論 文
35~3~1

# PROLOG에 의한 D-algorithm의 具現에 관한 硏究

## Implementation of D-algorithm by using PROLOG

金 明 起*·文 榮 得**
(Myung-Ki Kim · Young-Deuk Moon)

**Abstract**

This paper introduce a new test generation method based on built-in data base which is suitable for generating of test set by using PROLOG language.

The program presented in this paper deals with all the information required for fault detection from the rules describing output signals and internal signals.

Example shows the validity of proposed PROLOG program which results in a effective generation of test set comparable to the conventional D-algorithm.

## 1. 序 論

최근 LSI 및 VLSI 技術의 발전으로 論理回路의 복잡성이 증가함에 따라 故障發生 確率이 높아졌으며, 이에 따라 論理回路에 대한 fault detection 문제가 더욱 중요하게 되었으며 信賴度가 요청되기 때문에 論理回路의 故障檢出에 대해 여러가지 방법이 연구되어 왔다.[1),2),3),5),6),7)]

Test의 容易性을 위한 설계로 알려진 LSSD(level sensitive scan design) 方法은 IBM社가 提案한 것으로 LSI順序論理回路를 test할 때 flip-flop을 Shift resister가 되게 구성하고 組合論理回路를 별도로 test한 것이며,[1),9)] 가장 폭넓게 사용되는 것으로는 Roth[8]에 의해 보고된 D-algorithm이 있다. Truth table method를 기본으로 한 Boolean Difference와 Path Sensitizing Method 등이 論理回路 故障檢出 방법인데, 故障의 내용에 따라서 특정한 값을 갖는 신호를 선정하는 것이 필요하게 되어 提案된 것이 D-algorithm이다. 그러나 D-algorithm에서는 backtrack의 횟수가 문제가 되어 PODEM 과 FAN알고리즘이 발표되었다. PODEM[6]과 FAN[7] 알고리즘은 D-algorithm의 backtrack횟수를 감소시켜 testset생성의 시간을 단축시켰다.

본 논문에서는 PODEM과 FAN 알고리즘과는 달리 논리프로그래밍 언어인 PROLOG를 사용하여 backtrack의 횟수를 감소시키고 testset생성을 위한 回路의 표현도 PROLOG로 처리하여 보다 효율적인 testset생성방법을 다루려한다.

본 논문에서는 D-algorithm에서 처럼 D交差法에 의해서 testset를 구하지 않고 PROLOG를 사용하여 設計하려는 論理回路의 기본검사 pattern을 데이타베이스로 만들어 간편하고 신속하게 故障을 test할 수 있게 하였다. 이를 위해 2 장과 3 장에서 D-algorithm에 의한 故障檢出方法과 D-algorithm의 기본개념을 응용하여 새로운 故障檢出方法을 PROLOG로 具現하는 방법에 대하여 설명하고 4 장에서 결론을 내리기로 한다.

## 2. D-algorithm에 의한 故障檢出 方法

Eldred가 提案한 一次元活性化法은 게이트에서 나간 출력단자가 다른 게이트에서 다시 입력으로 된

*正 會 員 : 東亞大 工大 電子工學科 敎授 · 工博
**正 會 員 : 釜山外國語大學 電算學科 助敎授
接受日字 : 1985年 9月 21日

경우에는 일부 단자에 대해 故障檢出을 위한 test 입력을 생성시키지 못하는 경우가 생기게 된다. 그래서 여러개의 經路를 동시에 活性化하는 方法을 위해 D-algorithm을 사용한다.

이 論에서는 D-algorithm에 사용되는 용어의 定義를 간단히 설명하고 제3장에서는 PROLOG로 각각의 용어가 어떻게 具現되는지 설명하겠다. 다음 定義 1은 故障信號에 관한 定義이다.

定義 1) 0으로 固定된 신호와 1로 固定된 신호를 각각 D고장신호 및 $\bar{D}$고장신호라고 부른다.

