Implicit Prime Compatible Generation for Minimizing Incompletely Specified Finite State Machines

Hiroyuki HIGUCHI Yusuke MATSUNAGA

CAD Laboratory
FUJITSU LABS. LTD.
Kawasaki, Japan 211
Tel: +81-44-754-2663
Fax: +81-44-754-2664
e-mail: {higuchi, yusuke}@flab.fujitsu.co.jp

Abstract— This paper proposes a new implicit algorithm for excluding dominated compatibles. The algorithm utilizes a novel notion of signatures of compatibles to exclude dominated compatibles efficiently. Though this dominance check is weaker than the conventional one, experimental results show that in many cases the number of excluded compatibles is the same as that by class sets. Proposed method computes prime compatibles more efficiently than conventional methods for many tested large ISFSM's.

I. Introduction

Minimizing the number of states in incompletely specified finite state machines(ISFSM's) is an important step of FSM synthesis. A combination of the methods of Paull and Unger [1] and Grasselli and Luccio [2] provides a systematic method for finding a minimum-row state table for an ISFSM. The approach reduces the problem to the computation of prime compatibles and the selection of a minimum closed set of them by means of a binate covering step. If the machine has n states, the number of compatibles is $O(2^n)$. Prime compatibles are selected out of these compatibles. Therefore the computation of prime compatibles for large ISFSM's will be intractable with explicit enumeration of compatibles in [1, 2, 3].

Implicit techniques are based on the idea of operating on discrete sets by their characteristic functions represented by Binary Decision Diagrams (BDD's) [4]. For two-level logic minimization, efficient implicit algorithms have been proposed [5, 6, 7]. To minimize the number of states in large ISFSM's, Kam et al. proposed a fully implicit algorithm with Binary Decision Diagrams (BDD's) [8]. They showed that the implicit method can handle some ISFSM's which the explicit methods cannot minimize. In this paper, we propose a new implicit method for generating prime compatibles. We introduce a novel notion of signatures of compatibles in order to efficiently exclude dominated compatibles. The signatures of a compatible correspond to the minimal compatible dominated by the compatible. Therefore compatibles which cover at least one of these signatures are to be excluded. Furthermore, BDD's of relation between compatibles and their signatures are likely to be represented efficiently. By using these fact, dominated compatibles are likely to be excluded efficiently. Though this dominance check is weaker than that with class sets, experimental results shows that in many cases the number of excluded compatibles are the same as that by class sets. Our method takes less time than conventional implicit and explicit methods for many tested large ISFSM's.

The rest of this paper is organized as follows. Section 2 gives terminologies and theorems proved in previous papers. Implicit representations of compatibles are also introduced. Section 3 gives an algorithm of generating compatibles. Algorithms for implicit generation of prime compatibles are presented in Section 4. Experimental results are shown in Section 5. Conclusions are summarized in Section 6.

II. Preliminaries

A. Definitions

Definition 1 *A Finite State Machine (***FSM***) M is a 5-tuple $M = (I, O, S, \delta, \lambda)$, 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, where each (input, state) pair is related to exactly one next state and one output, is a **completely specified FSM.** *An* **incompletely specified FSM** *is one where either the next state or the output is not specified for at least one (input, state) pair.* □

Definition 2 *Two output sequences $\tilde{z}_a, \tilde{z}_b$ of machine M are* **compatible** *if and only if their corresponding outputs are not conflicting, i.e., identical whenever both outputs are specified.* □

We denote compatible output sequences $\tilde{z}_a, \tilde{z}_b$ by $\tilde{z}_a \sim \tilde{z}_b$.

Definition 3 *Two states s_a, s_b of machine M are* **compatible** *if and only if s_a, s_b yield compatible output sequences for every input sequence. Two states s_a, s_b are* **incompatible** *if they are not compatible.* □

PS	NS, z	
	$x=0$	$x=1$
A	C, 1	E, –
B	C, –	E, 1
C	B, 0	A, 1
D	D, 0	E, 1
E	D, 1	A, 0

Fig. 1. Machine M

We denote compatible states s_a, s_b by $s_a \sim s_b$.

