IEICE TRANS. INF. & SYST., VOL.E83–D, NO.10 OCTOBER 2000

LETTER

On Processing Order for Obtaining Implication Relations in Static Learning

Hideyuki ICHIHARA[†], **Seiji KAJIHARA**[††], *and* **Kozo KINOSHITA**[†††], ***Regular Members***

SUMMARY Static learning is a procedure to extract implication relations of a logic circuit. In this paper we point out that the number of the extracted implication relations by static learning depends on the order of signal lines processed. Also, we show four procedures for ordering signal lines processed and the effectiveness of the ordering procedures by experiments.

key words: *test generation, implication, static learning*

1. Introduction

Implication relations between signal value assignments in a logic circuit play an important role in the field of VLSI CAD. Implication relations are extracted from the netlist and can be used for various purposes. In test generation, static learning that has been proposed in SOCRATES [1], is known as one of such applications. Since the use of more implication relations can assign more logic values uniquely, signal lines whose values have not been assigned yet can be reduced. Thus the number of backtracks, which is strongly related to the run time of test generation, decreases. As other applications, there exist logic optimization, logic verification, and untestable-path analysis. Implication relations give information on redundancy in logic optimization [2], [3], and are used to check logic tautology in logic verification [4]. Also, they assist to find more necessary assignments for sensitizing a path in untestable-path analysis [5].

Static learning assigns values to lines by an *implication procedure* and obtains implication relations from the set of the value assignments by *contrapositive operation*. The implication procedure is done for each line in an order of lines. This processing order sometimes affects the number of implication relations obtained, while an example is given later. To obtain more implication relations, therefore, it is necessary to consider the better ordering method for processing. However, there are no studies which examine the effectiveness of the order of signal lines processed in static learning.

In this paper, we propose four ordering procedures which are based on the level of lines, and show experimental results to find out the better ordering procedure that can extract more implication relations.

The remainder of this paper is organized as follows. In Sect. 2, we explain static learning. In Sect. 3, we discuss the relationship between the order of processed lines and the number of implication relations. In Sect. 4, we propose four procedures to produce the order of processed lines, and show the some experimental results for static learning with these ordering procedures in Sect. 5. This paper is concluded in Sect. 6.

2. Static Learning

This paper deals with multi-level combinational circuits. One of three values 0, 1, X (don't-care or undefined) is assigned to each signal line. *Value assignment* is to assign logic value $v_s \in \{0, 1\}$ to signal line s and denoted by $s = v_s$. If value assignment $a = v_a$ uniquely determines value assignment $b = v_b (v_a, v_b \in \{0, 1\})$, this relation between two value assignments is called *implication relation* and denoted by $a = v_a \rightarrow b = v_b$.

Implication procedure is to assign values that are uniquely determined from a given value assignment. Basically, the implication procedure repeats *direct implication operation*.

Definition(Direct Implication Operation): Direct implication operation is defined as assigning signal values as follows, when value assignment $a = v_a$ is given.

1. In the case where line a is the output (an input) of gate G_a, assign all values on every input (the output and other inputs) which are uniquely determined by $a = v_a$ according to the truth table of gate G_a.
2. In the case where line a is the stem (a branch) of fanout F, assign v_a to all branches (the stem and the other branches) of F.

The implication procedure uses a set of implication relations that are extracted by static learning, in order to assign more values. In this paper, we call a set of implication relations extracted *implication dictionary*. Figure 1 shows the implication procedure starting from a value assignment $a = v_a$, where DIC is an implication dictionary. The algorithm decides value assignments by direct implication operation (line 10) and by looking implication dictionary up (lines 11-12) until no

Manuscript received April 21, 2000.

[†]The author is with the Dept. of Information Sciences, Hiroshima City Univ., Hiroshima-shi, 731–3194 Japan.

[††]The author is with the Computer Science and Electronics Dept., Kyushu Institute of Technology, Iizuka-shi, 820–0053 Japan.

