A Boolean Equivalence Testing Algorithm based on a Derivational Method

Gyo-Sik Moon

Abstract

The main purpose of the Boolean equivalence problem is to verify that two Boolean expressions have the same functionality. *Simulation* has been extensively used as the standard method for the equivalence problem. Obviously, the number of tests required to perform a satisfactory coverage grows exponentially with the number of input variables. However, *formal methods* as opposed to simulation are getting more attention from the community. We propose a new algorithm called *the Cover-Merge Algorithm* based on a *derivational method* using the concept of *cover* and *merge* for the equivalence problem and investigate its theoretical aspects. Because of the difficulty of the problem, we emphasize simplification techniques in order to reduce the search space or problem size. Heuristics based on types of merges are developed to speed up the derivation process by allowing simplifications. In comparison with widely used technique called *Binary Decision Diagram* or BDD, the algorithm proposed outperforms BDD in nearly all cases of input including standard benchmark problems.

I. Introduction

The equivalence problem is to decide, given two Boolean expressions, whether they are logically equivalent. The difficulty arises from the fact that the problem involves Boolean satisfiability problems or tautology checking problems which are known to be NP-complete [8].

Much of the work on the equivalence problem has concentrated on applications in the verification of Boolean circuits. Other application domains can be found in artificial intelligence, databases, etc., where information or knowledge can be described in Boolean expressions and searching (or matching) is involved.

The equivalence problem has been treated in many different ways. However, we can categorize them into a few groups depending on the types of algorithms used: (1) Algorithms based on divide-and-conquer and backtracking [2, 10, 14, 15, 18] (2) Graph-based algorithms [1, 3, 4, 7, 19] (3) Algorithms based on logic programming [5, 6, 16] (4) Algorithms based on counting [9, 17]. None of these works efficiently for every problem instance because of the difficulty of the problem. However, each algorithm can demonstrate a good performance under certain restrictions and problem characteristics. The BDD method proposed by Bryant [4] in 1986 has been most widely used in expressing and manipulating Boolean expressions. It has elegant and powerful properties of representing a canonical form for a Boolean expression and constructing graphs for many practical logic expressions. In order to demonstrate the capability of the algorithm presented in this paper we compared results with those for BDD because of its popularity and practicality.

We design and implement a new algorithm called the Cover-Merge Algorithm for the equivalence problem, based on a derivational method. We prove that the algorithm is sound and complete. Because of the NP-completeness of the problem, we emphasize simplifications to reduce the search space. Heuristics are developed to perform simplifications or to speed up the verification process. Results are compared with those for BDD on standard benchmark problems as well as on randomly generated expressions to perform an unbiased way of testing algorithms. The algorithm has demonstrated fairly stabilized and practical performances for big Boolean expressions.

II. The Derivational Method

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 *clause* (*term*) is a disjunction (conjunction) of literals. A *maxterm* (*minterm*) is a disjunction (conjunction) of n variables with each variable being negated or non-negated. A

Manuscript received December 20, 1996; accepted August 29, 1997.

G. S. Moon is with Department of Computer Education, Taegu National University of Education, Taegu, Korea.

term can be represented by a string of symbols: $a_1\ a_2 \cdots a_n$, where

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

We assume that both x_i and $\bar{x}_i$ do not occur in a term. For example, $\bar{x}_2\ x_3\ \bar{x}_5$ 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 $a[i]$ to represent a_i. Hereafter we abbreviate 'if and only if' to 'iff'.

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. σ_n is often used to represent a tautology. □

Definition 2. Let ϕ, ψ be two nonempty sets of terms to represent Boolean expressions in DNF(*disjunctive normal form*). (ϕ and ψ will be used as such throughout this paper.) ϕ *covers* ψ, denoted by $\psi \in^c \phi$, iff for each term $\beta \in \psi$, any assignment that satisfies β also satisfies a disjunction of some terms of ϕ. □

Definition 3. ϕ is *equivalent* to ψ iff $\phi \in^c \psi$ and $\psi \in^c \phi$. □

Example 1. Let ϕ = { $ddd1$, $d00d$, $0dd0$, $10dd$, $d1d0$ } and ψ = { $d0d1$, $0dd1$, $dd01$, 1111, $dd00$, $dd10$ }. Then ϕ and ψ are equivalent because $\psi \in^c \phi$ and $\phi \in^c \psi$. □

Definition 4. Let α, β be any arbitrary terms.

(1) *Complementary:* The position j of α, β is called a *complementary position* if $\alpha[j] \neq d$, $\beta[j] \neq d$, and $\alpha[j] = 1 - \beta[j]$ (or, $\beta[j] = 1 - \alpha[j]$). α and β are called *complementary terms* if there is a complementary position j and for any other position k, $\alpha[k] = \beta[k]$. (e.g. $0d1d$ and $0d0d$ are complementary.)

(2) *Disjoint:* α, β are called *disjoint terms* if there is a position j such that $\alpha[j]$ and $\beta[j]$ are complementary. We define the *distance* of two disjoint terms as the number of complementary positions. (e.g. $0d1d$ and $110d$ are disjoint with distance = 2.)