Theorem 1 *Two states s_a, s_b of M are compatible if and only if*

(i) $\forall x \in I,\ \lambda(x, s_a) \sim \lambda(x, s_b)$, *and*

(ii) $\forall x \in I,\ \delta(x, s_a) \sim \delta(x, s_b)$. □

Let us introduce an example machine M, which will be used throughout the paper. The state table to be minimized is shown in Fig.1. Since next states of states A, B of M are identical and outputs of A, B are not conflicting, States A, B of M are compatible. States A, C are incompatible, because next state of A, C for input 0 are conflicting.

Definition 4 *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, state pairs (B, C), (C, D), and (B, D) of M are compatible. Therefore set $\{B, C, D\}$ is compatible.

Definition 5 Implied set *$IMP(c, i)$ of compatible c for input i is the following set of states:*

$$IMP(c, i) = \bigcup_{s \in c} \delta(s, i).$$

□

The implied set of compatible c for input i is the set of next states of states in c for input i. For example, the implied set of compatible $\{B, C, D\}$ for input 1 is $\{A, E\}$.

Definition 6 *Next state pair $(\delta(s_a, i), \delta(s_b, i))$ of compatible state pair (s_a, s_b) for input i is called the* **implied pair** *of (s_a, s_b) for input i.* □

In this paper, a pair is also treated as a set with two elements.

Definition 7 Class set *$CS(c)$ of compatible c is the set of all the implied sets d such that:*

(i) $|d| > 1$, *and*

(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 $\{B, C, D\}$ is $\{\{A, E\}\}$. Since the implied set of the compatible for input 0, which is $\{B, C, D\}$, is the compatible itself, it is not a member of the class set.

Definition 8 *Compatible c* **dominates** *compatible c' if and only if*

1. $c' \subset c$, *and*
2. $CS(c') \supseteq CS(c)$. □

Compatible c expresses strictly weaker conditions than compatible c'. Therefore c is always a better choice for a minimum closed cover than c'.

Definition 9 *A* **prime compatible** *is one that is not dominated by any other compatibles.* □

Theorem 2 *At least one minimum closed cover consists entirely of prime compatibles [2].* □

The theorem shows that it is sufficient to consider a subset of compatibles called prime compatibles for the selection of one minimum closed set.

B. Implicit Representations of Compatibles

In state minimization, 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(BDD's). The representation of compatibles is the same as in [8].

Suppose an FSM has n states, there are 2^n possible sets of states. In order to represent them at once, each set is represented in **positional-set** form by using a set of n Boolean variables, $\vec{x} = (x_1, x_2, \cdots, x_n)$. Variable x_k takes the value 1 in the positional set if state s_k is a member of the set, while x_k takes the value 0 if state s_k is not a member. For example, if $n = 5$, the set $\{s_1, s_4\}$ is represented by 10010.

A set of sets of states is represented as a set S of positional-sets by a characteristic function $\chi_S : B^n \rightarrow B$. $\chi_S(\vec{x}) = 1$ 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.

III. Generation of Compatibles

The procedure of computing prime compatibles can be divided into two parts:

(i) generation of compatibles,

(ii) deletion of dominated compatibles.

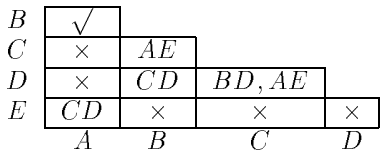

Fig. 2. Merger table

generate_compatible($MergerTable\ T$) {
1. $C \leftarrow Universe \backslash \{\emptyset\}$;
2. **for** each incompatible pair p in T {
3. $C \leftarrow C \backslash$(all the sets including p);
4. } /* C is the set of all the compatibles */
5. **return** C;
}

Fig. 3. Generation of compatibles

In this section, we discuss the first step. The second step will be discussed in the next section.

Here let us take the example machine M again. To generate all the compatibles, the *merger table* [9] is derived first. The merger table is also called the implication table [1] or compatibility table [2]. The merger table of a machine has cells corresponding to the compatible pairs defined by the intersection of the row and column headings. The incompatibility of two states is recorded by placing an "×" in the corresponding cell, while their compatibility is recorded by a check mark ($\surd$). The entries in cell s_a, s_b are the implied pairs of (s_a, s_b). The merger table is explicitly constructed and represented in our method, which is the same as in the method of Paull and Unger [1]. In [8], this step is also done implicitly by image computation technique. Since the number of cells in the merger table is at most $n(n-1)/2$, where n is the number of states, and also image computation requires large computation efforts, the explicit method is likely to be faster than the implicit one.