[†††]The author is with the Faculty of Informatics, Osaka Gakuin University, Suita-shi, 564–8511 Japan.

```
1  implication_procedure(a = v_a)
2  {
3     V : a stack of value assignments;
4     Set V = φ;
5     Push a = v_a into V;
6     while (V is not empty) {
7        Pop a value assignment b = v_b from V;
8        if(b = v̄_b has been assigned) return FAILURE;
9        Assign b = v_b;
10       Push value assignments obtained by direct implica-
         tion operation for b = v_b into V;
11       for (each implication relation in DIC such that b =
         v_b → c = v_c)
12          Push c = v_c into V;
13    }
14    return SUCCESS;
15 }
```

Fig. 1 Algorithm of implication procedure.

```
1  static_learning()
2  {
3     DIC: an implication dictionary;
4     V : a set of signal value assignments;
5     Set DIC = φ and V = φ;
6     for (each line i) {
7        for (each value v_i ∈ {0, 1}){
8           Set signal assignments obtained by implica-
            tion_procedure(i = v_i) to V;
9           for(each i = v_i → j = v_j ∈ V)
10             if (i = v_i → j = v_j satisfies the learning crite-
               rion)
11                add j = v̄_j → i = v̄_i to DIC (Contrapositive
                  operation for i = v_i → j = v_j) ;
12       }
13    }
14 }
```

Fig. 2 Algorithm of static learning.

new value are assigned.

DIC is constructed in static learning, which is based on the implication procedure and *contrapositive operation*, which is the operation to obtain implication relation $b = \overline{v_b} \rightarrow a = \overline{v_a}$ from $a = v_a \rightarrow b = v_b$.

Figure 2 shows the algorithm of static learning. The algorithm repeats processes from line 8 to line 11 according to a line order at line 6. The implication procedure is called at line 8. Implication relations obtained are checked by the learning criterion (line 10) so as to find implication relations not generated by only repeating direct implication operation. We call the order of lines selected at line 6 *processing order*, and call a given value assignment for the implication procedure, i.e. $i = v_i$ in Fig. 2, *start assignment*.

An example that implication relation $f = 1 \rightarrow c = 1$ is obtained by static learning is shown by using the circuit of Fig. 3. Consider that implication dictionary DIC is empty initially and line c is selected as value i at line 6 in Fig. 2. At line 8, implication_procedure($c = 0$) assigns $d = 0, e = 0, f = 0$ and $g = 1$. Then at line 10, since $c = 0 \rightarrow f = 0$ satisfies the learning criterion, implication relation $f = 1 \rightarrow c = 1$ is obtained by contrapositive operation at line 11.

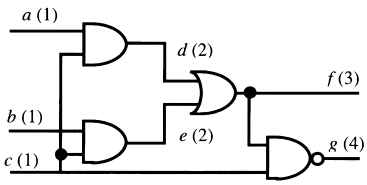

Fig. 3 Example circuit.

3. The Processing Order and Implication Relation

In static learning, an implication dictionary, which is constructed at line 11 of Fig. 2, is used for the implication procedure at line 11 of Fig. 1. The processing order affects the construction of the implication dictionary. As a result, there exist implication relations which can be found in a processing order, but not found in another processing order. We give an example below.

Suppose two processing orders for the circuit of Fig. 3: processing order A is that f appears before c and processing order B is that f appears after c. The results for two processing orders are shown in Table 1. In case of processing order A, the implication_procedure($f = 1$) does not derive any implication relation. Only implication relation $f = 1 \rightarrow c = 1$ is found by contrapositive operation from $c = 0 \rightarrow f = 0$, which is obtained by the implication_procedure($c = 0$). In case of processing order B, implication relation $f = 1 \rightarrow c = 1$ is first found by contrapositive operation after the implication_procedure($c = 0$). Therefore implication_procedure($f = 1$) can produce $f = 1 \rightarrow g = 0$. As a result, implication relation $g = 1 \rightarrow f = 0$ is obtained by contrapositive operation from implication relation $f = 1 \rightarrow g = 0$. Thus, processing order B can find more implication relations than processing order A.