定義에 의해 入力段故障과 出力段故障이 D 고장과 $\bar{D}$고장에 대해 어떻게 표현되었는지 그림 1에서 보여주고 있다.

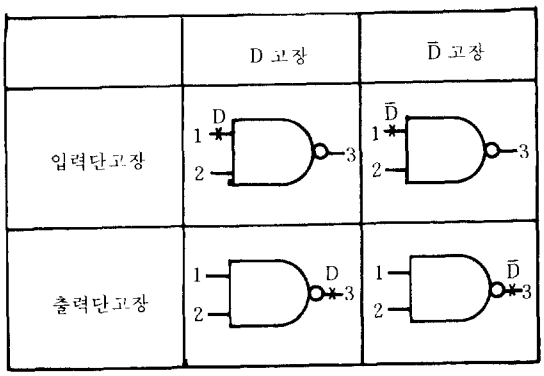



**그림 1.** D와 $\bar{D}$의 표시

**Fig. 1.** Representation of D and $\bar{D}$.

定義 2) 그림 2와 같이 나타낸 것을 故障의 基本 D-cube(primitive D-cube of failure;pdcf)라 한다.

여기서 $x$는 0이나 1로도 故障信號發生에 관여하지 않는 것을 나타낸다.

故障의 基本D-cube는 입력단자의 故障이 0으로 틀렸으면 D를, 1로 틀렸으면 $\bar{D}$로 故障단자를 표시하는 숫자 밑에 쓴다.

Example 1) 그림 1의 NAND게이트에서 입력단자 1이 D로 故障의 基本D-cube는

| | 1 | 2 | 3 |
|---|---|---|---|
| | D | $x$ | $x$ |

이다.

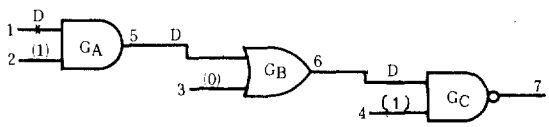
定義 3) 故障信號를 전달시키는 조건을 표시한 것을 전달D-cube(propagation D-cube;pdc)라 한다.

그림 3(b)에서 $C_0$는 pdcf이고, $C_1$, $C_2$, $C_3$는 게이트A, 게이트B, 게이트C의 전달D-cube이며, $C_3$의 전달D-cube에서 단자 7이 $\bar{D}$로 되는 것은 게이트C의 否定機能인 때문이며, $C_{0123}$는 단자1의 고장신호를 단자 7로 전달시키는 경우에 각 단자가 가지는 조건을 나타내고 있다. Cube $C_{0123}$는 "D交差(D-intersection)"라 불리우는 연산을 사용해서 $C_0$, $C_1$, $C_2$, $C_3$로 부터 만들어진다. 0, 1, $x$, D, $\bar{D}$等의 두가지로 組合된 D交差를 표시하면 표1과 같다.

표 1. D交差의 定義

**Table1.** Definition of D-intersection.

| ∩ | 0 | 1 | $x$ | D | $\bar{D}$ |
|---|---|---|---|---|---|
| 0 | 0 | q | 1 | Q | Q |
| 1 | q | 1 | 1 | Q | Q |
| $x$ | 0 | 1 | $x$ | D | $\bar{D}$ |
| D | Q | Q | D | D | Q |
| $\bar{D}$ | Q | Q | $\bar{D}$ | Q | $\bar{D}$ |




(a) 신호전달회로

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| $C_0$ = | D | $x$ | | | $x$ | | | } ····· pdcf |
| $C_1$ = | D | 1 | | | D | | | |
| $C_2$ = | | | 0 | | D | D | | } ····· pdc |
| $C_3$ = | | | | 1 | D | $\bar{D}$ | | |
| $C_{0123}$ = | D | 1 | 0 | 1 | D | D | $\bar{D}$ | |