The merger table for the example is shown in Fig.2. By using the merger table we generate all the compatibles implicitly. The algorithm is shown in Fig.3. For the sake of simplicity, the algorithms introduced in the sequel will be described on explicit sets of sets of states. These algorithms can be directly translated into implicit manipulation.

Since incompatibles are the sets containing at least one incompatible pair, all the compatibles C 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.

Machine M has 11 compatibles:

$$C = \{(E),(D),(C),(CD),(B),(BD),$$
$$(BC),(BCD),(A),(AE),(AB)\},$$

which is also written in the positional-set form as:

$$C = \{00001, 00010, 00100, 00110, 01000, 01010,$$
$$01100, 01110, 10000, 10001, 11000\}.$$

IV. Implicit Generation of Prime Compatibles

Prime compatibles are generated by excluding dominated compatibles. To exclude dominated compatibles efficiently in implicit manners , we use input-labeled pair-wise class sets instead of class sets.

A. Input-Labeled Pair-Wise Class Sets

Definition 10 *If (s_a, s_b) is an implied pair of (s_c, s_d) for input i, then (s_a, s_b, i) is called an* **input-labeled implied pair** *of (s_c, s_d).* □

Definition 11 Input-labeled pair-wise class set *$IPWCS(c)$ of compatible c is the set of all the input-labeled implied pairs (s_a, s_b, i) of any state pairs in c such that:*

(i) $s_a \neq s_b$, and

(ii) $\{s_a, s_b\} \not\subseteq c$. □

For example, let us consider compatible $\{B,C,D\}$ of machine M. Since the input-labeled implied pairs in the compatible are only $(A, E, 1)$, the input-labeled pair-wise class set is $\{(A, E, 1)\}$.

Intuitively a input-labeled pair-wise class set is the collection of the state pairs attached to input i in each implied sets in the class set for each input i.

We introduce the following dominance relation using input-labeled class sets instead of the conventional dominance relation.

Definition 12 *Compatible c* **pw-dominates***(or pair-wise-dominates) compatible c' if and only if*

1. $c' \subset c$, and

2. $IPWCS(c') \supseteq IPWCS(c)$. □

Definition 13 Signature *s of compatible c is a minimal subset of c such that $IPWCS(c) \subseteq IPWCS(s)$.* □

A signature of compatible c is a minimal compatible pw-dominated by c. Signatures of compatibles are obtained by collecting the state pairs implying each implied pair in pair-wise class sets. Though we could compute signatures for class sets, it requires reverse image computation, which takes a lot of time.

Definition 14 *A* **pw-prime compatible** *is one that is not pw-dominated by any other compatibles.* □

As for the relation between pw-dominance and the conventional dominance, following condition holds.

Theorem 3 *Compatible c is a pw-prime compatible if c is a prime compatible.* □

Proof. Suppose that there exists a prime compatible c such that c is pw-dominated by another compatible c'. Then (1) $c \subset c'$ and (2) $IPWCS(c) \supseteq IPWCS(c')$. Condition (2) means that $IMP(c,x) \supseteq IMP(c',x)$ for every input x. It follows that $CS(c) \supseteq CS(c')$. Therefore c is dominated by c', which is a contradiction. □

The theorem shows that dominance check by IPWCS's is weaker than that by class sets. That is, if a compatible is excluded by pw-dominance checking, it is also excluded by the conventional dominance checking.

From theorem2 and 3, the following theorem is derived.

