DNF 논리식에 대한 효율적인 반증 알고리즘

(An Efficient Falsification Algorithm for Logical Expressions in DNF)

문 교 식 †

(Gyo Sik Moon)

요 약 항진을 반증하는 문제는 항진을 검증하는 문제와 같은 난이도를 갖기 때문에 반증(또는 부당성 검증)을 위한 다항식 시간 알고리즘은 가능하지 않다. 이제까지의 그러한 알고리즘들은 분할 및 정복이나 그래프 표현에 기반 한 것이 대부분이다. 대부분의 알고리즘들은 어떤 제약조건 하에서 다양한 입력에 대한 만족할 만한 결과를 보여 주었다. 그러나, 규모가 큰 입력에 대하여 이들 알고리즘들은 어려움을 경험하고 있다. 이 논문에서는 DNF(선언표준형)로 표현된 입력식을 만족하지 못하는 최소항을 구성함으로써 반례를 산출하는 병합 규칙 기반의 새로운 반증 알고리즘을 제안한다. 또한, 제안된 알고리즘의 일관성과 건전성을 증명한다. 제안된 알고리즘은 반증 과정의 각 단계에서 이루어진 할당을 통하여 반증된 항의 수를 최대화하도록 greedy 방법을 기반으로 하고 있다. 실험 결과는 큰 입력의 무작위 비항진 문제 사례들을 반증하는 실용적 성능을 보여 주며 $O(nm^2)$ 시간을 소비한다. 여기서 n은 변수의 개수이고 m은 항의 개수이다.

Abstract Since the problem of disproving a tautology is as hard as the problem of proving it, no polynomial time algorithm for falsification(or testing invalidity) is feasible. Previous algorithms are mostly based on either divide-and-conquer or graph representation. Most of them demonstrated satisfactory results on a variety of input under certain constraints. However, they have experienced difficulties dealing with big input. We propose a new falsification algorithm using a Merge Rule to produce a counterexample by constructing a minterm which is not satisfied by an input expression in DNF(Disjunctive Normal Form). We also show that the algorithm is consistent and sound. The algorithm is based on a greedy method which would seek to maximize the number of terms falsified by the assignment made at each step of the falsification process. Empirical results show practical performances on big input to falsify randomized nontautological problem instances, consuming $O(nm^2)$ time, where n is the number of variables and m is the number of terms.

1. Introduction

The falsification problem is to decide, given an expression, whether it is a tautology or not. The difficulty arises from the fact that the problem involves satisfiability problems or tautology checking problems which are known to be NP-complete[1]. Algorithms have been developed to solve this type of problem, which can be grouped into a few categories: (i) divide-and-conquer and backtracking algorithms[2, 3] (ii) graph-based algorithms[4, 5, 6, 7] (iii) logic programming approaches[8, 9]. Most of the algorithms have shown practical performances on a variety of input expressions subject to input constraints. However, they have shown difficulties dealing with big input. The complexity of the problem depends on the number of input variables in which the search space grows exponentially as the number of variables increases. The input is considered quite big in the community when the number of variables is 30 because there can be 2^{30} states at worst cases. The falsification algorithm may not be

† 정 회 원 : 대구교육대학교 전산교육과 교수
gmoon@taegu-c.ac.kr
논문접수 : 2000년 7월 3일
심사완료 : 2001년 7월 10일

directly compared with other algorithms because of the scarcity of falsification results. Instead, results on the falsification algorithm can be compared with those from previous works on tautology-related problems which come up with time/space explosion when input is big, stated in[2, 4, 6, 10, 11]. This paper presents an efficient falsification algorithm showing superior performance on large scale input expressions using reasonable amount of computing resources.

Preliminaries

We use truth-functional propositional connectives as usual. We define individual *variables* as $x_1, \cdots, x_n$, each of which value is either true or false. A *literal* is a variable or the negation of a variable. A *term* is a conjunction of literals. A *minterm* is a conjunction of n variables with each variable being negated or non-negated. A term can be represented by a string of symbols: $a_1, \cdots, a_n$, where

$$a_i\ (1 \le i \le n) = \begin{cases} 1: & x_i \text{ occurs in the term} \\ 0: & \overline{x_i} \text{ occurs in the term} \\ d: & \text{neither } \overline{x_i} \text{ nor } x_i \text{ occurs in the term.} \end{cases}$$

We assume that both x_i and $\overline{x_i}$ do not occur in a term. For example, $\overline{x_2}x_3\overline{x_5}$ is represented by $d01d0$, when $n=5$. A string composed of $\{0,1,d\}$ is to be regarded as a term or a disjunction of minterms for the term depending on the context. We use $\alpha[i]$ to represent a_i.