(b) 전달D-cube

**그림 3.** 신호전달 회로와 전달D-cube

**Fig. 3.** Signal propagation circuit and propagation D-cube.

| | D 고장 | | | $\bar{D}$ 고장 | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 1 | 2 | 3 |
| 입력단고장 | D | $x$ | $x$ | $\bar{D}$ | $x$ | $x$ |
| 출력단고장 | 0 | $x$ | D | 1 | 1 | $\bar{D}$ |
| | $x$ | 0 | D | | | |

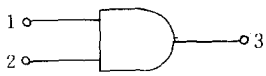
**그림 2.** 故障의 基本 D-cube

**Fig. 2.** Primitive D-cube of failure.

표에서 $C_\alpha \cap C_\beta = q$ 또는 Q로 된 때는 cube $C_\beta$의 요구 조건에 따라 틀린 신호를 전달시킬 수 없다는 것을 의미하는 것이다.

Example 2) 그림 3(a)의 回路는 단자 1 에서 발생하는 故障信號D를 단자 5 에 전달시키는 경우를 나타내주는 것으로 故障의 基本D-cube로 표시하면 그림 3(b)와 같다.

定義 4) 特異cover(singular cover)는 게이트에 대한 진리표를 간단히 표현한 것과 같은 것을 말한다. 그림 4 에서 特異cover를 보여주고 있다.




| 1 | 2 | 3 |
|---|---|---|
| x | 0 | 0 |
| 0 | x | 0 |
| 1 | 1 | 1 |

**그림 4**. AND게이트와 특이cover
**Fig. 4**. AND gate and singular cover.

처음 행에서 0 x 0는 입력단자 1의 신호치가 0이면 입력단자 2의 신호치에 관계없이 출력단자 3의 신호치는 0이 되는 것을 나타내며 다른 행도 같은 내용이다. 回路의 特異cover는 각 게이트의 特異cover를 그대로 중첩하므로 구하여진다.

Example 3) 그림 5 에서 보여주는 回路를 예로 하여 特異cover로 나타내면 그림 6 과 같다.

그림 5 에서 단자 5 를 출력단자로 하는 게이트 A는

$$C_3 = \begin{array}{|ccccccccc|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & 1 & & & 0 & & & & \\ \hline \end{array}$$

을 근거로하여 게이트A의 cube $C_A$는

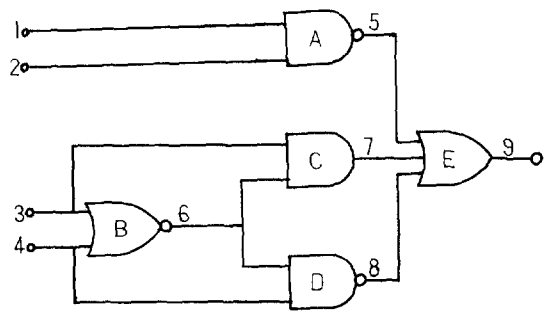



**그림 5**. 故障回路의 例
**Fig. 5**. Example of fault circuit.

$$G_A \left\{ \begin{array}{|ccccccccc|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & 1 & & & 0 & & & & \\ 0 & x & & & 1 & & & & \\ x & 0 & & & 1 & & & & \\ \hline \end{array} \right.$$

$$G_B \left\{ \begin{array}{|ccccccccc|} \hline 0 & 0 & & & 1 & & & & \\ 1 & x & & & 0 & & & & \\ x & 1 & & & 0 & & & & \\ \hline \end{array} \right.$$

$$G_C \left\{ \begin{array}{|ccccccccc|} \hline 1 & & & & 1 & 1 & & & \\ 0 & & & & x & 0 & & & \\ x & & & & 0 & 0 & & & \\ \hline \end{array} \right.$$

$$G_D \left\{ \begin{array}{|ccccccccc|} \hline 1 & & 1 & & & 0 & & & \\ 0 & & x & & & 1 & & & \\ x & & 0 & & & 1 & & & \\ \hline \end{array} \right.$$

$$G_E \left\{ \begin{array}{|ccccccccc|} \hline 1 & & & x & x & 1 & & & \\ x & & & 1 & x & 1 & & & \\ x & & & x & 1 & 1 & & & \\ 0 & & & 0 & 0 & 0 & & & \\ \hline \end{array} \right.$$

**그림 6**. 그림 5 의 特異cover
**Fig. 6**. Singular cover of Fig. 5.

$$C_A = \begin{array}{|ccccccccc|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & 1 & 0 & 0 & 0 & D & \bar{D} & \bar{D} & \bar{D} \\ \hline \end{array}$$

가 얻어진다.