Theorem 4 *At least one minimum closed cover consists entirely of pw-prime compatibles.* □

B. Example

To explain how to exclude pw-dominated compatibles, let us consider the example in Fig.1. In Section III, all the compatibles of the machine are computed. First we compute the signatures of compatibles. For each compatible, all the subsets of the compatible are generated and the subsets are coupled with the compatible as shown in the leftmost column of Fig.4. Note that compatibles and their signatures are represented in the positional-set form. Then pick up an input-labeled implied pair, for example, $(C,D,0)$. (C,D) is an implied pair of (A,E) for input 0. It is also an implied pair of (B,D) for input 0. It is written in the topmost row in Fig.4. Here (s_1,s_2,i) denotes an implied pair (s_1,s_2) for input i.

For each compatible such that

$$\text{“}c \not\supseteq (C,D)\text{” and “}c \text{ contain } (A,E) \text{ or } (B,D)\text{”}, \quad (1)$$

all the subsets of c containing (A,E) or (B,D) are chosen. Other subsets of c are dropped and "×"'s are placed in corresponding cells(shown in the column p_1). Any subsets of compatibles that do not satisfy (1) also still remain. This procedure is done for all the input-labeled implied pairs one by one. When it is finished, the remaining subsets for each compatible are ones pw-dominated by the compatible. Minimal subsets are signatures. In Fig.4, signatures of each compatible are checked in the rightmost column "sig.". Finally pw-dominated compatibles are excluded. In the example, the pw-prime compatibles are 00001, 00010, 00100, 01010, 01110, 10001, 11000. These are the same as the conventional prime compatibles.

These calculation is suitable for implicit calculation. In the next subsection, an implicit procedure is shown.

C. Implicit Generation of Prime Compatibles

The implicit algorithm for generating the pw-prime compatibles is shown in Fig.5.

All the operations in *gen_prime_compatible*$(C,\ T)$ is done by BDD manipulation. The set C of all the compatibles is represented by a BDD as described in subsection

compatible	subset	implied pair † p_1	p_2	p_3	sig.
00001	00001				
	00000				√
00010	00010				
	00000				√
00100	00100				
	00000				√
00110	00110				√
	00100		×		
	01000		×		
	00000		×		
01000	01000				
	00000				√
01010	01010				√
	01000	×			
	00010	×			
	00000	×			
01100	01100				√
	01000		×		
	00100		×		
	00000		×		
01110	01110				
	01100				√
	01010		×		
	00110				√
	01000		×		
	00100		×		
	00010		×		
	00000		×		
10000	10000				
	00000				√
10001	10001				√
	10000	×			
	00001	×			
	00000	×			
11000	11000				
	10000				
	01000				
	00000				√

† p_1 : $(CD,0) \leftarrow AE, BD$
p_2 : $(AE,1) \leftarrow BC, CD$
p_3 : $(BD,0) \leftarrow CD$

Fig. 4. Generation of signatures

B. To generate pw-prime compatibles, pairs of compatibles and their signatures are represented as a BDD of $2n$ variables, $\vec{x} = (x_1, x_2, \cdots, x_n)$ and $\vec{x'} = (x'_1, x'_2, \cdots, x'_n)$. Since x_i and x'_i are closely related, these variables should be placed adjacently in the order of variables on BDD's to reduce the size of them. Therefore the variables for compatibles and these for signatures are interleaved.

On line 1 in Fig.5, for each compatible c, all the subsets s are generated. These subsets are candidates for signatures of the compatible. A recursive function *SubsetDoubling* is introduced for the subset generation. If the set C of compatibles is $C = \overline{x_i} \cdot C_i^0 + x_i \cdot C_i^1$, then

$$SDoubling(C) = \overline{x_i}\cdot\overline{x'_i}\cdot SDoubling(C_i^0) + x_i \cdot SDoubling(C_i^1).$$

Set R on line 3 is the set of all the candidates to be dropped for input-labeled implied pair (p,i). For example, let us consider a target input-labeled implied pair $(p,i) = (C,D,0)$ in Fig.4 in order to show how to calculate the

gen_prime_compatible($Compatibles\ C,\ Merger\ T$) {
1. $P \leftarrow \{(c,s)|c \in C,\ s \subseteq c\}$;
2. **for** each input-labeled implied pair (p,i) in T {
3. $R \leftarrow \{(r,r')|r \not\supseteq p, PWCS(r) \supseteq p, PWCS(r') \not\supseteq p\}$;
4. $P \leftarrow P \backslash R$;
5. }
6. $P \leftarrow Prime(P)$;
7. $PC \leftarrow ExtractPrime(P)$;
8. **return** PC;
}

Fig. 5. Implicit generation of pw-prime compatibles

BDD. Since (A,E) and (B, D) imply p, R is calculated as follows:

$$R = (\overline{C} + \overline{D})(AE + BD)(\overline{A'E' + B'D'}).$$

$Prime(P)$ on line 6 is the function to calculate the pw-prime compatibles. Note that for every $(c,s) \in P$, c contains s throughout procedure *gen_prime_compatible*.