(3) *Inclusive:* The position j of α, β is called an *inclusive position* if *either* (a) $\alpha[j] = d$, $\beta[j] = 0$ or 1 *or* (b) $\alpha[j] = 0$ or 1, $\beta[j] = d$. α and β are called *inclusive* if one is covered by the other. $I(\alpha, \beta)$ denotes the number of *inclusive positions* between the two terms α, β. (e.g. $0d1d$ and $011d$ are inclusive.)

(4) *Shared:* α and β are called *shared* if α and β are not *disjoint*. (e.g. $0d1d$ and $01d0$ are shared.) □

1. The Merge Rule and Derivation

The Quine-McCluskey method [11, 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 simplifies by combining two complementary terms. It repeats the process until no new simplification is possible. The Merge-Rule introduced in this section has similar idea of combining two terms. But two terms need not be complementary to be merged.

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

Definition 5. 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, denoted by $\{\alpha, \beta\} \mapsto \gamma$. A *merge position* for $\{\alpha, \beta\} \mapsto \gamma$ is the position j in which $\alpha[j]$ and $\beta[j]$ are complementary and $\gamma[j] = d$. □

There are two special types of merges: *unit merge* and *simple merge*, which are quite useful for simplification.

Definition 6. γ is said to be a *unit merge* of two terms α, β iff they can be merged and at least one of them is a unit term. γ is said to be a *simple merge* of two terms α and β iff they are complementary. □

Definition 7. Δ is called a *derivation* of a term γ from a set of terms Γ iff (1) Δ is a finite sequence of terms, (2) the last term in Δ is γ, and (3) each term of Δ is either a member of Γ or an immediate consequence by the Merge Rule of two terms preceding it in the sequence. $\Gamma \mapsto \gamma$ denotes a derivation of a term γ from a set of terms Γ. $\Gamma \not\mapsto \gamma$ denotes that it is not the case that $\Gamma \mapsto \gamma$. □

Definition 8. A merge γ of two terms from Γ is said to be a *nondegenerating merge* if γ is not covered by any term of Γ. Otherwise, γ is called a *degenerating merge*. □

Definition 9. β is a *generalization* of a term α iff β is the result of replacing zero or more 0's or 1's in α by d's iff for each position i, $\beta[i] = \alpha[i]$ or $\beta[i] = d$. A term α is said to be *derivable* from a set of terms Γ iff there is a derivation of a generalization of α from Γ. □

2. Operations

We define operations on terms, which will be frequently used in subsequent algorithms.

Definition 10. (1) *Difference:* The *difference* of two terms α, β or *relative complement* of β *with respect to* α, denoted by $\alpha \ominus \beta$, is $\{t \mid t$ is a minterm such that $t \in^c \alpha,\ t \notin^c \beta\}$.

(2) *Projection, Complementary Projection:* 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 $\beta^{\Pi}|_{\beta}$ to β^{Π}. The *complementary projection* of α with respect to β, denoted by $\alpha^{-\Pi}|_{\beta}$, is exactly like $\alpha^{\Pi}|_{\beta}$ except that i is the jth position of β whose value is not d. Similarly, we define $\Gamma^{-\Pi}|_{\beta}$ and $\beta^{-\Pi}$. We abbreviate $\alpha^{\Pi}|_{\beta}$, $\Gamma^{\Pi}|_{\beta}$, $\alpha^{-\Pi}|_{\beta}$, $\Gamma^{-\Pi}|_{\beta}$ to α^{Π}, Γ^{Π}, $\alpha^{-\Pi}$, $\Gamma^{-\Pi}$, respectively if there is no ambiguity. □

Example 2. Let $\Gamma = \{ ddd11,\ 1ddd1,\ d001d,\ 0d1d1,\ 100dd \}$ and $\beta = d0d1d$. By applying the projection rule, we obtain, $\beta^{\Pi} = ddd$ and $\Gamma^{\Pi}|_{\beta} = \{ dd1,\ 1d1,\ d0d,\ 011,\ 10d \}$. For complementary projections, $\beta^{-\Pi} = 01$ and $\Gamma^{-\Pi}|_{\beta} = \{ d1,\ dd,\ 01,\ dd,\ 0d \}$. □

3. The Merge-Loop

In this section, we propose the Merge_Loop_1 and provide its proof.

Algorithm Merge_Loop_1

Input: a set, Γ, of terms and a term η.

Output: True if Γ covers η. False, otherwise.

Procedure:

Let $\Gamma^0 = \Gamma$.

$k \leftarrow 0$;

while $\tilde{\eta} \notin \Gamma^k$ ($\tilde{\eta}$ is a generalization of η) do

Let γ^k be a *nondegenerating merge* of two terms from Γ^k.

if Γ^k is not found, **return** False;

Let $\Gamma^{k+1} = (\Gamma^k - \{ \text{term(s) covered by } \gamma^k \}) \cup \{ \gamma^k \}$.

$k \leftarrow k+1$;

end.while;

return True;

end.procedure;

It is obvious that the Merge_Loop_1 terminates because (i) only nondegenerating merges can be added to the system, (ii) a nondegenerating merge cannot be added more than once because of (i), and (iii) the set of nondegenerating merges is finite.

Theorem 1. The Merge_Loop_1 returns *true* iff $\eta \in^c \Gamma$.

