Pseudo-dichotomies and its use in simultaneous state minimization and state assignment

Ney Laert Vilar Calazans

Abstract: The objective of this work is to introduce a discussion about the simultaneous state minimization and state assignment of finite state machines. In this work, the pseudo-dichotomy concept is evaluated in its capacity to provide a unified framework, useful to represent the set of constraints corresponding to each of the problems. The discussion is based on the development of examples.

I - Introduction

The automatic synthesis of digital systems, as it is performed today requires the extensive use of hierarchical decomposition of the design process into abstraction levels. The logic level is intermediate between what is now known as high-level synthesis (either behavioural or structural) and low-level (or layout) synthesis. The logic level plays an important role in the synthesis of finite state machines (FSMs), a widespread and general means to implement many of the functionalities of modern digital circuits.

It is in the logic level that some of the harder problems associated with the synthesis of FSMs appear, such as state assignment, state minimization and combinational logic minimization. The first and the last of these problems have received continued attention for a long time, and are still regarded as not satisfactorily solved questions, even though some methods have appeared that solve restricted versions of them [Bra84] [Bra87] [Dev88] [Sau87] [Vil89]. On the other hand, the state minimization of FSMs experimented a period of effervescence in the 60's, when the basic theory was set up [Pau59] [Gra65], its popularity declining in the next two decades. One proof of this is the nonexistence of state minimization programs in most commercial and academic FSMs synthesis systems. It seems that the designers of these systems simply took for granted that the generation of FSM specifications without redundant states was easy. Quite recently, some works have shown [Ave90] [Hac91] [Kan91] that this is not the case. Some well known machines, used as benchmarks for state assignment synthesis simply turned out to be composed of only one compatibility class comprising all states in the machine. This implies that the functionality of the machine can be implemented by a single combinational circuit. Most modern synthesis systems, when trying to implement the specification of these machines produce, at the end, a complete sequential circuit, with a combinational part implementing the transition and output functions and a register to store the present state. This lack of insight of present systems results in waste of precious semiconductor surface.

The idea of this work is to propose the integration of state minimization procedures in the ordinary design process for the logic level of abstraction. Also, we intend to investigate the relationship between state minimization and state assignment of FSMs. Intuitively, we know that these two synthesis steps are not independent, and too strong a state minimization can actually deteriorate the performance level accessible to the best available state assignment technique. Since both state minimization and assignment rely upon the satisfaction of constraints extracted from the machine description, we intend to formulate both problems within a unified framework. One starting point are the recently introduced symbolic minimization methods [dMi85] [dMi86], that allow a quick estimation of the potential results of a good assignment to be obtained. On the other hand, the introduction of the concept of pseudo-dichotomies [Yan91] [Sal91], also called simply dichotomies, provides us with a first tool to approach both problems using a single formulation, as it will be shown here.

In the next section we provide a case study of synthesis of FSM. At the end of the case study, a discussion about the feasibility of using pseudo-dichotomies for the simultaneous solution of state minimization and assignment is presented. The term definitions needed to follow the exposition below can be found in [dMi85], [Gra65] and [Yan91].

Solving the state reduction and assignment problems concurrently is not a totally new idea [Lee84][Ave91], but the formulation used in previous works is both incomplete and fragmentary.

II - Case Study

The example we chose is an FSM called **lion9**. This machine is part of the benchmark set of the Octtools synthesis system. It is described in the **kiss** format, where inputs and outputs are already binary coded, while states are represented symbolically. We first give the state transition table in Table 1 below. The machine has 2 (binary) inputs, 1 (binary) output and 9 states. This machine presents a particularity in its structure that makes it useful for a first assessment of the problems, it contains no conditionally compatible states. This means that no non-trivial closure constraint is imposed on the state minimization phase.

lion9	00	01	10	11
st0	st0,0	-,-	st1,0	-,-
st1	st0,0	-,-	st1,0	st2,0
st2	-,-	st3,0	st1,0	st2,0
st3	st4,1	st3,1	-,-	st2,1
st4	st4,1	st3,1	st5,1	-,-
st5	st4,1	-,-	st5,1	st6,1
st6	-,-	st7,1	st5,1	st6,1
st7	st8,1	st7,1	-,-	st6,1
st8	st8,1	st7,1	-,-	-,-

