assignment is attempted using programs like NOVA and MUSTANG.

5 Conclusion

In this paper efficient heuristic algorithms have been developed for the minimization of FSMs. The algorithms are effective in finding the minimal cover (for completely specified machines) and the optimal closed cover (for incompletely specified machines). Though the problem of machine minimization is computationally expensive, we have been able to show that it is possible to minimize large FSMs (e.g. *scf* with 121 states) in reasonable time. It has been found that the performance of state assignment algorithms (NOVA and MUSTANG) improves considerably when applied to the reduced machine. The reduction in area is considerable and the total time (for state reduction and assignment) is found to be less than or equal to the state assignment time alone when applied to the original machine.

References

[1] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978.

[2] J. Hopcroft, "An n log n Algorithm for Minimizing States in a Finite Automaton," in *Theory of Machines and Computations* (Z. Kohavi and A. Paz, eds.), New York: Academic Press, 1971.

[3] M. C. Paull and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 356–367, 1959.

[4] A. Grasselli and F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IEEE Trans. on Electronic Comptrs.*, vol. EC-14, pp. 350–359, 1965.

[5] F. Luccio, "Extending the Definition of Prime Compatibility Classes of States in Incomplete Sequential Machine Reduction," *IEEE Trans. on Computers*, vol. C-18, pp. 537–540, 1969.

[6] S. C. DeSarkar, A. K. Basu, and A. K. Choudhury, "Simplification of Incompletely Specified Flow Tables with the Help of Prime Closed Sets," *IEEE Trans. on Electronic Comptrs.*, pp. 953–956, 1969.

[7] S. C. D. Sarkar, A. K. Basu, and A. K. Choudhury, "On the Determination of Irredundant Prime Closed Sets," *IEEE Trans. on Computers*, pp. 933–938, 1971.

[8] J. Kella, "State Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. on Computers*, vol. C-19, no. 4, pp. 342–348, 1970.

[9] C.-C. Yang, "Closure Partition Method for Minimizing Incomplete Sequential Machines," *IEEE Trans. on Computers*, vol. C-22, pp. 1109–1122, December 1973.

[10] N. N. Biswas, "State Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. on Computers*, vol. C-23, no. 1, pp. 80–84, 1974.

[11] C. V. S. Rao and N. N. Biswas, "Minimization of Incompletely Specified Sequential Machine," *IEEE Trans. on Computers*, vol. C-24, no. 11, pp. 1089–1100, 1975.

[12] C. P. Pfleeger, "State Reduction in Incompletely Specified Finite-State Machines," *IEEE Trans. on Computers*, vol. C-22, pp. 1099–1102, December 1973.

[13] R. G. Bennetts, J. L. Washington, and D. W. Lewin, "A Computer Algorithm for State Table Reduction," *Radio and Electronic Engineer*, vol. 42, no. 11, pp. 513–520, 1972.

[14] M. A. Perkowski and N. Nguyen, "Minimization of Finite State Machines in System SuperPeg," *28th Midwest Conference on Circuits and Systems*, pp. 139–147, August 1985.

[15] S. F. Unger, *Asynchronous Sequential Switching Circuits*. Wiley Interscience, 1969.

[16] W. S. Scott, R. N. Mayo, G. Hamachi, and J. K. Ousterhout, "1986 VLSI Tools: Still More Works by the Original Artists," Tech. Rep. UCB/CSD 86/272, Computer Sciences Division (EECS), University of California, Berkeley, CA 94720, Dec 1985.

[17] R. Spickelmier, ed., *Oct Tools Distribution 2.1*. University of California, Berkeley: ERL, 1988.

[18] T. Villa and A. Sangiovanni-Vincentalli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *Proc. 26th ACM/IEEE Design Automation Conference*, pp. 327–332, 1989.