Proof. *Part A. (Soundness Proof) If η is derivable from Γ^0 by the Merge_Loop_1, then Γ^0 covers η.* We prove the following theorem by induction on the length l of a derivation of $\tilde{\eta}$ from Γ^0, where $\tilde{\eta}$ is a generalization of η: If there is a derivation of $\tilde{\eta}$ from Γ^0, then Γ^0 covers $\tilde{\eta}$. If l=0, then the theorem immediately follows from $\eta \in \Gamma^0$. Suppose the theorem holds for $l = k-1$ ($k \geq 1$). If $l = k$ and $\{x, y\} \mapsto \tilde{\eta}$, then, by the hypothesis, Γ^0 covers both x and y. Then, Γ^0 covers $\tilde{\eta}$.

Part B. (Completeness Proof) If Γ^0 covers η, then η is derivable from Γ^0 by the Merge_Loop_1. We prove the following theorem by induction on l (the number of d's in η): If η contains exactly l d's, and Γ^0 covers η, then η is derivable from Γ^0, ($\eta \notin \Gamma^0$, $0 \leq l \leq n$). *1. Basis:* $l = 0$. Then, there is an immediate derivation $\Gamma^0 \mapsto \tilde{\eta}$, where $\tilde{\eta}$ is a *generalization* of η. *2. Hypothesis:* If η contains exactly ($l-1$) d's and Γ^0 covers η, then η is derivable from Γ^0. *3. Induction Step:* Let $r_1, \cdots, r_l$ be the positions where d appears in η. Consider two terms α and β such that they are exactly like η except $\beta[r_j] = 1 - \alpha[r_j] \neq d$ for any $1 \leq j \leq l$. Then, by the hypothesis, β (α) is derivable from Γ^0 if $\eta \in^c \Gamma^0$. Let $\tilde{\beta}$ ($\tilde{\alpha}$) be a generalization of β (α), derived from Γ^0. Then, if $\tilde{\beta}$ or $\tilde{\alpha}$ is a generalization of η, then the theorem immediately follows. If not, $\tilde{\beta}[r_j] = 1 - \tilde{\alpha}[r_j] \neq d$. Then, $\{\tilde{\beta}, \tilde{\alpha}\} \mapsto \tilde{\eta}$. □

If η is a complete term, then the Merge-Loop conducts a special type of cover testing, namely *tautology checking*.

Example 3. Let $\Gamma^0 = \{ d11d1,\ 01ddd,\ ddd00,\ 0d01d,\ dd0d1,\ d01dd,\ 1dd10 \}$. Direct application of the algorithm requires 34 steps to complete the process. □

It is interesting to observe that only 13 merges are essential to the derivation of σ_5 in the above example, indicating many redundancies are involved. The bottleneck is the merge process which blindly generates all nondegenerating merges. The focalization of next section will be mainly on how these redundancies can be eliminated.

III. Simplifications during the Merge-Loop

Without simplifications, the Merge-Loop would soon be overwhelmed by enormous search space for even a small size input.

Definition 11. A merge, $\{\alpha, \beta\} \mapsto \gamma$, is called (1) a *forward merge* if γ covers α or β (or both). (2) a *backward merge* if it is not *forward* and $I(\alpha, \beta) \geq 3$.[1] □

Obviously, a forward merge can be viewed as a simplification. So, it should be given a higher priority than backward merges. Suppose a set of terms Γ covers a term β. Then, it is possible that some terms of Γ may not be essential to the cover. We will show that removing this type of redundancy not only gives a means for simplification, but provides some of the crucial properties for the proofs of the subsequent theorems for our equivalence testing algorithm.

Definition 12. A set of terms Γ is said to be a *minimal cover* for a single term β iff (1) Γ covers β and (2) no proper

1) $I(\alpha, \beta)$ denotes the number of inclusive positions of α and β (Definition 4).

subset of Γ covers β. □

We now show that there exists a subset of Γ which is a minimal cover for a term β, provided that Γ covers β.

Theorem 2. If Γ covers β, then there exists a subset Γ' of Γ such that Γ' is a minimal cover for β.

Proof. Immediately follows from a proof by the construction of a minimal cover. □

Definition 13. A minterm t of a term α is said to be *unique* with respect to a set of terms Γ iff $t \in {}^c\alpha$ and there is no $\beta \in \Gamma - \{\alpha\}$ such that β covers t. $\breve{U}(\alpha)$ denotes the set of all minterms of α that are unique with respect to Γ. A term $\alpha \in \Gamma$ is called *unique* with respect to Γ iff it contains a *unique minterm*. A term $\alpha \in \Gamma$ is called *redundant* with respect to Γ iff it is not *unique*. □

Lemma 1. Let a set of terms Γ be a minimal cover for σ_n and $\alpha \in \Gamma$. If (1) α cannot be merged with each $\beta \in \Gamma$, and (2) there is a minterm $t \in {}^c\breve{U}(\alpha)$, and (3) there is a minterm $\hat{t}$ and a position l such that $\alpha[l] \neq d$, $\hat{t}[l] = \overline{1 - t[l]}$ and for each position $p(\neq l)$, $\hat{t}[p] = t[p]$, then $\hat{t} \notin {}^c\Gamma$.