Table 1 - Symbolic state transition table for machine **lion9**

In order to implement this machine efficiently, we devised a procedure capable of generating a good state assignment, while at the same time reducing the number of states. This procedure is based upon the method proposed in [Yan91]. In fact, we just imagined to relax the injectivity constraints required by this method (that says, for short, that distinct states should correspond to distinct codes), so as to allow the coding of all states belonging to a single compatibility class with identical or compatible codes. This is of course equivalent to doing simultaneous state reduction and state assignment, at least for the class of machines we address now, i. e. those with no closure constraints

To illustrate the procedure, we are going to apply it to the **lion9** machine. We start from the **kiss** description, presented in Figure 1 below. This description is to be used as the source to the first step of the method, i. e. the generation of the input constraints. Note that we will restrict ourselves to the state assignment based on input encoding techniques only. Nonetheless the method has already been extended in [Cie91] to deal with both, input and output encoding constraints. The restriction we impose here is done to keep the clarity of the exposition, since the generalization of the procedure should pose no major problem.

In order to generate the input constraints, we must perform the symbolic minimization of the cover in Figure 1. To do so, we translate the description **kiss** into a symbolic multiple-valued cover. The inputs remain in their original binary format, but states are transformed into an equivalent one-hot encoding [dMi85], the same transformation being performed on the outputs, to allow for output-disjoint minimization. The resulting description is shown in Figure 2. Note that states are now associated to columns in the symbolic cover, **st0** to the first column, **st1** to the second one and so on. The symbolic cover is submitted, then, to a multiple-valued minimizer. In our case, we used **Espresso-MV** [Rud85], a tool from the already mentioned Octtools package. The resulting minimized symbolic cover is presented in Figure 3. This result is in fact an upper bound for the final solution. At least one solution of the encoding problem exists where the number of cubes of a PLA implementation is exactly the same as the number of cubes in the final symbolic cover. In our case, the final minimized symbolic cover have 11 cubes.

```
.i 2
.o 1
.p 25
.s 9
10 st0 st1 0
00 st0 st0 0
00 st1 st0 0
10 st1 st1 0
11 st1 st2 0
10 st2 st1 0
11 st2 st2 0
01 st2 st3 0
11 st3 st2 1
01 st3 st3 1
00 st3 st4 1
01 st4 st3 1
00 st4 st4 1
10 st4 st5 1
00 st5 st4 1
10 st5 st5 1
11 st5 st6 1
10 st6 st5 1
11 st6 st6 1
01 st6 st7 1
11 st7 st6 1
01 st7 st7 1
00 st7 st8 1
01 st8 st7 1
00 st8 st8 1
.e
```

Figure 1 - The **kiss** description of **lion9**

The second column of the cover in Figure 3 corresponds to the input constraints we were looking for. Let us isolate this column, generating the input constraints matrix $\mathbf{M^R}$. This matrix is shown in Figure 4 with the associated seed dichotomies. For simplicity, the states are represented from now on as single integers (**st0** is **0**, **st1** is **1**, and so on). Note that the reduced input constraints matrix mentioned in [Yan91], $\mathbf{M_S{}^R}$ is identical to $\mathbf{M^R}$, since neither duplicated rows nor rows containing only 1s are present. At this point, the differences between the approach of [Yan91] and ours begin to appear. In [Yan91], while generating the seed dichotomies, any dichotomy covered by a dichotomy previously created is stripped from the final set. We strip only repeated dichotomies, since our approach is to reduce this first set by eliminating some chosen constraints. Thus, we cannot strip covered dichotomies in this step, since this could lead to inconsistencies in the final set of seed dichotomies.

```
.mv 5 2 9 9 2
.kiss
.p 25
```

```
10  100000000  010000000  10
00  100000000  100000000  10
00  010000000  100000000  10
10  010000000  010000000  10
11  010000000  001000000  10
10  001000000  010000000  10
11  001000000  001000000  10
01  001000000  000100000  10
11  000100000  001000000  01
01  000100000  000100000  01
00  000100000  000010000  01
01  000010000  000100000  01
00  000010000  000010000  01
10  000010000  000001000  01
00  000001000  000010000  01
10  000001000  000001000  01
11  000001000  000000100  01
10  000000100  000001000  01
11  000000100  000000100  01
01  000000100  000000010  01
11  000000010  000000100  01
01  000000010  000000010  01
00  000000010  000000001  01
01  000000001  000000010  01
00  000000001  000000001  01
. e
```