이 $C_A$에서는 q도 Q도 나타나지 않는다. 그러므로 test입력 1, 2, 3, 4 가 1, 1, 0, 0로 구하여지는 것을 알 수 있으며 단자 9 의 값이 $\bar{D}$로 되어 있어 test입력은 回路가 정상적이면 출력치가 0이 되고, 故障이 났으면 1이 되는 것을 알 수 있다.

지금까지 D-algorithm에서 使用되는 용어를 중심으로 살펴보았다. D-algorithm은 PROLOG로그대로 구현하는 것은 무의미하므로 보다 개선된 방법을 제 3 장에서 다룬다.

故障檢出에 필요한 정보를 데이타베이스로 구축하여 故障檢出을 PROLOG의 backtracking과 pattern matching을 활용하여 D-algorithm을 보다 더 간소화 시킬 수 있다.

## 3. PROLOG에 의한 D-algorithm의 具現

앞 章에서 D-algorithm에 대하여 간단히 설명하였다.

본 章에서는 故障檢出 과정에서 PROLOG가 어떻게 효율적으로 使用되는지 D-algorithm과 비교하여 설명하겠다. 먼저 PROLOG의 특징은 다음과 같다.

PROLOG는 일반적인 컴퓨터언어와는 달리 論理에 기반을 둔 언어로서 어떤 문제를 해결하기 위해 다음 세가지로 구성되고 있다.

1. Object와 그들의 관계에 대한 사실선언(fact)
2. Object와 그들의 관계에 대한 규칙정의(rule)
3. Object와 그들의 관계에 대한 질문(query)

사실선언(fact)은 D-algorithm의 pdcf, pdc와 sc를 동시에 처리할 수 있다. 그리고 규칙정의(rule)에서는 論理回路의 論理的 表現을 다루게 되고, 질문(query)는 故障檢出 時 필요한 정보를 물어보는 것으로 모든 경우의 故障檢出을 요구하지 않고 해당 고장검출만을 알고 싶은 경우 주로 사용할 수있다.

표 2는 PROLOG를 사용하여 고장검출용 프로그램을 작성할 경우 필요한 구성요건들이다. D-algorithm과는 달리 많은 과정이 PROLOG의 fact로 데이타베이스화 될 수 있고 구성된 데이타베이스를 사용하는 규칙(rule)도 backtracking과 pattern matching 때문에 간소화되어 고장검출용 software 의 효율을 높일 수 있다.

**표　2.** 고장검출용 프로그램의 구성요건
**Table2.** Requirement of the fault detection program.

| |
|---|
| 論理게이트에 대한 pdcf, pdc, sc의 데이타베이스化 |
| 論理回路의 論理的 表現 |
| Backtracking을 위한 rule의 表現 |
| 自動的인 입력선택을 위한 rule의 表現 |