Proof. From (1), the relationship between α and β must be either disjoint with distance ≥ 2 or shared. From (2) and (3), we obtain, $t \notin {}^c\beta$ and $\hat{t} \notin {}^c\alpha$ for each $\beta(\neq\alpha) \in \Gamma$. There are two cases to be considered as stated above: *Case 1.* α and β are disjoint with distance ≥ 2: Then, $\hat{t} \notin {}^c\beta$ because $\hat{t}$ and t are different at exactly one position, while t and β are disjoint with distance ≥ 2. *Case 2.* α and β are shared: Then, $\hat{t}$ and $\alpha[l]$ are complementary because $\alpha[l] = t[l]$. $\beta[l] = d$ or $\alpha[l]$ because β and α are shared. Let k be a position where $t[k]$ and $\beta[k]$ are complementary. If $\beta[l] = \alpha[l]$, then $\hat{t}[l]$ and $\beta[l]$ are complementary because $\hat{t}[l]$ and $\alpha[l]$ are complementary. Then, $\hat{t} \notin {}^c\beta$. If $\beta[l] = d$, then, $k \neq l$, and $\hat{t}[k] = t[k]$. Then, $\hat{t}[k]$ and $\beta[k]$ are complementary because $t[k]$ and $\beta[k]$ are complementary. Then, $\hat{t} \notin {}^c\beta$. Hence, $\hat{t} \notin {}^c\beta$ for each $\beta \in \Gamma$. This completes the proof. □

Theorem 3. If Γ is a minimal cover for σ_n and $\sigma_n \notin \Gamma$, then $\forall\alpha\,[\,\alpha \in \Gamma \rightarrow \exists\beta\,[\,\beta \in \Gamma, \beta \neq \alpha, \alpha$ and β can be merged$\,]\,]$.

Proof. Suppose that there exists a term $\alpha \in \Gamma$ such that, for each $\beta(\neq\alpha) \in \Gamma$, α and β can be merged. Since Γ is a minimal cover for σ_n, there exists a minterm $t \in {}^c\breve{U}(\alpha)$. Let $\hat{t}$ be the minterm resulting from taking the negation of $t[l]$ where l is any arbitrary position of α such that $\alpha[l]$ is either 0 or 1. Then, it immediately follows from Lemma 1 that $\hat{t} \notin {}^c\Gamma$. Then, Γ is not a minimal cover for σ_n. A contradiction. □

We now present the simplification techniques which are used extensively for the Merge-Loop.

(1) Covered Term Reduction:

Simplifications during the Merge-Loop are mostly done by removing covered terms due to forward merges. Unit-merges and simple-merges tend to reduce greatly and cause the process to quickly approach a goal.

(2) Projection Reduction:

The following theorem justifies simplification based on projection called *projection reduction*, preserving the notion of minimal cover. It reduces the number of input variables for a particular testing, which exponentially reduces the search space.

Theorem 4. If $\Gamma = \{\alpha_1, \cdots, \alpha_m\}$ is a minimal cover for β, then $\Gamma^{\Pi}|_{\beta} = \{\alpha_1^{\Pi}|_{\beta}, \cdots, \alpha_m^{\Pi}|_{\beta}\}$ is a minimal cover for β^{Π}.

Proof. We first show that Γ^{Π} covers β^{Π}. By the definition of minimal cover, we have,

(1) $\forall\Delta\,[\,\Delta$ is a proper subset of $\Gamma \rightarrow \exists t\,[\,t \in {}^c\beta,\ t \notin {}^c\Delta\,]\,]$, and

(2) $\forall t\,[\,t \in {}^c\beta \rightarrow \exists\alpha\,[\,\alpha \in \Gamma, t \in {}^c\alpha\,]\,]$.

Using the definition of projection and complementary projection, we get

(3) $\forall t\,[\,t \in {}^c\beta \rightarrow t^{-\Pi} = \beta^{-\Pi},\ t^{\Pi} \in {}^c\beta^{\Pi}\,]$ and

(4) $\forall\alpha\,[\,\alpha \in \Gamma \rightarrow [\,\alpha^{\Pi} \in \Gamma^{\Pi}, \forall t\,[\,t \in {}^c\alpha \rightarrow t^{\Pi} \in {}^c\alpha^{\Pi}, t^{-\Pi} \in {}^c\alpha^{-\Pi}\,]\,]\,]$.

(2) and (4) implies that,

(5) $\forall t\,[\,t \in {}^c\beta \rightarrow \exists\,\alpha^{\Pi}\,[\,\alpha^{\Pi} \in \Gamma^{\Pi},\ t^{\Pi} \in {}^c\alpha^{\Pi}\,]\,]$.

From (3) and (5), we derive, $\forall t[\,t \in {}^c\beta \rightarrow t^{\Pi} \in {}^c\beta^{\Pi}, \exists\,\alpha^{\Pi}\,[\,\alpha^{\Pi} \in \Gamma^{\Pi},\ t^{\Pi} \in {}^c\alpha^{\Pi}]]$, which implies $t^{\Pi} \in {}^c\beta^{\Pi} \rightarrow t^{\Pi} \in {}^c\alpha^{\Pi}$ for some $\alpha^{\Pi} \in \Gamma^{\Pi}$, provided that $t \in {}^c\beta$. Hence,

(6) $\{\alpha_1^{\Pi}, \cdots, \alpha_m^{\Pi}\}$ covers β^{Π}.

