On the Properties of Combination Set Operations

Hiroshi G. Okuno [a,1], Shin-ichi Minato [b] and Hideki Isozaki [a]

[a] *NTT Basic Research Laboratories, Nippon Telegraph and Telephone Corporation, 3-1 Morinosato-Wakamiya, Atsugi, Kanagawa, 243-0198, Japan.*

[b] *NTT Optical Network Systems Laboratories, Nippon Telegraph and Telephone Corporation, 1-1 Hikari-no-Oka, Yokosuka, Kanagawa, 239-0847, Japan.*

Received 23 July 1997; revised 16 March 1998; accepted 8 April 1998
Communicated by T. Asano

Abstract

The problem of constructing a combination set to represent a collection of all solutions by solving constraint satisfaction problems is considered. Two kinds of combination set operations, *restriction* and *exclusion*, are invented. Simplification theorems on these two operations plays an important role to avoid combinatorial explosions. In addition, a zero-suppressed BDD, a variation of ordered binary decision diagrams, is adopted to represent a combination set and efficient implementations of the two operations are presented.

Keywords: combination set operations; restriction and exclusion operators; constraint satisfaction problem; binary decision diagram; zero-suppressed BDD

1 Introduction

A combination set can be manipulated efficiently by means of a zero-suppressed binary decision diagram (ZBDD) [5], a variant of reduced ordered BDD [1]. We apply combination sets to construct a collection of all solutions by solving a constraint satisfaction problem (CSP) [4].

The conventional way to collect all solutions of combinatorial problems is backtracking. Since backtracking requires only a record of decision points, it can enumerate solutions efficiently using a stack. There are, however, many drawbacks to this approach. One is that it is not easy to inspect the details of solutions or to compare solutions. Another is that there are many redundant

[1] Corresponding author. Email: okuno@nue.org

computations, since backtracking discards all the intermediate computations. Yet another is that when further restrictions are imposed on the solutions, the whole computation has to be repeated.

The issue in collecting all solutions is how to avoid combinatorial explosions in constructing ZBDDs. First, we model the process of solving CSPs in terms of ZBDD. Then, new combination set operations, *restriction* and *exclusion*, are introduced. Simplification theorems on these operations are presented and shown to play an important role in efficient solving of CSP by combination sets. In addition, an efficient implementation of these operations is presented and evaluated.

Section 2 describes operations of combination sets, and Section 3 presents their implementations. Section 4 shows some theorems on combination set operations. Section 5 evaluates the implementations, and Section 6 makes concluding remarks.

2 Operations on Combination Sets

A combination, α, is a set of variables, and is represented by a multiple of variables to discriminate it from a set of combinations. For example, $var(a \times b \times c) = \{a, b, c\}$. The union and difference of combinations are also referred to as *multiplication* "$\times$" and *quotient* "/", respectively. Let $var(\alpha)$ be the set of variables appearing in α. A unit combination and zero combination are represented as Λ and 0, respectively. For example, $\alpha \times \Lambda = \alpha$, $\alpha \times 0 = 0$. Note that $\alpha/\beta = 0$ iff $var(\beta) - var(\alpha) \neq \{\}$. For example, $(a \times b \times c) \ / \ (a \times d \times f) = 0$.

A *combination set* is a set of combinations. Operations on combination sets are listed below. A combination set that does not contain no combination is called *consistent*. In this paper, we assume any combination to be consistent. To illustrate the operations, we use a simple example. Suppose we have the two combination sets $P = \{a \times b, a \times b \times c, b \times c \times d\}$ and $Q = \{a \times b \times c, b \times c\}$.

(1) *Multiplication* of a variable and a combination set, "$\times$", concatenates a variable to each combination. E.g., $P \times d = \{a \times b \times d, a \times b \times c \times d, b \times c \times d\}$

(2) *Cross product* of combination sets, "$\otimes$":
$F \otimes C \equiv \{\alpha_i \times \beta_j \mid \alpha_i \in F, \beta_j \in C\}$.
E.g., $P \otimes Q = \{a \times b \times c, a \times b \times c \times d, b \times c \times d\}$

(3) *Quotient* of combination sets, " / ", and its *remainder*, " % ":
$F \ / \ C \equiv \bigcap_{\alpha \in C}(F/\alpha)$. $F \ \% \ C \equiv F - (F/C) \otimes C$.
E.g. $P \ / \ C = 0$, $P \ \% \ Q = \{a \times b, a \times b \times c, b \times c \times d\}$, $(P \otimes Q) \ \% \ Q = \{a \times b \times c\}$

(4) *Equality* and *inclusion relations* between combinations:
For example, $\alpha \succeq \beta$ iff $var(\alpha) \supseteq var(\beta)$. E.g. $a \times b \times c \times d \succeq a \times b$.