Definition 1. A *unit term* is a term which contains a single literal in it. A *complete term* with n variables consists only of n occurrences of the symbol d, denoted by σ_n, which is often used to represent a tautology. □

Definition 2. Let α, β be any arbitrary terms. The position j of α, β is called a *complementary position* iff $\alpha[j] \neq d$, $\beta[j] \neq d$, and $\alpha[j] = 1-\beta[j]$ (or, $\beta[j] = 1-\alpha[j]$). α and β are called *complementary terms* iff there is a complementary position j and for any other position k, $\alpha[k] = \beta[k]$. (e.g., $0d1d$ and $0d0d$ are complementary.) A pair of terms is said to be *contradictory* iff it contains two complementary unit terms. □

The Quine-McCluskey method[12, 13] was the first algebraic approach to produce a simplified expression for a Boolean function. The tabulation method starts from the list of minterms that specify a Boolean function and are simplified by combining two complementary terms. It repeats the process until new simplification is not possible. The Merge-Rule introduced in this section has a similar idea of combining two terms. But two terms need not be complementary to be merged.

Definition 3. A *merge* γ of two terms α and β is a term such that (1) α and β have exactly one complementary position and (2) $\gamma[i]$, for each position i, is defined by the *Merge Rule* shown in Table 1. γ is called an *immediate consequence* of two terms α and β. The *merge position* of γ is the position j in which $\alpha[j]$ and $\beta[j]$ are complementary and $\gamma[j] = d$. □

Definition 4. Let Γ be a nonempty set of terms to represent a Boolean expression in DNF (*disjunctive normal form*). (Γ will be used as such throughout the paper.) Let α and β be two terms of Γ. The *projection of* α *with respect to* β, denoted by $\alpha^{\Pi}|_{\beta}$, is $\alpha^{\Pi}[j]|_{\beta} = \alpha[i]$, where i is the jth position of β whose value is d. The *projection of* Γ *with respect to* β, denoted by $\Gamma^{\Pi}|_{\beta}$, is the set of projections $\{\alpha_1^{\Pi}|_{\beta}, \cdots, \alpha_m^{\Pi}|_{\beta}\}$, where $\alpha_i^{\Pi}|_{\beta}$ is the projection of α_i with respect to β for each $\alpha_i (\neq \beta) \in \Gamma$. We abbreviate $\alpha^{\Pi}|_{\beta}$, $\beta^{\Pi}|_{\beta}$, $\Gamma^{\Pi}|_{\beta}$, to α^{Π}, β^{Π}, Γ^{Π} respectively if there is no ambiguity. □

Example 1. Let $\Gamma = \{ddd11, 1ddd1, d001d, 0d1d1, 100dd\}$ and $\beta = d0d1d$. By applying the projection rule, we obtain, $\beta^{\Pi} = ddd$ and $\Gamma^{\Pi}|_{\beta} = \{dd1, 1d1, d0d, 011, 10d\}$. □

2. Falsification Algorithm

The goal of a falsification algorithm is to find a minterm $\hat{t}$ which falsifies Γ. A falsification

algorithm is said to be *consistent* iff the value of $\hat{t}[j]$ for a position j will remain unchanged throughout the falsifying process once it is assigned. A falsification algorithm is said to be *sound* iff any minterm returned from the algorithm guarantees to falsify the input. A falsification algorithm is said to be *complete* iff the algorithm guarantees to construct a falsifying minterm for any nontautological expression.

To show that Γ is not a tautology, it is enough to find a minterm such that it cannot be satisfied by Γ. To do that, we design a falsification algorithm named Algorithm F which tries to construct a falsifying minterm in the linear time under some conditions stated in Lemma 1, which is a preliminary work to provide the major part of the Algorithm Falsify.