Next, we show that no proper subset of Γ^{Π} covers β^{Π}. Let $\Delta = \{\delta_1, \cdots, \delta_p\}$ be any proper subset of Γ. Let $\Delta^{\Pi} = \{\delta_1^{\Pi}, \cdots, \delta_p^{\Pi}\}$. Then, by the universal instantiation, (1) can be rewritten as

(7) $\exists\,t_1\,[\,t_1 \in {}^c\beta,\ t_1 \notin {}^c\delta_1, \cdots,\ t_1 \notin {}^c\delta_p\,]$.

For any $\delta \in \Delta$, Δ is not disjoint with β. Then, $\delta^{-\Pi}$ is not disjoint with $\beta^{-\Pi}$, which further implies that

(8) $\beta^{-\Pi} \in {}^c\delta_1^{-\Pi}, \cdots,\ \beta^{-\Pi} \in {}^c\delta_p^{-\Pi}$.

From (3) and (8), we get,

(9) $\forall t[\,t \in {}^c\beta \rightarrow t^{\Pi} \in {}^c\beta^{\Pi},\ t^{-\Pi} \in {}^c\delta_1^{-\Pi}, \cdots,\ t^{-\Pi} \in {}^c\delta_p^{-\Pi}]$.

(7) and (9) implies that $\exists\,t_1\,[\,t_1^{\Pi} \in {}^c\beta^{\Pi}, t_1^{\Pi} \notin {}^c\delta_1^{\Pi}, \cdots, t_1^{\Pi} \notin {}^c\delta_p^{\Pi}\,]$.

That is,

(10) for any proper subset Δ^{Π} of Γ^{Π}, $\exists\,t_1\,[\,t_1^{\Pi} \in {}^c\beta^{\Pi}, t_1^{\Pi} \notin {}^c\Delta^{\Pi}\,]$.

From (6), (10), and the definition of minimal cover, we conclude,

$\Gamma^{\Pi} = \{\alpha_1^{\Pi}, \cdots, \alpha_m^{\Pi}\}$ is a minimal cover for β^{Π}. □

Example 4. Let $\Gamma = \{ddd11,\ 1dd01,\ d00d1,\ 0d101,\ d100d\}$ and $\beta = dddd1$. Then, Γ is a minimal cover for β. By applying the projection reduction, we obtain, $\beta^{\Pi} = \sigma_4$, $\Gamma^{\Pi} = \{ddd1,\ 1dd0,\ d00d,\ 0d10,\ d100\}$ and Γ^{Π} is a minimal cover for β^{Π}. □

(3) Nonmerge Reduction:

Directly from Theorem 3, we obtain the following simplification rule called *nonmerge reduction.*

Theorem 5. Let $\hat{\Gamma} = \Gamma - N$, where N is the set of all terms of Γ such that no two terms in the set can be merged. Then, $\sigma_n \in {}^c\Gamma$ iff $\sigma_n \in {}^c\hat{\Gamma}$.

Proof. ($\Rightarrow$) Directly from Theorems 2 and 3. ($\Leftarrow$) Obvious. □

(4) Nonessential Term Reduction:

We observe that some terms may never be participated as essential elements in a derivation of a goal term. Removing them will reduce the number of merges at current merge step as well as subsequent steps. The reduction is justified by the following theorem.

Theorem 6. Let Δ be a derivation of σ_n from Γ. If there is a position j and a term α such that $\alpha[j] \neq d$ and $\alpha[j] \neq 1 - \beta[j]$ for each term $\beta(\neq\alpha) \in \Gamma$, then α is nonessential to Δ.

Proof. Directly from a simple induction on $|\Delta|$. □

Nonessential term reduction and nonmerge reduction are found to be quite effective in many experimental cases.

IV. Heuristics for the Merge-Loop

Although the Merge_Loop_1 guarantees correct answer, it might contain redundant merges and unnecessary operations. To enhance the algorithm, we propose a set of *prioritized heuristics* which can significantly reduce redundant computations in most cases. We use ' σ_r', a complete term with r d's, to denote a goal term for our cover testing problem.

Heuristics from H1 to H6 are incorporated into the algorithm to speed up the Merge-Loop approaching σ_r. Heuristics are ordered by priorities in which H1 has the highest priority and H7 has the least. Each heuristic is tested in this order whether or not its condition is met at each iteration of the Merge-Loop.

H1. Perform unit merges before any other merges are tried. If a unit term is found, unit merges are performed throughout the input. Since a unit merge always reduces the number of literals of the terms involved, it is treated as a simplification called *unit-merge reduction.*

H2. Perform *simple merges.* A simple merge is generally considered as a simplification that can immediately eliminate its original terms. We name this type of simplification as *simple-merge reduction.*

H3. Terms which have participated in forward merges are more likely to trigger off other forward merges than terms involved in other types of merges. Since a forward merge always simplifies at least one of its base terms, it is considered as a simplification called *forward-merge reduction.* If two or more forward merges are found, apply H4 to choose the best one.