Name	Original machine				after FSMRED			
	i	o	p	s_o	s_m	s_c	p	time
bbara	4	2	60	10	7	7	42	2
bbtas	2	2	24	6	NCS			0.05
beecount	3	4	28	7	4	4	20	1
cse	7	7	91	16	NCS			0.73
dk14	3	5	56	7	NCS			0.12
dk15	3	5	32	4	NCS			0.1
dk16	2	3	108	27	NCS			2.06
dk17	2	3	32	8	NCS			0.1
dk27	1	2	14	7	NCS			0.07
dk512	1	3	30	15	NCS			0.25
donfile	2	1	96	24	ACS			-
ex2	2	2	72	19	4	10	40	3
ex4	6	9	21	14	NCS			0.17
ex5	2	2	32	9	2	4	16	1
ex6	5	8	34	8	NCS			0.1
ex7	2	2	36	10	3	4	16	1
keyb	7	3	170	19	NCS			1.97
kirkman	12	6	370	16	NCS			2.88
koha329	1	1	12	6	4	4	8	1
koha331-1	1	1	4	8	NCS			0.03
koha335	1	1	10	5	2	2	4	1
lion	2	1	11	4	NCS			0.02
mark1	5	16	22	15	12	12	31	1
mc	3	5	10	4	NCS			0.03
modulo12	1	1	24	12	ACS			-
opus	5	6	22	10	9	9	28	1
planet	7	9	115	48	NCS			15.9
s1	8	6	107	20	NCS			0.7
s8	10	1	20	5	ACS			-
sand	11	9	184	32	NCS			4.22
scf	27	56	166	121	97	97	238	998
shiftreg	1	1	16	8	NCS			0.05
sse	7	7	56	16	13	13	53	1
styr	9	10	166	30	NCS			4.1
tav	4	4	49	4	NCS			0.1
tbk	6	3	1569	32	16	16	784	35
train11	2	1	25	11	4	4	15	1
train4	2	1	14	4	NCS			0.05
ung41	2	1	36	9	2	5	20	1
ung46	1	1	12	6	3	3	6	1

Table 1: Performance of FSMRED with benchmark examples

Table 4. Performance of FSMRED and MUSTANG (multi-level)
* MISII failed to minimize even after many hours

Name	w/o FSMRED		w/FSMRED		% red
	time NOVA	area	time F+N	area	area
bbara	7	550	2+5	380	30.9
beecount	5	228	1+4	144	36.8
donfile	1357	700	3+0	0	100
ex2	18	609	3+10	342	43.8
ex5	17	252	1+4	84	66.67
ex7	6	306	1+4	108	64.7
koha329	4	72	1+4	45	37.5
koha335	4	72	1+4	18	75
modulo12	4	180	2+0	0	100
s8	5	180	1+0	0	100
tbk	3469	4620	35+92	1404	69.6
train11	15	153	1+4	66	56.9
ung41	7	289	1+5	112	61.2
ung46	4	96	1+4	45	53.1

Table 2: Performance of FSMRED and NOVA (2-level)

Name	w/o FSMRED		w/FSMRED		% red
	time	lit	time	lit	lit
bbara	23	69	17	50	27.5
beecount	10	55	9	37	32.7
donfile	1379	97	3	0	100
ex2	40	115	23	87	24.3
ex5	28	68	9	19	72
ex7	18	77	10	25	67.5
koha329	7	16	8	14	12.5
koha335	7	19	5	4	78.9
modulo12	11	29	2	0	100
s8	9	29	1	0	100
tbk	4730	506	341	278	45
train11	22	43	8	18	58.1
ung41	17	71	10	32	54.9
ung46	7	22	6	12	45.5

Table 3: Performance of FSMRED and NOVA (multi-level)

Name	w/o FSMRED		w/FSMRED		% red
	time	lit	time	lit	lit
bbara	52	125	23	62	50.4
beecount	34	106	18	62	41.5
donfile	464	297	3	0	100
ex2	292	221	141	130	41.2
ex5	145	106	21	30	71.7
ex7	35	80	16	35	56.3
koha329	12	30	7	19	36.7
koha335	9	25	4	7	72
modulo12	15	88	2	0	100
s8	17	111	1	0	100
tbk	-*	-*	2362	436	-
train11	85	80	19	32	60
ung41	29	95	25	45	52.6
ung46	15	30	6	14	53.3