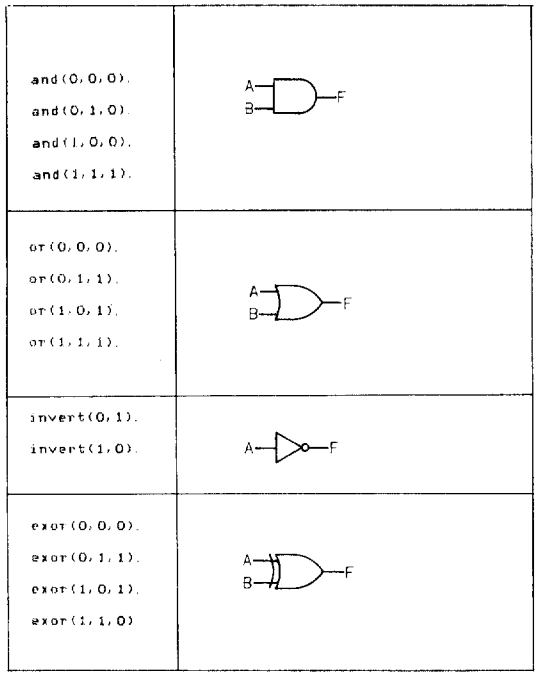
論理게이트의 데이타베이스 設計는 각 論理게이트의 진리표 구성과 같다. 다시 말하면 D-algorithm에서 사용하는 singular cube와 같다고 볼 수 있다. 따라서 프로그램 1과 같이 각 論理게이트에 대해서 데이타베이스를 만들 수 있다.

and의 fact인 and(0, 0, 0)를 例로 설명하면 첫번째 0은 AND게이트의 A신호값이고, 두번째 0은 B신호값이며, 세번째 0은 출력값을 의미한다. 다른 게이트에 대해서도 똑같이 적용할 수 있다. 또한 論理回路 上에 나타날 수 있는 AND, NOR, EX-NOR게이트 표현은 프로그램의 데이타베이스로부터 만들어 낼 수 있다.

프로그램 2는 출력단에 inverter가 추가되는 경우에 얻어지는 gate에 관한 rule이다.

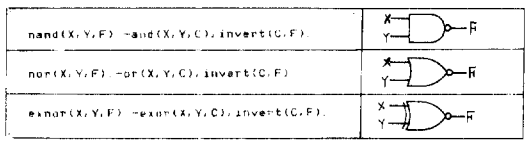
프로그램 1과 프로그램 2를 이용하면 프로그램 상에서의 論理回路의 시뮬레이션은 물론 故障檢出을 위한 고장검출 set를 생성시킬 수 있다.

먼저 論理回路를 PROLOG로 표현할 수 있는 方法을 예제로 설명한다.




**프로그램 1.** 기본논리게이트에 대한 PROLOG 데이타베이스

**Prog. 1.** PROLOG data base for logic gates.




**프로그램 2.** PROLOG에 의한 NAND, NOR, EX-게이트의 표현.

**Prog. 2.** Representation of gates (NAND, NOR, EX-NOR) by using PROLOG.

예제) PROLOG를 사용한 論理回路의 표현

그림 7과 같은 論理回路는 프로그램 3과 같이 구성될 수 있다. 이 때 사용되는 論理게이트의 데이타베이스는 프로그램 1과 프로그램 2에 나타나 있다. 사실 각 論理게이트 사이의 연결 관계를 PROLOG로 표현한 것 뿐이지만 이를 효과적으로 사용하면 어떠한 論理回路도 PROLOG로 표현할 수 있게 된다.

PROLOG를 사용하여 論理的으로 표현된 論理回路를 어떤 입력에 대해 출력을 얻도록 구동시키려면 입력의 가능한 조합을 만들어내는 입력 생성용 rule이 필요하다. 대부분의 論理回路가 2進(bina-

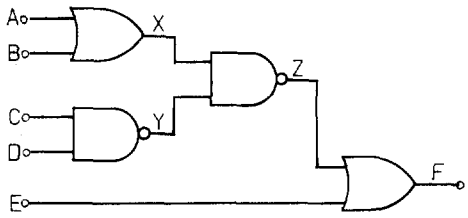



**그림 7**. 조합한 論理回路의 例
**Fig. 7.** Example of combinational logic circuit.

ry) 값을 취하기 때문에 PROLOG의 backtracking을 이용하여 만들어낼 수 있다.

프로그램 4는 입력의 가능한 조합을 출력시키는 PROLOG의 프로그램이다. 여기서A, B, C, D,E는 입력변수를 의미한다.