H4. If γ_1 and γ_2 are both forward merges such that $\{\alpha, \beta\} \mapsto \gamma_1$, $\{\alpha, \eta\} \mapsto \gamma_2$, and $I(\alpha, \beta) < I(\alpha, \eta)$, then choose γ_1 before γ_2.

Table 2. Derivation Sequence using Heuristics for Example 5.

No.	Merge	Heuristics Used
1	$\{u, y\} \mapsto \gamma^0 (= 0d1dd)$	H5
2	$\{w, \gamma^0\} \mapsto \gamma^1 (= 0dd1d)$	H3, H4
3	$\{z, \gamma^1\} \mapsto \gamma^2 (= ddd10)$	H3, H4
4	$\{v, \gamma^2\} \mapsto \gamma^3 (= dddd0)$	H2
5	$\{s, \gamma^3\} \mapsto \gamma^4 (= d11dd)$	H1
6	$\{x, \gamma^3\} \mapsto \gamma^5 (= dd0dd)$	H1
7	$\{s, \gamma^5\} \mapsto \gamma^6 (= d1dd1)$	H1
8	$\{x, \gamma^5\} \mapsto \gamma^7 (= d0ddd)$	H1
9	$\{\gamma^0, \gamma^5\} \mapsto \gamma^8 (= 0dddd)$	H1
10	$\{\gamma^4, \gamma^5\} \mapsto \gamma^9 (= d1ddd)$	H1
11	$\{\gamma^7, \gamma^9\} \mapsto \gamma^{10} (= ddddd)$	H1, H2

$s = d11d1$, $u = 01ddd$, $v = ddd00$, $w = 0d01d$,
$x = dd0d1$, $y = d01dd$, $z = 1dd10$

H5. Terms containing small number of literals are more likely to produce big merges which would allow more simplifications than terms with large number of literals.

H6. Postpone backward merges, if possible. Since backward merges increase the number of literals as the result of the operation, they are less likely to be involved in a successful derivation as essential merges. However, there are some cases in which backward merges are essential to a derivation sequence.

Heuristics presented here are incorporated into the Merge-Loop and tested successfully.

Example 5. Table 2 demonstrates some of the heuristics on the same input for Example 3. Note that the first few heuristics help the process to quickly reach a unit term which would start a series of unit merges. Because of the heuristics used, we are able to shorten the length of the derivation from 34 to 11. Nevertheless, the solution is not optimal (8 is the smallest). □

V. Algorithm Cover-Merge

We now present the final version of the Merge-Loop, the Merge_Loop_2 as an improvement over the Merge_Loop_1, which is an implementation of heuristics and simplification rules.

Algorithm Merge_Loop_2;

Input, Output: same as in the Merge_Loop_1

Procedure:

P1. Let $\Gamma^0 = \Gamma$.
$k \leftarrow 0$;

P2. **while** $\tilde{\eta} \notin \Gamma^k$ ($\tilde{\eta}$ is a generalization of η) **do**
Choose the first heuristic from H1 to H6.
Perform a merge process, using the heuristic chosen.

Perform simplifications according to covered term reduction, nonmerge reduction, and nonessential term reduction.

Let γ^k be a *nondegenerating merge* of two terms from Γ^k.

if γ^k is not found, **return** False.

Let $\Gamma^{k+1}=(\Gamma^k - \{\text{term(s) covered by } \gamma^k\}) \cup \{\gamma^k\}$.

$k \leftarrow k+1$;

end.while;

return True;

end.procedure;

Theorem 7. The Merge_Loop_2 returns *true* iff $\eta \in^c \Gamma$.

Proof. Immediately from the following: (1) The procedure terminates because (i) only nondegenerating merges can be added to the system as the result of using heuristics, and (ii) the Merge_Loop_1 terminates. (2) The Merge_Loop_1 is sound and complete (Theorem 1). (3) Simplification rules employed in the Merge_Loop_2 preserves the notion of minimal cover. □

We present the top level algorithm Cover-Merge, which calls the Algorithm Cover to test whether or not ϕ covers a term β of ϕ.

Algorithm Cover;

Input: ϕ and a term β.

Output: returns '*Yes*' if ϕ covers β. '*No*', otherwise.

Procedure:

P1. Let Γ_0 be the set of all terms of ϕ.

if any term of Γ_0 covers β, **then return** '*Yes*';

Let Γ_1 be $\{\alpha \mid \alpha \in \Gamma_0,\ \alpha$ is not disjoint with $\beta\}$.

Let $\Gamma = \Gamma_1^{\Pi}|_{\beta}$.

P2. Call the Merge-Loop to test whether $\sigma_r \in^c \Gamma$.

if $\sigma_r \in^c \Gamma$, **return** '*Yes*';

else return '*No*';

end.procedure;

Theorem 8. The Algorithm Cover returns '*Yes*' iff $\beta \in^c \phi$.