Algorithm F

Input: a set, Γ, of terms, satisfying
(1) for each term $\alpha \in \Gamma$, $\alpha \neq \sigma_n$, where n is the number of variables, and
(2) for each pair of terms $\alpha, \beta \in \Gamma$, α and β cannot be merged.
Output: a minterm $\hat{t}$ which is not satisfied by Γ.
Procedure:
  $\Delta_0 = \Gamma$;
  $\hat{t}[1..n] \leftarrow 0$;
  $k \leftarrow 0$;
  while Δ_k *is not empty do*
    $k \leftarrow k+1$;
    $A \leftarrow \{\alpha \mid \alpha \in \Delta_{k-1}, \alpha[k]=1\}$;
    if A *is not empty, then* $\hat{t}[k] \leftarrow 0$;
      else begin
        $A \leftarrow \{\alpha \mid \alpha \in \Delta_{k-1}, \alpha[k]=0\}$;
        $\hat{t}[k] \leftarrow 1$;
      end;
    end.if;
    $\Delta_k \leftarrow \Delta_{k-1} - A$;
  end.while;
  return $\hat{t}$;
end.procedure;

Example 2. Let $\Gamma = \{d0dd1d01d,\ 1dd0d1011,\ dd001dd1d,\ dd00ddd11,\ 1dd011ddd,\ 1ddddd011,\ 0111d01dd\}$. Table 2 shows the trace of Algorithm F on Γ for constructing $\hat{t}$. □

Lemma 1. Let Γ be a set of terms. The Algorithm F returns $\hat{t}$ if (1) for each term $\alpha \in \Gamma$, $\alpha \neq \sigma_n$, and (2) for each pair of terms $\alpha, \beta \in \Gamma$, α and β cannot be merged.

Proof. The algorithm removes members of A from Δ_{k-1} at each iteration whenever $A \neq \emptyset$. Thus as long as $A \neq \emptyset$, Δ_{k-1} will be reduced to Δ_k whose cardinality is less than that of Δ_{k-1}. Then, it is possible that Δ_k becomes $\emptyset$ at kth iteration $(1 \leq k < n)$. Hence, the algorithm terminates and returns $\hat{t}$. If $A = \emptyset$ at kth iteration $(1 \leq k < n)$, then we proceed k to the next position. Suppose we have just reached the last iteration, where $k = n$. Then, it is sufficient to show that $\Delta_n = \emptyset$. Suppose Δ_n is not empty. Then, there exists a term $\alpha \in \Delta_n$ and we observe: (i) there is no $j \leq n$ such that $\alpha[j] = 1$ because otherwise $\alpha \notin \Delta_j$ (i.e., α would be excluded at step j). (ii) there is some $j \leq n$ such that $\alpha[j] = 0$ because otherwise $\alpha = \sigma_n$. So, let j be the last $j \leq n$ such that $\alpha[j] = 0$. Then, there must be another term $\beta \in \Delta_{j-1}$ such that $\beta[j] = 1$. For all $i < j$, $\beta[i] \neq 1$, otherwise $\beta \notin \Delta_{j-1}$ because β would be excluded earlier. For all $j < k \leq n$, $\alpha[k] = d$. So, $\alpha[j] = 1 - \beta[j]$ and there are no other complementary positions except j. Then, α and β can be merged. A contradiction. □

We present a *sound* algorithm which can be used efficiently for most of the problem instances, although it may not be complete because of the NP-completeness of the falsification problem. The algorithm is based on a greedy method designed to find an assignment maximizing the number of terms falsified by the partial assignment made at each step of choosing a variable and its truth value. This can be done by choosing a merge position which is used most frequently and by assigning the negation of the truth value which takes place most frequently on the position. Experimental results show that the algorithm works efficiently for most cases.