論理回路가 정상출력을 갖는 경우 프로그램 1과 프로그램 2에서 다룬 데이타베이스를 사용하여 프로그램 5를 실행시키면 출력을 만들어낼 수 있다. 이 때의 PROLOG의 backtracking과 pattern matching이 효과적으로 동작한다. 정상출력일 때만 다루는 경우는 별로 없기 때문에 비정상출력을 위한 論理게이트의 데이타베이스구축도 필요하다. 비정상출력은 pdcf와 pdc로 구해지기 때문에 pdcf와 pdc 에 관한 데이타베이스를 만들면 된다.

```
circuit(A,B,C,D,E,F) :-
                        or(A,B,F),
                        nand(C,D,Y),
                        nand(X,Y,Z),
                        or(Z,E,F)
```

**프로그램 3**. 그림 7 의 PROLOG프로그램
**Prog. 3.** PROLOG program for Fig. 7.

```
truthtable(A,B,C,D,E) :-
                        logic(A),
                        logic(B),
                        logic(C),
                        logic(D),
                        logic(E)

logic(0).
logic(1)
```

**프로그램 4**. 모든 입력조합을 출력시키는 PROLOG 프로그램
**Prog. 4.** PROLOG program for generating all the combination of inputs.

```
gentable :-
        truthtable(A,B,C,D,E),
        circuit(A,B,C,D,E,F),
        write(A),
        write(B),
        write(C),
        write(D),
        write(E),
        display(' '
        write(F),
        nl,fail.
gentable :-display(' THE END OF TRUTH-TABLE GENERATION ').
```

**프로그램 5**. 출력발생용 PROLOG 프로그램
**Prog. 5.** PROLOG program for generating output.

D와 $\bar{D}$의 표현은 PROLOG에서는 대문자의 사용에 제한을 받기 때문에 atom은 d와 r로 나타내기로 한다. 프로그램 6 은 비정상출력을 위한 論理게이트의 pdc와 pdcf를 PROLOG로 표현한 것이다.

pdc와 pdcf에 해당하는 데이타베이스를 사용하여 PROLOG의 rule에서 pattern matching하면 D交差가 필요 없게 되는 장점이 있다. 따라서 고장검출하려는 論理回路만 PROLOG로 기술된다면 PROLOG의 backtracking에 의해 fault생성용 set를 구할 수 있다.

프로그램 7 은 그림 7 의 論理回路에 대한 fault set generation을 위한 PROLOG program의 일부이다.

```
and(1,1,d).
and(0,1,r).
and(1,0,r).
and(0,0,r).

or(1,0,d).
or(0,1,d).
or(0,0,r).
or(1,1,d).

invert(0,d).
invert(1,r).

exor(0,1,d).
exor(1,0,d).
exor(0,0,r).
exor(1,1,r).
```

(a) 논리게이트의 pdc

```
and(d,1,d).
and(1,d,d).
and(r,1,r).
and(1,r,r).

or(0,d,d).
or(d,0,d).
or(0,r,r).
or(r,0,r).

invert(d,r).
invert(r,d).

exor(d,1,r).
exor(1,d,r).
exor(r,1,d).
exor(1,r,d).
exor(0,d,d).
exor(0,r,r).
exor(d,0,d).
exor(r,0,r)
```

(b) 논리게이트의 pdcf

**프로그램 6**. 비정상 출력을 위한 논리게이트의 pdc와 pdcf의 PROLOG 프로그램
**Prog. 6.** PROLOG program for database of pdc and pdcf.

```
testset(A,B,C,D,E,F):-
                        gen(A,B,C,D,E,X,Y,Z,F),
                        or(A,B,X),
                        nand(C,D,Y),
                        nand(X,Y,Z),
                        or(Z,E,F),
                        display(' fault location is XYZF'),nl,
                        display(
                        write(X),write(Y),write(Z),write(F),
                        display('

gen(A,B,C,D,E,X,Y,Z,F):-
                          logic(A),logic(B),logic(C),logic(D),logic(E),
                          testvalue(X),testvalue(Y),testvalue(Z),
                          testvalue(F).

testvalue(0).
testvalue(1).
testvalue(d).
testvalue(r).
```