Figure 2 - The mixed binary/multiple-valued description for **lion9**

```
.mv  4  2  9  9  2
.p  11
00  110000000  100000000  10
01  001000000  000100000  10
10  111000000  010000000  10
11  011000000  001000000  10
11  000100000  001000000  01
00  000000011  000000001  01
01  000110000  000100000  01
10  000011100  000001000  01
00  000111000  000010000  01
11  000001110  000000100  01
01  000000111  000000010  01
. e
```

Figure 3 - The minimized mixed binary/multiple-valued description for **lion9**

The next step in the Yang and Ciesielski procedure would be to add the seed dichotomies to guarantee the injectivity of the encoding mapping, in case any is needed. Here resides the fundamental change we propose. Instead of adding the injectivity constraints, we propose first to make a state-pair compatibility analysis for the machine. The result of this analysis is depicted in Figure 5. Note here that this machine has no conditionally compatible state pair, as stated before. This implies that any set of compatibles covering the initial set of states is acceptable as a solution of the state minimization problem. Closure constraints are, in general harder to satisfy than the covering ones, so we decide to treat first the special case where they do not exist or are trivial.

M^R	**Seed Dichotomies**
110000000	**- (01:2) (01:3) (01:4) (01:5) (01:6) (01:7) (01:8)**

001000000	**- (2:0) (2:1) (2:3) (2:4) (2:5) (2:6) (2:7) (2:8)**
111000000	**- (012:3) (012:4) (012:5) (012:6) (012:7) (012:8)**
011000000	**- (12:0) (12:3) (12:4) (12:5) (12:6) (12:7) (12:8)**
000100000	**- (3:0) (3:1) (3:2) (3:4) (3:5) (3:6) (3:7) (3:8)**
000000011	**- (78:0) (78:1) (78:2) (78:3) (78:4) (78:5) (78:6)**
000110000	**- (34:0) (34:1) (34:2) (34:5) (34:6) (34:7) (34:8)**
000011100	**- (456:0) (456:1) (456:2) (456:3) (456:7) (456:8)**
000111000	**- (345:0) (345:1) (345:2) (345:6) (345:7) (345:8)**
000001110	**- (567:0) (567:1) (567:2) (567:3) (567:4) (567:8)**
000000111	**- (678:0) (678:1) (678:2) (678:3) (678:4) (678:5)**

Figure 4 - The input constraints matrix M^R and the seed dichotomies for machine **lion9**

Figure 6 shows the merge graph for this example, from which the maximal compatibles (represented by the maximal cliques in the graph) can be easily extracted by inspection.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	**-**							
2	**-**	**-**						
3	**X**	**X**	**X**					
4	**X**	**X**	**X**	**-**				
5	**X**	**X**	**X**	**X**	**-**			
6	**X**	**X**	**X**	**X**	**X**	**-**		
7	**X**	**X**	**X**	**X**	**X**	**X**	**-**	
8	**X**	**X**	**X**	**X**	**X**	**X**	**-**	**-**