Algorithm Falsify

Input: *a set, Γ, of terms.*
Output: *(1) returns a falsifying minterm $\hat{t}$ if it is found.*
(2) returns 'Fail', otherwise.
Procedure:
P0. { Initialization of global data structures }
Let $x_1, \cdots, x_n$ *be the variables.*
Let $S=\{(\alpha,\beta) \mid \alpha,\beta\in\Gamma,\ \alpha$ and β *can be merged}*
$R\leftarrow\emptyset$;
$\hat{t}[1..n] \leftarrow d$;
P1. { Main Body }
repeat
if there exists a unit term $\alpha\in S$,
then begin
Let j *be the position where* $\alpha[j]\neq d$.
$\hat{t}[j]\leftarrow 1-\alpha[j]$;
end;
else begin
Let j *be the position with the smallest index at which merge positions take place most frequently for all pairs in* S.
Let $N_j(w)$ *be the number of occurrences in* S *of* $w(=0$ or $1)$ *at position* j *of distinct terms of* S.
if $N_j(0) > N_j(1)$, *then* $\hat{t}[j]\leftarrow 1$; *else* $\hat{t}[j]\leftarrow 0$; *end.if;*
end;
end.if;
Perform P3 using $\hat{t}[j]$;
until (S is empty) or (S contains a contradictory pair);
if S contains a contradictory pair, then return 'Fail';
else if R is empty, then return $\hat{t}$; *end.if;*
else goto P2;
end.if;
P2. { Call Algorithm F }
Let $R^{\Pi}\mid_{\hat{t}} = \{\ \alpha^{\Pi}\mid_{\hat{t}} \mid\ \alpha\in R\}$.
Call Algorithm F for constructing a minterm t^* *which falsifies* $R^{\Pi}\mid_{\hat{t}}$.
for each position j *of* t^*,
$\hat{t}[l]\leftarrow t^*[j]$, *where* l *is the* j*th position of* $\hat{t}$ *whose value is* d.
end.for;
return $\hat{t}$;
P3. { Reductions }
Perform reductions on S *using* $\hat{t}[j]$ *as follows:*
for each term α *of* S,
if $\alpha[j]=\hat{t}[j]$, *then* $\alpha[j]=d$; *end.if;*
if $\alpha[j]=1-\hat{t}[j]$, *then remove* α *from* S; *end.if;*
end.for;
for each member $s\in S$,
if s *contains a single term* η,
then begin
$S\leftarrow S-\{s\}$;
$R\leftarrow R\cup\{\eta\}$;
end;
end.if;
end.for;
Remove each member of R *from* R *if it appears in a pair of* S.
Perform reductions on the set R *using* $\hat{t}[j]$ *as follows:*
for each term $\alpha\in R$,
if $\alpha[j]=\hat{t}[j]$, *then* $\alpha[j]=d$; *end.if;*
if $\alpha[j]=1-\hat{t}[j]$, *then remove* α *from* R; *end.if;*
end.for;
end.procedure;

Example 3. Let $\Gamma = \{d1dd1,\ 01ddd,\ ddd00,\ dd0d1,\ d01dd,\ 1ddd0\}$. Table 3 shows the trace of Falsify on Γ. The procedure falsifies the set by making five assignments of truth values to $\hat{t}$ in which two assignments are made by Algorithm F at P2. □

It is obvious that Falsify is consistent. Also it terminates because S becomes empty by the reduction operation at P3 by $\hat{t}[j]$ or S contains a contradictory pair. We show that the existence of a contradictory pair of unit terms is the necessary and sufficient condition for the failure of constructing $\hat{t}$.

Observation 1. If two complementary unit terms are generated by Falsify, then the two terms must be paired in S.

Proof. If they were not paired, then they could not be merged. Then, there must be another disjoint position j, where both literals on j must have been reduced to d's by Falsify, which is impossible. □

Upon completion of P1, if both R and S are empty, then we conclude that $\hat{t}$ falsifies Γ. If S is empty but R is not, R contains those terms that have not yet been falsified by $\hat{t}$. We then perform the next part of falsification. Right before P2 is initiated, the following facts are understood:

Observation 2. For each term $\alpha \in R$, (1) $\alpha \neq \sigma_n$, (2) $\forall p\ [\ \hat{t}[p] \neq d \Rightarrow \alpha[p] = d\ ]$, and (3) $\exists p[\ \hat{t}[p] = d$ and $\alpha[p] \neq d\ ]$. □

Observation 3. The remaining terms of the set R cannot have a merge among them when P2 is initiated.

Proof. Let α', β' be any two remaining terms of R when P2 is initiated. Let α (β) be the original term for α' (β') respectively. If α and β can be merged, then the pair (α, β) would be removed from S by a reduction at P3 and at least one of α' and β' would be removed from R before P2 is initiated. So, α and β cannot be merged. If α and β have no complementary position, then α' and β' cannot be merged. If α and β have two or more complementary positions, there are two cases to be considered: (i) If there is a complementary position p such that $\hat{t}[p] \neq d$, then one of α' and β' would be removed from R before P2 is initiated. (ii) If, for each complementary position p, $\hat{t}[p] = d$, then α' and β' cannot be merged. □

Observation 4. At P2, We observe: (1) For each $\alpha^{II}|_{\Gamma} \in R^{II}|_{\Gamma}$, $\alpha^{II}|_{\Gamma}$ is not a complete term. (2) Each pair of terms of $R^{II}|_{\Gamma}$ cannot be merged. □

Theorem 1. (Soundness) If Falsify returns $\hat{t}$, then Γ does not satisfy $\hat{t}$.

Proof. Directly from Observations 1 to 4 and Lemma 1. □

We believe that Falsify is not complete because the problem of searching a falsifying minterm is as hard as the original problem of checking the tautologyhood in general, while the time complexity of Falsify is only $O(nm^2)$, where n is the number of variables and m is the number of terms of Γ. So, 'Fail' returned from the algorithm does not necessarily imply that Γ is a tautology.