The relationship between start assignments and implication relations extracted by static learning can be described as a directed acyclic graph. Figure 4 illustrates the relationship between start assignments and implication relations in the circuit of Fig. 3. The leftmost vertices show the start assignments and the other vertices show implication relations obtained by static learning. A directed edge from a start assignment to an implication relation means that the implication relation is extracted from the start assignment. A directed edge between two implication relations means that the pointed implication relation is extracted us-

Table 1 Example of processing order.

	start assignment	obtained implication relation
Processing Order A	1. $f = 1$	None
	2. $c = 0$	$f = 1 \rightarrow c = 1$
Processing Order B	1. $c = 0$	$f = 1 \rightarrow c = 1$
	2. $f = 1$	$g = 1 \rightarrow f = 0$

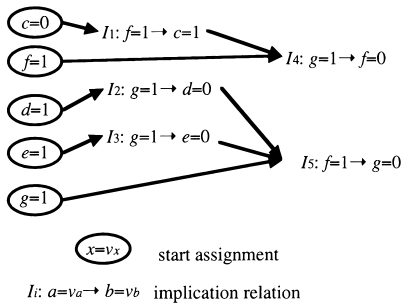

Fig. 4 Relationship between implication relations and start assignments.

ing the pointing implication relation. For example, implication relation $I_4 : g = 1 \rightarrow f = 0$ is obtained from start assignment $f = 1$ using implication relation $I_1 : f = 1 \rightarrow c = 1$.

Implication relations are classified into two types: one is extracted without using other implication relations such as I_1, I_2, I_3 in Fig. 4. The others are extracted using other implication relations such as I_4, I_5. Although the former type of implication relations is extracted regardless of the processing orders, the latter type of implication relations may not be extracted in some processing orders. In this example, it is desirable that processing order contains sequence *c-d-e-f-g*.

4. Ordering Procedures

Here, we consider line ordering procedures introduced into static learning. Since the number of line orders for a circuit is $n!$ where n is the number of lines, it will be difficult to find the best order for a given circuit. In this section, we introduce four basic ordering procedures to verify the influence of processing orders. They are based on the level of line.

Definition(Level of Line): The level of line a is defined as the maximum number of multi-input-gates on paths from primary inputs to a and denoted by L^a .

Four ordering procedures are Forward with Breadth-first (FB), Backward with Breadth-first (BB), Forward with Depth-first (FD) and Backward with Depth-first (BD) ordering procedures. Forward means that lines are numbered from inputs to outputs, while backward means that lines are from outputs to inputs. Breadth-first and Depth-first mean giving priority to select lines with same and different levels, respectively. Since these procedures are known as basic searching algorithms, they are suitable to verify the influence of processing orders.

```
1  [FB/BB]()
2  {
3     Porder : a sequence of lines;
4     Set Porder = φ;
5     for (each k from [0/max_level] to [max_level/0])
6        Concatenate sequence of lines {a|L^a = k, a; line} to
         Porder;
7     return Porder;
8  }
```

Fig. 5 Ordering procedure FB/BB. (The left word in the bracket [] is for FB and the right for BB.)

```
1  [FD/BD]()
2  {
3     Porder : a sequence of lines;
4     S : a set of lines;
5     Set Porder = φ;
6     for (each primary [input/output] p) {
7        Set lines in transitive fanout of p to S;
8        for (each k from [0/max_level] to [max_level/0])
9           Concatenate sequence of lines {a|L^a = k, a ∈ S}
            to Porder;
10    }
11    return Porder;
12 }
```

Fig. 6 Ordering procedure FD/BD. (The left word in the bracket [] is for FD and the right for BD.)

We give FB and BB in Fig. 5 and FD and FB in Fig. 6. The *max_level* means the maximum level of the circuit.

For the circuit of Fig. 3, the level of line is shown in parentheses. Each ordering procedure provides the following processing order.

FB: *a-b-c-d-e-f-g*
BB: *g-f-d-e-a-b-c*
FD: *a-d-f-g-b-e-c*
BD: *f-d-a-e-b-c-g*

5. Experimental Results

We implemented static learning with the proposed ordering procedures in C language and applied them to ISCAS'85 benchmark circuits and combinational parts of ISCAS'89 benchmark circuits on a PC (CPU: Pentium Pro 150 MHz, MEMORY: 128 M, OS: FreeBSD).