Figure 5 - The state pairs compatibility analysis for machine **lion9**

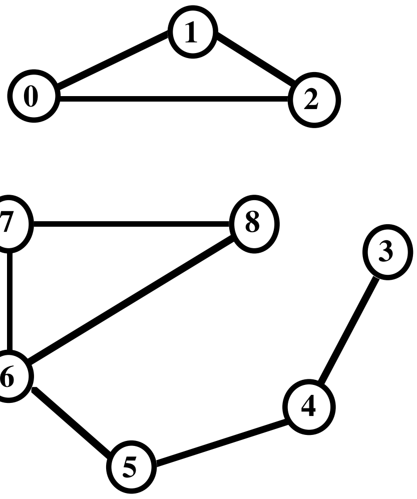

Figure 6 - The merge graph for machine **lion9**

Our procedure is the following. Associate with each pair of states one elementary dichotomy, in the obvious way. Each dichotomy corresponding to a pair of incompatible states is to be added to the set of seed dichotomies coming from the input coding constraints, in case it is not already covered by any of the seed dichotomies. This amounts to impose that distinct (or incompatible) codes are to be assigned to the states of an incompatible pair of states. On the other hand, each dichotomy in the present set of seed dichotomies that has elements of a compatible pair in different blocks (in other words, each dichotomy that covers some elementary dichotomy generated by a pair of compatibles) is to be transformed, so as to eliminate the part of the dichotomy that prevents the assignment of identical (or compatible) codes to states belonging to a single class of compatibility. The set of dichotomies created by the compatible pairs of states are thus **forbidden** dichotomies. In our example, no elementary dichotomy is added, since every elementary dichotomy coming from pairs of incompatible states is already covered by some seed dichotomy. On the other hand, some seed dichotomies must be replaced. The forbidden elementary dichotomies, together with the set of seed dichotomies resulting from the replacement process described above are depicted in Figure 7. Replaced dichotomies are marked with *. In some cases, the dichotomy vanishes completely, since no constraint remains after eliminating the forbidden part.

Finally, we eliminate those seed dichotomies covered by others in the set. The resulting set of constraints to be satisfied by the implementation is shown in Figure 8. Note that the order of the operations described here is not the only one possible. In fact, if the forbidden dichotomies are generated in the first place, there is no need to store the whole set of seed dichotomies, making a practical application more efficient in terms of memory space.

Forbidden Dichotomies: (0:1) (0:2) (1:2) (3:4) (4:5) (5:6) (6:7) (6:8) (7:8)

M^R		Seed Dichotomies
110000000	**-**	**(01:3) (01:4) (01:5) (01:6) (01:7) (01:8)**
001000000	**-**	**(2:3) (2:4) (2:5) (2:6) (2:7) (2:8)**
111000000	**-**	**(012:3) (012:4) (012:5) (012:6) (012:7) (012:8)**
011000000	**-**	**(12:3) (12:4) (12:5) (12:6) (12:7) (12:8)**
000100000	**-**	**(3:0) (3:1) (3:2) (3:5) (3:6) (3:7) (3:8)**
000000011	**-**	**(78:0) (78:1) (78:2) (78:3) (78:4) (78:5) (78:6)**
000110000	**-**	**(34:0) (34:1) (34:2) (3:5)* (34:6) (34:7) (34:8)**
000011100	**-**	**(456:0) (456:1) (456:2) (56:3)* (45:7)* (45:8)***
000111000	**-**	**(345:0) (345:1) (345:2) (34:6)* (345:7) (345:8)**
000001110	**-**	**(567:0) (567:1) (567:2) (567:3) (67:4)* (5:8)***
000000111	**-**	**(678:0) (678:1) (678:2) (678:3) (678:4) (78:5)***

Figure 7 - The forbidden dichotomies and the seed dichotomies set after elimination for machine **lion9**

M^R		Seed Dichotomies
110000000	**-**	
001000000	**-**	
111000000	**-**	**(012:3) (012:4) (012:5) (012:6) (012:7) (012:8)**
011000000	**-**	
000100000	**-**	**(3:5)**
000000011	**-**	
000110000	**-**	**(34:6)**
000011100	**-**	**(456:0) (456:1) (456:2)**
000111000	**-**	**(345:0) (345:1) (345:2) (345:7) (345:8)**
000001110	**-**	**(567:0) (567:1) (567:2) (567:3)**
000000111	**-**	**(678:0) (678:1) (678:2) (678:3) (678:4)**