**프로그램 7.** Fault testset 생성을 위한 PROLOG 프로그램

**Prog. 7.** PROLOG program for generating the fault test set.

최종적으로 論理回路의 각 선상에 나타나는 신호값들을 표로 나타내기 위해 프로그램 8 을 작성하여 볼 수 있으며 이 표에 의해서 fault signal의 전달을 확인할 수 있다.

입력조합은 faultgen.에서 처리되어 출력되고, 論理回路 내부의 값은 testset에서 출력된다. faultgen. 을 실행시킨 결과를 하나 들어 예로 설명한다.

fault location is XYZF=011d, ABCDEF=0000 0d의 출력은 내부신호값이 X=0, Y=1, Z=1, F=d임을 나타내고, 입력신호는 A=0, B=0, C=0, D=0, E=0 이다. 그리고 fault location is 011d 00000d의 경우에 대해서 fault가 제대로 검출되었는지 그림 8 에서 확인해 보기로 한다.

```
faultgen:-
            testset(A,B,C,D,E,F),
            write(A),
            write(B),
            write(C),
            write(D),
            write(E),
            display('    '),
            write(F),nl,fail.
```

**프로그램 8.** 최종적인 fault testset 생성을 위한 PROLOG 프로그램

**Prog. 8.** Final PROLOG program for generating the fault testset.

그리고 최종적인 fault test set를 위한 PROLOG 프로그램을 실행한 결과를 나타내면 그림 9와 같다.

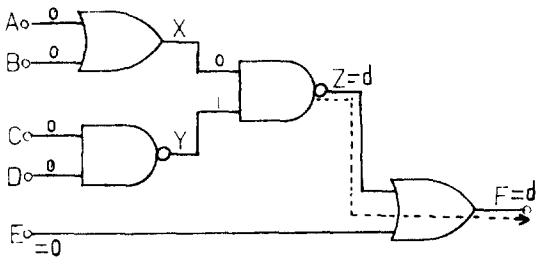



**그림 8.** 출력Z에 대한 활성화
**Fig. 8.** Sensitized path for Z.

```
?- faultgen
fault   location   is   1011   00000   1
fault   location   is   101d   00000   d
fault   location   is   10dd   00000   d
fault   location   is   10dd   00000   d
fault   location   is   1000   00001   0
fault   location   is   100r   00001   r
fault   location   is   10rr   00001   r
fault   location   is   10rr   00001   r
fault   location   is   1rOr   00001   r
fault   location   is   1000   00010   0
fault   location   is   100r   00010   r
fault   location   is   10rr   00010   r
fault   location   is   10rr   00010   r
                .      .      .       .
                .      .      .       .
                .      .      .       .
                .      .      .       .
                .      .      .       .
                .      .      .       .
                .      .      .       .
                .      .      .       .
fault   location   is   rrOr   11110   r
fault   location   is   rrOr   11110   r
fault   location   is   0000   11111   0
fault   location   is   000r   11111   r
fault   location   is   00rr   11111   r
fault   location   is   0rOr   11111   r
fault   location   is   rrOr   11111   r
fault   location   is   rrOr   11111   r
```

**그림 9.** 프로그램 8의 실행 결과
**Fig. 9.** Result of executing program 8.

## 4. 結 論

Roth에 의해 제안된 D-algorithm이 발표된 후 많은 연구가 수행되어 개선된 D-algorithm들이 많이 발표되었으나 本 論文에서는 D-algorithm의 기본 개념만을 도입하여 새로운 fault testset 생성을 위한 방법을 소개하였다. 사용된 PROLOG언어는 제어구조가 backtracking과 pattern matching이 용이하도록 設計되어 D-algorithm의 처리가 몇개의 predicate로 작성될 수 있다. 따라서 D-algorithm상에서 작성되는 pdc, pdcf, sc 모두가 하나의 data로 PROLOG내에서 구축되어 여러 절차를 행하지 않고도 곧 바로 testset를 생성시킬 수 있는 장점이 있다. 또한 backtracking이 故障回路의 출력단부터 시작되기 때문에 backtracking의 횟수를 감소시킬 수 있고 PROLOG를 사용한 論理回路 표현과 일치시킬 수 있어서 효율적이다.