Table 2 shows the result of static learning with each ordering procedure. The column "the number of implication relations" shows the number of implication relations obtained by each ordering procedure. The column "max" shows the number of implication relations obtained by the procedure iterating static learning until no new implication relation is obtained. The column "the ratio to #max" is the ratio of the number

Table 2 Number of implication relations obtained by static learning with four ordering procedures.

circuit	the number of implication relations					the ratio to #max				
name	FB	BB	FD	BD	max	FB	BB	FD	BD	max
c432	66	57	57	57	66	1.00	0.86	0.86	0.86	1.00
c499	40	40	40	40	40	1.00	1.00	1.00	1.00	1.00
c880	85	85	85	85	85	1.00	1.00	1.00	1.00	1.00
c1355	208	208	208	208	208	1.00	1.00	1.00	1.00	1.00
c1908	1021	517	648	609	1024	1.00	0.50	0.63	0.59	1.00
c2670	1171	843	908	905	1174	1.00	0.72	0.77	0.77	1.00
c3540	4687	4063	4157	4466	4714	0.99	0.86	0.88	0.95	1.00
c5315	2928	1429	1458	2025	2933	1.00	0.49	0.50	0.69	1.00
c6288	1027	663	678	1072	1081	0.95	0.61	0.63	0.99	1.00
c7552	9341	3633	4112	5475	9753	0.96	0.37	0.42	0.56	1.00
s9234	28596	19639	20964	26765	29593	0.97	0.66	0.71	0.90	1.00
s13207	100844	73470	74934	100520	102793	0.98	0.71	0.73	0.98	1.00
s15850	39901	24909	34943	40091	41854	0.95	0.60	0.83	0.96	1.00
s35932	2811	2811	2811	2811	2811	1.00	1.00	1.00	1.00	1.00
s38417	20247	18618	18833	19753	20321	1.00	0.92	0.93	0.97	1.00
s38584	252420	239100	240397	249227	253374	1.00	0.94	0.95	0.98	1.00

Table 3 Number of iterations of static learning.

circuit	# of iterations				one pass
name	FB	BB	FD	BD	time [sec.]
c432	1	2	2	2	0.06
c499	1	1	1	1	0.17
c880	1	2	1	1	0.16
c1355	1	1	1	1	1.19
c1908	2	7	6	4	1.24
c2670	2	7	5	6	1.43
c3540	2	4	4	3	6.92
c5315	1	6	6	4	3.97
c6288	2	2	2	2	3.19
c7552	2	10	9	6	10.94
s9234	2	4	4	3	31.83
s13207	2	2	3	2	220.60
s15850	2	3	3	2	67.25
s35932	1	1	1	1	414.26
s38417	2	3	3	2	260.38
s38584	2	4	4	3	1182.00
average	1.62	3.68	3.43	2.68	147.04

of implication relations obtained by each ordering procedure to "max", respectively. This result shows that the processing order affects significantly the number of implication relations to be found. For c1908, c5315 and c7552 the number of implication relations obtained by BB is no more than the half of FB. Moreover the number of implication relations found by FB is comparable to the maximum for many circuits. Therefore FB is the best ordering procedure among four for these circuits.

Table 3 shows results of iterating static learning. The each column of "# of iterations" shows the number of iterations of static learning required to obtain maximum implication relations, which are shown in "max" of Table 2, in each processing order. The column "one pass time" means the run time of static learning with FB. FB requires fewer iterations than others. Especially, the number of iterations required for FB is less than 2.

6. Conclusion

In this paper we discussed the importance of the line processing order in static learning. In the experiment for benchmark circuits, we showed that the ordering procedure FB, which is based on a Forward with Breadth-first manner, can derive almost all implication relations of the circuits.

References

[1] M.H. Schulz, E. Trischler, and T.M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," IEEE Trans. Comput.-Aided Des. Integrated. Circuits & Syst., vol.7, no.1, pp.126–137, 1988.

[2] W. Kunz and P.R. Menon, "Multi-level logic optimization by implication analysis," Proc. Int'l Conf. on CAD, pp.6–13, 1994.

[3] H. Ichihara and K. Kinoshita, "On acceleration of logic circuits optimization using implication relations," Proc. 6th ATS, pp.222–227, 1997.

[4] W. Kunz, "HANNIBAL: An efficient tool for logic verification based on recursive learning," Proc. Int'l Conf. on CAD, pp.538–543, 1993.

[5] S. Kajihara, K. Kinoshita, I. Pomeranz, and S.M. Reddy, "A method for identifying robust dependent and functionally unsensitizable paths," Proc. 10th IEEE Int'l Conf. on VLSI Design, pp.82–87, 1997.

[6] H. Ichihara, S. Kajihara, and K. Kinoshita, "An efficient procedure for obtaining implication relations and its application to redundancy identification," The Seventh Asian Test Symposium, pp.58–63, 1998.