Proof. The proof of *P2* is shown in Theorem 7. It is sufficient to prove that $\beta \in^c \Gamma_0$ iff $\sigma_r \in^c \Gamma$, where r is the number of d's in β. *Part A:* Suppose $\beta \in^c \Gamma_0$. Then, $\beta \in^c \Gamma_1$. Then, $\sigma_r \in^c \Gamma$, justified by the projection reduction (Theorem 4). *Part B:* Conversely, suppose $\sigma_r \in^c \Gamma$. Let $\hat{\beta}=\beta^{-\Pi} \| \beta^{\Pi}$[2] and $\hat{\Gamma}_1=\{\alpha_1^{-\Pi} \| \alpha_1^{\Pi}, \cdots, \alpha_m^{-\Pi} \| \alpha_m^{\Pi}\}$ according to the definition of $\beta^{-\Pi}$, β^{Π}, $\alpha^{-\Pi}$ and α^{Π} from projection operation (Definition 4), where $\Gamma_1=\{\alpha_1, \cdots, \alpha_m\}$ and $\beta^{\Pi}=\sigma_r$. Recall that each term $\alpha \in \Gamma_1$ is not disjoint with β. Then, (1) for each position i, $\alpha[i]$ and $\beta[i]$ are not complementary, and (2) for each position i, $\beta^{-\Pi}[i] \neq d$, by the definition of $\beta^{-\Pi}$. From (1) and (2), there are only two cases to be considered:

- $\beta^{-\Pi}[i] = \alpha^{-\Pi}[i] = 0$ or 1.
- $\beta^{-\Pi}[i] = 0$ or 1, and $\alpha^{-\Pi}[i] = d$.

Then, it follows that $\beta^{-\Pi} \in^c \alpha^{-\Pi}$. That is, $\beta^{-\Pi} \in^c \alpha_1^{-\Pi}, \cdots, \beta^{-\Pi} \in^c \alpha_m^{-\Pi}$. Then, clearly, $\{\beta^{-\Pi} \| \alpha_1^{\Pi}, \cdots, \beta^{-\Pi} \| \alpha_m^{\Pi}\}$ covers $\beta^{-\Pi} \| \beta^{\Pi}$, which implies that $\{\alpha_1^{-\Pi} \| \alpha_1^{\Pi}, \cdots, \alpha_m^{-\Pi} \| \alpha_m^{\Pi}\}$ covers $\beta^{-\Pi} \| \beta^{\Pi}$. So, $\hat{\beta} \in^c \hat{\Gamma}_1$. Hence, $\beta \in^c \Gamma_1$. Therefore, $\beta \in^c \Gamma_0$ because Γ_0 is a superset of Γ_1. □

Algorithm Cover-Merge;

Input: ϕ and ψ.

Output: returns '*Yes*' if $\phi \equiv \psi$. '*No*', otherwise.

Procedure:

for each $\alpha \in \phi$,

Call Algorithm Cover to test whether $\alpha \in^c \psi$.

if $\alpha \notin^c \psi$, **then return** '*No*';

end.for;

for each $\beta \in \psi$,

Call Algorithm Cover to test whether $\beta \in^c \phi$.

if $\beta \notin^c \phi$, **then return** '*No*';

end.for;

return '*Yes*';

end.procedure;

Theorem 9. The Algorithm Cover-Merge returns '*Yes*' iff $\phi \equiv \psi$.

Proof. Directly from Theorem 8 and Definition 3. □

VI. Experimental Results

A program was written in C on a Sequent Symmetry, implementing the algorithms and heuristics presented in this paper. The results show that the Cover-Merge Algorithm combining with heuristics works very efficiently for most cases in comparison with BDD, especially for big inputs where BDD shows no hope of completing the verification because of memory shortage caused by its memory greedy representational scheme.

1. Results on Benchmarks

We ran the program Cover-Merge on the benchmarks in the PLASCO[3] group of IFIP. The results show that the program has successfully verified the benchmarks. Table 3 shows the comparison of running time of BDD and the program. Both programs demonstrate practical performances. However, the benchmark problems are not big enough to demonstrate the true capacity of the algorithms.

2. Results on Random Input

We use the popular model [7, 12] to define our random expressions in DNF so that each term is randomly generated according to the paradigm of the popular model. Thus, a *random input* can be generated with parameters: v (the number of variables), t (the number of terms), and p (the probability that

2) $x \| y$ denotes the concatenation of two strings x and y.

3) PLASCO consists of a group of standard benchmark problems for Boolean equivalence testing.

Table 3. Equivalence Testing of Benchmark Problems.

Programs	*Benchmarks (10 milliseconds)*							
	c	*d*	*i*	*h*	*m*	*p*	*t*	*w*
BDD	7	93	1238	6	7	335	64	1
Cover-Merge	7	31	206	4	5	52	23	1

Benchmarks: *c*:counter, *d*:d3, *h*:hostint1, *m*:mul, *p*:pitch, *t*:table, *w*:werner

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.

Our random expressions are generated with the values of the parameters determined as the following: The number of variables (v) is the main cause for the exponential growth of search space for the problem and 30 is considered to be quite big in the community. We test the algorithms on various values of v ranging from 10 to 50. Experiments show that a randomly generated term is highly likely to contain both x and $\bar{x}$ for some variable x (*contradictory*) when $p > 0.2$ and $v > 20$. And a random term may become very sparse when $p < 0.1$. So, for a practical purpose, a feasible range of p would be [0.1, 0.2] and 0.15 is picked as the value of p for the purpose. Finally, the number of terms (t) is set to 2000, sufficiently large for the tests.