Figure 8 - The final set of seed dichotomies after eliminating covered dichotomies for machine **lion9**

From this last set of dichotomies we have to arrive to the coding of states in the best possible way. The graph colouring approach can be used, but more powerful cost functions must be developed for evaluating the quality of the result, since we have more degrees of freedom for generating the final encoding. For this particular example, we know that the least number of states possible is four, and the least number of encoding bits is thus two. A first solution can be obtained by using the knowledge we have from the problem. We know the maximal compatibility classes and we can force the merging of dichotomies to be as balanced (according to [Yan91]) as possible. One solution for the minimum covering of the set of seed dichotomies is presented in Figure 9, with the corresponding assignments shown.

Final Dichotomies: **(01234:5678)**
(01278:345)
(456:012)

Encodings

Initial

St	Code
0	001
1	001
2	001
3	01-
4	010
5	110
6	1-0
7	10-
8	10-

Final

St	Code
{0, 1, 2}	001
{3, 4}	010
{5, 6}	110
{6, 7, 8}	100

Figure 9 - The solution of the seed dichotomy set covering for machine **lion9**

Note that the first encoding is partial, allowing don't-cares to appear in the code of some states. These don't-cares can then be used to arrive to a final solution where the compatibles are not necessarily disjoint, i. e. the general case for traditional state minimization procedures. The cover is obviously closed, due to the nature of the example, as discussed before. Note also that in the final solution, the last bit of the code can be discarded completely, since it is not needed to distinguish between any pair of states. This arises because the two last pseudo-dichotomies are compatible, and can thus be merged. This was not done here, so as to illustrate the possibility of having don't cares in the first solution of the encoding.

III - Discussion

We have seen that pseudo-dichotomies provide an easy way to unify state assignment and state minimization for a special class of machines, those without closure constraints. An important question is: can extend the techniques above to deal also with closure constraints? In other words could we translate closure constraints into dichotomies? Our experience showed that this is not an easy task. The explanation resides in the mathematical weaknesses of the pseudo-dichotomy concept. The concept cannot be mapped directly to any well known algebraic structure, even though pseudo-dichotomies emanate directly from the mathematically well behaved partition concept. The sum of dichotomies is defined only if special conditions hold on the members of the structure, the same being true of the product of dichotomies. In the general case, neither supremum nor infimum elements can be defined for a pair of dichotomies.

When we merge dichotomies, we automatically lose some degrees of freedom. For example, suppose we have two dichotomies **(12:34)** and **(15:36)**. If we merge them, we have **(125:346)**, which has the additional seed dichotomies **(4:5)** and **(2:6)**, neither of them present in any of the original dichotomies. In our case study, we can see that the initial set of dichotomies does not allow the merging of states 4 and 5 in a single state, since their codes are incompatible. Nonetheless, we know from the compatibility analysis that these states do form a compatible. If dichotomies are to be retained as a general framework, better algorithms for the dichotomy merging have to be found.

Another problem is how to proceed systematically from the final set of seed dichotomies until finding the most adequate set of merged dichotomies. The cost functions to minimize must be carefully studied, since they must lead to:

a) <u>the least number of distinct codes</u> - meaning greatest state minimization;

b) <u>the code the most sparse</u> - meaning the greatest possibility of minimization of the final combinational part and use of state splitting techniques;

c) <u>the code with the minimum length in bits</u> - meaning the least number of outputs for the combinational part of the machine.

To allow c) above to arise, we need to minimize the cardinality of the final set of merged dichotomies. To allow b) above to be attained, the dichotomies must be the smaller the possible, which in fact conflicts indirectly with the requirement of c). Finally, to easy the fulfilment of a), compatible states must be the more frequently the possible in the same block of the dichotomies. Also, the last desirable situation which favours b) to happen can be seen as a secondary goal to obtain compatible codes for sets of compatible states, since this creates more possibilities for merging codes (due to the arising of don't-cares).