3. Experimental Results

The Falsify algorithm was implemented in C on an HP 9000 workstation to test the efficacy of the algorithm on a wide range of random expressions. The results demonstrate practicality and efficiency of the algorithm. The algorithm is especially strong on big inputs with more than 20 variables.

1. Parameterized random sets of problem instances

The popular model[6, 11] was used to construct random sets of problem instances in DNF. The popular model generates random expressions using the parameters: v(the number of variables), t(the number of terms), and p(the probability that each literal appears in a term). A *term* is generated randomly by independently selecting the $2v$ literals with probability p. A random expression is generated randomly by independently forming t terms excluding meaningless instances such as null,

contradictory and duplicate terms. The number of variables is the main cause for the exponential growth of search space. If the number of variables is less than 20, it would not be big enough to test the efficiency of the algorithm. If the number is greater than 50, it would be too big to be practical. The values of v are so chosen: 20, 30, 40, and 50. The sparsity which tells the density of literals in a term depends directly on the value of the probability p: We pick three values of p for generating random terms of different sparsity: 0.2 for sparse, 0.5 for medium, and 0.9 for dense terms. And finally, the number of terms, t, is tested on the following values: 1000, 2000, 3000, 4000, and 5000. So, there are 60 different cases in total utilizing the three parameters. And ten random instances are generated for each case in order to obtain unbiased characteristics of input. Thus, a total of 600 random expressions are prepared for the experiment. The analysis of the random data shows that 110 out of 600 cases are found to be tautologies and the rest(490 cases) are likely to be nontautologies. Table 4 shows the number of tautologies for each value of v.

2. Results of the algorithm Falsify

The Falsify algorithm was tested independently on a set of 600 randomized problem instances of which 490 instances are likely to be nontautologies. The algorithm was able to falsify 408 instances out of 490(83.2%), which shows a fairly good performance. Table 5 shows the falsification rate for each value of v. Also, the algorithm uses reasonable amount of time for most of the cases.

4. Conclusion

We proposed a new algorithm to falsify expressions in DNF using formal methods. We presented the Algorithm F stating that a set of terms in which there exists no merge among the members of the set according to the merge rule can be falsified by a minterm constructed by the algorithm and we provided the proof of it. Later, we proposed the Falsify algorithm which chooses a most favorable merge position in order to speed up the falsification process and quickly approach the situation where the Algorithm F can be utilized. And we proved both the correctness and soundness of the algorithm. Experiments on the set of randomized problem instances showed that the algorithm works correctly and efficiently for a wide range of big inputs with high falsification rate (83.2%). The idea of the falsification algorithm may be extended to first order logic for further research.

Table 1 The Merge Rule

$\alpha[i]$	0	0	0	1	1	1	d	d	d
$\beta[i]$	0	1	d	0	1	d	0	1	d
$\gamma[i]$	0	d	0	d	1	1	0	1	d

Table 2 Trace of Algorithm F for Example 2

k	1	2	3	4	5
A	*1dd0d1011* *1dd011ddd* *1ddddd011*	*0111d01dd*	*dd001dd1d* *dd00ddd11*	$\emptyset$	*d0dd1d01d*
$\hat{t}[k]$	0	0	1	1	0
Δ_k	*d0dd1d01d* *dd001dd1d* *dd00ddd11* *0111d01dd*	*d0dd1d01d* *dd001dd1d* *dd00ddd11*	*d0dd1d01d*	*d0dd1d01d*	$\emptyset$