$Prime(P)$ can be also expressed with recursive equations. Let $P = \overline{x_i} \cdot \overline{x_i'} \cdot P_i^{00} + x_i \cdot \overline{x_i'} \cdot P_i^{10} + x_i \cdot x_i' \cdot P_i^{11}$: then

$$\begin{aligned} Prime(0) &= 0 \\ Prime(1) &= 1 \\ Prime_i^{00}(P) &= Prime(P_i^{00} \backslash EqDom(P_i^{00}, P_i^{11})) \\ Prime_i^{10}(P) &= Prime(P_i^{10} \backslash Dominated(P_i^{01}, P_i^{11}) \\ Prime_i^{11}(P) &= Prime(P_i^{11} \backslash Dominated(P_i^{11}, P_i^{10})) \end{aligned}$$

$EqDom(P,Q)$ is an operation that calculates all the compatibles in P pw-dominated by or equivalent to at least one compatible in Q. P and Q are sets of compatible-signature pairs. It can be calculated as follows:

$$\begin{aligned} EqDom(0,Q) &= 0 \\ EqDom(P,0) &= 0 \\ EqDom(P,1) &= \mathrm{P} \\ EqDom(P,P) &= \mathrm{P} \\ EqDom_i^{00}(P,Q) &= EqDom(P_i^{00}, Q_i^{00} \cup Q_i^{10}) \\ EqDom_i^{10}(P,Q) &= EqDom(P_i^{10}, Q_i^{10} \cup Q_i^{11}) \\ EqDom_i^{11}(P,Q) &= EqDom(P_i^{11}, Q_i^{11} \cup Q_i^{10}) \end{aligned}$$

$Dominated(P,Q)$ is an operation that calculates all the compatibles in P pw-dominated by at least one compatible in Q. P and Q are also sets of compatible-signature pairs. It is calculated in the recursive manner like $EqDom$.

When set P is represented by BDD's, it requires as many Boolean variables as class sets. That is, it requires $2n$ variables. However, in a pair (c,s) in P, s is always subset of c, while this does not hold when class sets are used. This is likely to be suitable for BDD's, especially for Ternary Decision Diagrams (TDD's). In TDD's, these functions require only n variables, while three edges are necessary for each node. Though a nodes of TDD's require about one and a half times as much memories as that of BDD's,the TDD representing compatibles and their signature still require much less memories than the BDD representing compatibles and their class sets.

V. Experimental Results

We implemented the algorithm described in the previous sections in a program. To represent a set of compatible-signature pairs, we use BDD's with the interleaving variable ordering instead of TDD's.

We ran our program IPWCS on large FSM benchmarks reported in [8]. Comparisons are made with ISM [8] and STAMINA [3] (shortened as STAM. in the table). ISM is an implicit method and STAMINA is an explicit one.

Table I summarizes the result of prime compatible generation. In Table I, N_s, N_{comp} are the numbers of states, and compatibles, respectively. Column N_{prime} shows the numbers of pw-prime compatibles or prime compatibles. Entry "u.f." shows that the number of pw-prime compatible cannot be computed from the resulting BDD's because of underflow. Column `time(sec)` shows the run time required for generating all the pw-prime (or prime) compatibles. The run times of ISM are reported in [8]. It was run on DECstation 5000/260(440Mb). IPWCS and STAMINA were run on Sun SPARCstation 10(96Mb).