Every time states are not coded directly as equal values but present, nonetheless compatible codes, we can merge them. For example, state 6 code in the case study was "customized" to belong to more than one compatibility class.

Due to all the above results in trying to use dichotomies for studying the correlation between state minimization and assignment, we are presently investigating alternatives to the unified framework. The most promising way, up to now has been the application of Boolean equations to model all of the constraints associated with both problems.

IV - Bibliography

[Ave90] M. J. Avedillo, J. M. Quintana, J. L. Huertas, "State reduction of incompletely specified finite sequential machines." In: *Proceedings of the IFIP Working Conference on Logic and Architecture Synthesis*, pp. 107-115, Paris, 30 May - 1 Jun, 1990.

[Ave91] M. J. Avedillo, J. M. Quintana, J. L. Huertas, "SMAS: a program for concurrent state reduction and state assignment of finite state machines." In: *Proceedings of the IEEE International Symposium on Circuits and Systems - ISCAS*, vol. 3, pp. 1781-1784, Singapore, 11-14 Jun, 1991.

[Bra84] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Higham, MA: Kluwer Academic, 1984.

[Bra87] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple level logic optimization system," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 1062-1081, Nov. 1987.

[Cie91] M. J. Ciesielski, J.-J. Shen, and M. Davio, "A unified approach to input-output encoding for FSM state assignment." In: *The Proceedings of the 28th Design Automation Conference - DAC*, pp. 176-181, Jun, 1991.

[Dev88] S. Devadas, H. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations." *IEEE Transactions on Computer-Aided Design*, vol. 7, pp. 1290-1299, Dec. 1988.

[dMi85] G. de Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, pp. 269-284, Jul. 1985.

[dMi86] G. de Micheli, “Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros,” *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, pp. 597-616, Oct. 1986.

[Gra65] A. Grasselli, and F. Luccio, “A method for minimizing the number of internal states in incompletely specified sequential networks,” *IRE Transactions on Electronic Computers,* vol. EC-14, pp. 350-359, Jun. 1965.

[Hac91] G. D. Hachtel, J.-K. Rho, and F. Somenzi, “Exact and heuristic algorithms for the minimization of incompletely specified state machines.” In: *The Proceedings of the European Conference on Design Automation - EDAC*, pp. 192-196, Amsterdam, 184-191 Feb, 1991.

[Kan91] L. N. Kannan, and D. Sarma, “Fast heuristic algorithms for finite state machine minimization.” In: *The Proceedings of the European Conference on Design Automation - EDAC*, pp. 192-196, Amsterdam, 25-28 Feb, 1991.

[Lee84] E. B. Lee and M. Perkowski. "Concurrent Minimization and State Assignment of Finite State Machines". In *The Proceedings of The 1984 International Conference on Systems Man and Cybernetics,* Halifax, Oct. 1984.

[Pau59] M. C. Paull, and S. H. Unger, “Minimizing the number of states in incompletely specified sequential switching functions,” *IRE Transactions on Electronic Computers,* vol. EC-8, pp. 356-367, Sep. 1959.

[Rud85] R. Rudell and, A. L. M. Sangiovanni-Vincentelli, “Espresso-MV: algorithms for multiple-valued logic minimization," In: *IEEE 1985 Custom Integrated Circuits Conference,* pp. 230-234, June 1985.

[Sal91] A. Saldanha, T. Villa, R. K. Brayton, and A. S.-Vincentelli, “A framework for satisfying input and output encoding constraints.” In: *The Proceedings of the 28th Design Automation Conference - DAC*, pp. 170-175, Jun, 1991.

[Sau87] G. Saucier, M. C. de Paulet, and P. Sicard, “ASYL: A rule-based system for controller synthesis,” *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 1088-1097, Nov. 1987.

[Vil89] T. Villa, and A. S.- Vincentelli. "NOVA: state assignment of finite state machines for optimal two-level implementations". In: *The Proceedings of the 26th Design Automation Conference - DAC,* Jun-Jul 1989.

[Yan91] S. Yang, and M. Ciesielski, “Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization,” *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 4-12, Jan. 1991.