Table 3 Traces of Falsifying for Example 3

Proc	*Reductions*	*S*	*R*	$\hat{t}$
P0		{(*d1dd1,ddd00*),(*d1dd1,d01dd*),(*d1dd1,1ddd0*),(*01ddd,d01dd*),(*01ddd,1ddd0*),(*ddd00,dd0d1*),(*dd0d1,d01dd*),(*dd0d1,1ddd0*)}	$\emptyset$	$\hat{t}[1..5] \leftarrow d$
P1		{(*d1dd1,ddd00*),(*d1dd1,d01dd*),(*d1dd1,1ddd0*),(*01ddd,d01dd*),(*01ddd,1ddd0*),(*ddd00,dd0d1*),(*dd0d1,d01dd*),(*dd0d1,1ddd0*)}	$\emptyset$	$\hat{t}[5]=0$
P3	Remove (*d1dd1,ddd00*),(*d1dd1,d01dd*),(*d1dd1,1ddd0*),(*ddd00,dd0d1*),(*dd0d1,d01dd*),(*dd0d1,1ddd0*) from *S*	{(*01ddd,d01dd*),(*01ddd,1dddd*)}	{*ddd0d*}	
P1		{(*01ddd,d01dd*),(*01ddd,1dddd*)}	{*ddd0d*}	$\hat{t}[1]=0$
P3	Remove (*01ddd,1dddd*) from *S*	{(*d1ddd,d01dd*)}	{*ddd0d*}	
P1		{(*d1ddd,d01dd*)}	{*ddd0d*}	$\hat{t}[2]=0$
P3	Remove (*d1ddd,d01dd*) from *S*	$\emptyset$	{*ddd0d*, *dd1dd*}	
P2		$\emptyset$	{*ddd0d*, *dd1dd*}	$\hat{t}[3]=0$ $\hat{t}[4]=1$

Table 4 The counts of tautologies

v	20	30	40	50	Sum
tautoglogies	50	50	10	0	110
likely to be nontautologies	100	100	140	150	490
Total cases	150	150	150	150	600

Table 5 Falsification rate

v	20	30	40	50
run time(sec); avg : max	.17:.5	.75:5.3	15.1:115.9	17.8:116.8
The number of terms likely to be nontautologies	100	100	140	150
The number of terms falsified by the algorithm	70	100	107	131
Falsification rate	70%	100%	76%	87%
Overall falsification rate	408 / 490 = 83.2%			

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freemann, San Francisco, 1979.

[2] P. Lammens, L. Claesen and H. D. Man, "Tautology checking benchmarks: results with TC," *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, Vol. 2, pp.600-604, 1989.

[3] F. Vlach, "Simplification in a satisfiability checker for VLSI applications," *Journal of Automated Reasoning*, Vol. 10, No. 1, pp.115-136, 1993.

[4] J. Jain, J. Bitner, M. S. Abadir, J. A. Abraham, and D. S. Fussell, "Indexed BDDs: Algorithmic Advances in Techniques to Represent and Verify Boolean Functions," *IEEE Trans. on Computer*, Vol. 46, No. 11, pp.1230-1245, 1997.

[5] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, Vol. 24, No. 3, pp.293-318, 1992.

[6] M. Fujita, H. Fujisawa and Y. Matsunaga, "Variable ordering algorithms for ordered binary decision diagrams and their evaluation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 1, pp.6-12, 1993.

[7] J. Lafferty and A. Vardy, "Ordered Binary Decision Diagrams and Minimal Trellises," *IEEE Trans. on Computers*, Vol. 48, No. 9, pp.971-986, 1999.

[8] M. Dincbas, P. V. Hentenryck, H. Himonis, A. Aggoun, T. Graf and F. Berthier, "The constraint logic programming language CHIP," *Proceedings in the International Conference on Fifth Generation Computer Systems FGCS-88*, ToKyo, Japan, 1988.

[9] H. Simonis and T. L. Provost, "Circuit verification in CHIP: Benchmark results," *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, Vol. 2, pp.570-574, 1989.

[10] G. S. Moon, "A Boolean equivalence testing algorithm based on a derivational method," *Journal of Electrical Engineering and Information Science*, Vol. 2, No. 5, pp.1-8, 1997.

[11] P. Purdom, "Random satisfiability problems," *International Workshop on Discrete Algorithms and Complexity, The Institute of Electronics, Information and Communication Engineers*, Tokyo, Japan, pp.253-259, 1989.

[12] E. J. Jr. McCluskey, "Minimization of Boolean functions," *Bell System Tech. J.*, Vol. 35, No. 6, pp.1417-1444, 1956.

[13] W. V. Quine, "The problem of simplifying truth functions," *Am. Math. Monthly*, Vol. 59, No. 8, pp. 521-531, 1952.

문 교 식

1982년 경북대학교 공과대학 컴퓨터공학과(공학사). 1982년 ~ 1986년 KIST 시스템공학연구소, 연구원. 1989년 University of Oklahoma 대학원 전산학과(이학석사). 1995년 University of North Texas 대학원 전산학과(이학박사). 1996년 3월 ~ 1997년 2월 (부산) 동명정보대학교 컴퓨터공학과 조교수. 1997년 3월 ~ 현재 대구교육대학교 전산교육과 전임강사, 조교수. 관심분야는 전산교육, 알고리즘, 인공지능