Experimental results show that in many tested benchmarks our method(IPWCS) takes much less time than ISM, though machines for experiments are different. Note that for all the examples that STAMINA completes STAMINA takes equal or more time than IPWCS. On the other hands, ISM takes more time than STAMINA for some examples. For *fo.70* ours is about 20 times faster than ISM. IPWCS failed on the benchmarks *rubin600, rubin1200, rubin2250*, while ISM completed them. These examples are artificially constructed to have the number of prime compatibles exponential in the number of states and have huge number of implied pairs to be dealt with, which is the main reason our program failed.

Though our dominance checking is weaker than that in ISM or STAMINA, in many cases the number of the pw-prime compatibles is the same as that of the prime compatibles. The difference between the numbers of both pw-prime and prime compatibles is not negligible for the example from *e271* to *e680*, which have been randomly generated. These results indicates that proposed method is likely to generate almost the same pw-prime compatibles as prime compaitibles efficiently for practical ISFSM's.

VI. conclusions

We proposed a new implicit algorithm of generating prime compatibles for state minimization of ISFSM's. A new idea of signatures of compatibles are introduced. Although the dominance check is weaker than that by class

TABLE I
EXPERIMENTAL RESULTS

FSMs	N_s	N_{comp}	N_{prime}		time(sec.)		
			IPWCS	ISM/STAM.	IPWCS †	ISM ‡	STAM. †
alex1	42	55928	787	787	14	24	22
intel_edge.dummy	28	9432	396	396	6	37	6
isend	40	2.876e8	480	480	43	13	fails
pe-rcv-ifc.fc	46	1.528e11	148	148	108	114	fails
pe-rcv-ifc.fc.m	27	1.793e6	38	38	0	3	304
pe-send-ifc.fc	70	5.071e17	–	506	fails	571	fails
pe-send-ifc.fc.m	26	8.978e6	23	23	1	3	332
vbe4a	58	1.756e12	u.f.	2072	36	109	309
vmebus.master.m	32	3.842e7	28	28	1	26	fails
th.30	31	97849	33064	33064	10	21	2450
th.40	41	1.456e6	529420	529420	42	75	fails
th.55	55	3.622e7	1.555e7	1.555e7	215	1273	fails
fo.20	21	42193	12762	12762	2	2	397
fo.50	51	3.643e7	1.697e7	1.697e7	62	216	fails
fo.70	71	9.622e10	4.524e10	4.524e10	1002	22940	fails
ifsm0	38	1.0649e6	18686	18686	75	43	3147
ifsm1	74	43006	u.f.	8925	39	25	abort
ifsm2	48	497399	774	774	4	267	837
rubin18	18	4095	4095	4095	0	0	640
rubin600	600	$2^{400}-1$	–	$2^{400}-1$	fails	1978	fails
rubin1200	1200	$2^{800}-1$	–	$2^{800}-1$	fails	27105	fails
rubin2250	2250	$2^{1500}-1$	–	$2^{1500}-1$	fails	271134	fails
e271	19	393215	106461	96383	50	21	fails
e285	19	393215	121665	121501	47	13	fails
e304	19	393215	265537	264079	39	93	fails
e423	19	204799	161428	160494	19	102	fails
e680	19	327699	193788	192803	53	151	fails

†: It was run on Sun SPARCstation10(96Mb)
‡: It was run on DECstation 5000/260(440Mb)

sets, the number of dominated compatibles are the same in many cases. Experimental results show that the proposed method can generates prime compatibles efficiently for almost all the practical ISFSM's.

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 FSM benchmarks.

REFERENCES

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

[2] A. Grasselli and F. Luccio. "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks". *IRE Trans. on Electronic Computers*, 14(3):350–359, June 1965.

[3] 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*, 13(2):167–177, February 1994.

[4] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[5] G. Swamy, R. Brayton, and P. McGeer. "A Fully Implicit Quine-McCluskey Procedure Using bdd's". In *UCB Technical Report No.UCB/ERL M92/127*, June 1992.

[6] O. Coudert, J. C. Madre, and H. Fraisse. "A New Viewpoint on Two-Level Logic Minimization". In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 625–630, 1993.

[7] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli. Espresso-Signature: A New Exact Minimizer for Logic Functions. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 618–624, June 1993.

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

[9] Z. Kohavi. *"Switching and Finite Automata Theory 2/e"*. Tata McGraw-Hill, 1978.