設計된 프로그램은 回路 내부신호 뿐만 아니라 입력 및 출력에 관한 모든 정보를 출력시켜 주므로 回路를 검사할 때 유용한 표로 사용할 수 있는 장점이 있다. 또한 backtrack과 backtracking 결과를 표로 부터 쉽게 확인할 수 있어서 具現하려는 論理回路의 고장상황을 쉽게 파악할 수 있다.

本 論文에서 사용된 論理回路는 조합논리회로(combinationl logic circuit)이지만 비동기식순차논리회로(Asynchronous Sequential Logic Circuit) 에도 적용이 가능하다. 단, 입력 수의 증가에 따라 프로그램을 일부 수정하여야 하며 논리회로를 PROLOG로 다시 표현하여야 한다. 디지탈 시스템의 설계에 있어서 gate의 Fan-in수가 2개 이상일 경우가 많으므로, 이 경우 해당 gate에 대한 새로운 database(pdc, pdcf 및 sc용)를 작성하여야 할 것이다. 그리고 MSI레벨 이상의 IC에 대해서도 고장검출을 위한 새로운 정보가 만들어진다면 PROLOG를 사용하여 보다 높은 수준의 고장검출을 할 수 있을 것이다.

## 參 考 文 獻

1) A.R. VIRUPAKSHIA AND V.C. PRATAPA REDDY, "A Simple Random Test Procedure for Dectection of Single Intermittent Fault in Combinational Circuits," IEEE TRANSACTIONS ON COMPUTERS, VOL. C-32, NO. 6, PP. 594—597, JUNE 1983.

2) TERUHIKO YAMADA AND TAKASHI NANYA,

"Stuck-At Fault Tests in the Presence of Undetectable Bridging Faults", IEEE TRANSACTIONS ON COMPUTERS, VOL. C-33, NO. 8, PP. 758—761, AUGUST 1984.

3) K.K. SALUJA AND S.M. REDDY, "Fault Detecting Test Sets for Reed-Muller Canonic Networks", IEEE TRANSACTIONS ON COMPUTERS, PP. 995—998, OCTOBER 1975.

4) 後藤滋樹, "PROLOG 入門," サイエンス社, 1984, PP. 20—43.

5) ARTHUR D. FRIEDMAN, "Diagnosis of Short-Circuit Faults in Combinational Circuits", IEEE TRANSACTIONS ON COMPUTERS, PP. 746—752, JULY 1974.

6) PRABHAKAR GOEL, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE TRANSACTIONS ON COMPUTERS, VOL. C-30, NO. 3, PP. 215—222, MARCH 1981.

7) HIDEO FUJIWARA, TAKESHI SHIMONO, "On the Acceleration of Test Generation Algorithms", IEEE TRANSACTIONS ON COMPUTERS, VOL. C-32, NO. 12, PP. 1137—1144, DECEMBER 1983.

8) J.P. Roth, "Diagnosis of automata failure: A calculus and a method," IBMJ. Res. Develop., Vol. 10, PP. 278—291, JULY 1966.

9) D.M. MILLER AND J.C. MUZIO, "Spectral Fault Signatures for Single Stuck-At Faults in Combinational Networks", IEEE TRANSACTIONS ON COMPUTERS, VOL. C-33, NO. 8, PP. 765—769, AUGUST 1984.

10) 安部憲広, "Prolog プログラシング入門," 共立出版株式会社, 1985, PP. 126—140.