A pair of random expressions can be generated in two different modes for the equivalence problem: (1) *Mode* 1: Two random expressions are generated independently. (2) *Mode* 2: An expression, ϕ_1, is generated randomly and then another expression, ϕ_2, is derived from ϕ_1 using *transformation rules*[4] which preserve the logical equivalence.

Tables 4 and 5 show the comparison of the programs run on different sizes of random expressions. Results are obtained from averaging ten runs for each v. '∞' indicates that jobs are terminated manually after about six hours of execution on the Sequent.

VII. Conclusion

We have presented a new algorithm based on a derivational method using the concept of cover and merge to solve the equivalence problem. We have proved the correctness of the algorithm and developed heuristics which have been implemented and tested successfully on a large scale of data. We have shown that it is an improvement over the state-of-the-art technology in this area in terms of performance.

Any algorithm for solving this type of problem must incorporate a wide variety of simplification techniques, and perform simplifications as much as possible. The algorithm Cover-Merge employs various types of simplification techniques as well as prioritized heuristics as its essential part. Big speed-ups has been achieved by extensively using these techniques.

The algorithm presented in this paper is widely different from other methods popular on the subject. It is not clear how the algorithm can be combined with other methods such as BDDs. However, this algorithm can be used as an alternative approach where algorithms based on representational schemes such as BDDs experience difficulties in manipulating big Boolean expressions.

Table 4. Equivalence Testing of Two Random Expressions (Mode 1): $t=2000$, $p=0.15$.

Prog.	*v (10 milliseconds)*								
	10	15	20	25	30	35	40	45	50
BDD	105	304	1981	36106	∞	∞	∞	∞	∞
Cover-Merge	3	3	3	2692	3573	5028	5143	5861	7529

Table 5. Equivalence Testing of Two Random Expressions (Mode 2): $t=2000$, $p=0.15$.

Prog.	*v (10 milliseconds)*								
	10	15	20	25	30	35	40	45	50
BDD	189	497	2517	44380	∞	∞	∞	∞	∞
Cover-Merge	3	4	395	7614	8037	8221	8613	9127	9866

References

[1] P. Ashar, S. Devadas and A. Ghosh, "Boolean satisfiability and equivalence checking using general binary decision diagrams," *The International Conference on Computer Design*, pp. 259-264, Cambridge, Mass., IEEE, New York, 1991.

[2] C. Bayol and J. Paillet, "Using TACHE for proving circuits," *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, Vol. 2, pp. 585-589, 1989.

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

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on computers*, Vol. C-35, No. 8, pp. 677-691, 1986.

[5] W. Buttner and H. Simonis, "Embedding Boolean expressions into logic programming," *Journal of Symbolic Computation*, Vol. 4, pp. 191-205, 1987.

4) Let ϕ_1 be a random expression. Define the following transformation rules: *(1) Combine:* The *combine* operation is to replace two complementary terms α, $\beta \in \phi_1$ by γ, where γ is the merge of α and β. *(2) Divide:* The *divide* operation is the reverse of *combine*. If a variable x does not occur in a term $\beta \in \phi_1$, then replace α by $\alpha \cdot x + \alpha \cdot \bar{x}$. *(3) Remove-Intersection:* This function removes the intersection of two terms $\alpha, \beta \in \phi_1$. A subexpression $\alpha + \beta$ of ϕ_1 is replaced by $\alpha + (\beta \ominus (\alpha \cap \beta))$ or $\alpha \ominus (\alpha \cap \beta) + \beta$.

[6] 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.

[7] 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.

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

[9] K. Iwama, "CNF satisfiability test by counting and polynomial average time," *SIAM J. Comput.*, Vol. 18, pp. 385-391, 1989.

[10] 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.

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

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

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

[14] J. P. Roth, *Computer Logic, Testing and Verification*, Computer Science Press, Dotomac, Md, 1980.

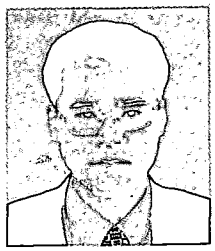
[15] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Trans. AIEE*, Vol. 57, pp. 713-723, 1938.

[16] 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.

[17] Y. Tanaka, "A dual algorithm for the satisfiability problem," *Information Processing Letters*, Vol. 37, pp. 85-89, 1991.

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

[19] I. Wegener, "On the complexity of branching programs and decision trees for clique functions," *J. ACM*, Vol. 35, No. 2, pp. 461-471, 1988.



Gyo-Sik Moon is currently a full-time lecturer at Taegu National University of Education, Taegu, Korea. Previously, he was an assistant professor at TongMyong University of Information Technology, Pusan, Korea, in 1996. He obtained a B.E. from Kyungpook National University in 1982, M.S. and Ph.D. in Computer Science from University of Oklahoma in 1989 and University of North Texas in 1995, respectively. He had worked at Systems Engineering Research Institute in Korea Advanced Institute of Science and Technology as a research scientist from 1982 to 1986. His research interests are formal reasoning and algorithm.