2.1 Restriction and Exclusion Operations

Let $\{\mathcal{X}_1, \cdots, \mathcal{X}_n\}$ be a set of problem variables and X_i be the domain for problem variable $\mathcal{X}_i$. For $i \neq j$, $X_i \cap X_j = \phi$ (empty). Given a set of constraints among problem variables, a constraint satisfaction problem is to find a set of instantiations of problem variables that satisfy all the constraints. Constraints may be specified as a set of *allowable combinations* or a set of *prohibited combinations*.

A combination C satisfies a set of allowable combinations S_a iff the combination C is a superset of some member of the set S_a. A combination C satisfies a set of prohibited combinations S_p iff the combination C is not a superset of any member of the set S_p. Johnson-Laird suggests using allowable combinations to represent knowledge [3].

Consider a simple constraint satisfaction problem of three problem variables: $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$. Their domains are $X_1 = \{a, b\}$, $X_2 = \{c, d, e\}$, and $X_3 = \{f, g\}$, respectively. Constraints on allowable combinations between two variables are $C_{12} = \{b \times c, b \times d, b \times e\}$, $C_{13} = \{b \times f, b \times g\}$, and $C_{23} = \{d \times f, e \times g\}$. Its solution is $\{b \times d \times f, b \times e \times g\}$.

To encode a CSP in combination sets, we introduce the following pair of operations on combination sets.

(1) *Restriction* operation, $(F \triangle C)$: $F \triangle C \equiv \{\alpha \in F \mid \exists \beta \in C,\ \alpha \succeq \beta\}$
The restriction operation calculates a subset of F such that the subset satisfies a constraint C. A constraint C specifies a set of allowable combinations to restrict combinations. E.g., $P \triangle Q = \{a \times b \times c, b \times c \times d\}$

(2) *Exclusion* operation, $(F \bigtriangledown C)$: $F \bigtriangledown C \equiv F - (F \triangle C)$
The exclusion operation removes from F those elements that satisfy a constraint C. A constraint C specifies a set of prohibited combinations. If C has a single combination set, $F \bigtriangledown C$ (if $C = \{\alpha\}$) is equivalent to $F \,\%\, \alpha$. E.g., $P \bigtriangledown Q = \{a \times b\}$

The restriction operation, $\triangle$, is equivalent to the *SupSet* operation [2]. Its opposite operation is *SubSet*. However, we call the SubSet operation the *permission* operation, as defined below:

(3) *Permission* operation, $(F \oslash C)$: $F \oslash C \equiv \{\alpha \in F \mid \exists \beta \in C \quad \alpha \preceq \beta\}$
It returns a set of combinations in F each of which is a subset of some combination in C. A constraint C specifies a set of maximal permissible combinations. E.g., $P \oslash Q = \{a \times b, a \times b \times c\}$

The simple problem given in Section 2.1 is encoded as

$$F = (X1 \otimes X2 \otimes X3) \triangle (C12 \otimes C13 \otimes C23). \tag{1}$$

Thus, the solution is obtained as

$$F = \{b \times d \times f, b \times e \times g\}. \tag{2}$$

The computation of (1) generates a lot of intermediate combinations. If the problem size grows, a combinatorial explosion, or state space explosion, may occur. That is, the computation cannot finish even though the final size of the combinations fits within the memory size of the system.

3 Some Properties for Efficiency

To avoid a combinatorial explosion, we introduce two simplification theorems on restriction and exclusion operations.

THEOREM 1 (Simplification Theorem on Restriction)

$$F \triangle \Big(\bigcup_i Ci\Big) = \bigcup_i \Big(F \triangle C_i\Big) \tag{3}$$

$$\Big(\bigcup_i X_i\Big) \triangle C = \bigcup_i \Big(X_i \triangle C\Big) \tag{4}$$

$$F \triangle \Big(\bigotimes_i C_i\Big) = \bigcap_i \Big(F \triangle C_i\Big) \tag{5}$$

$$= \Big(\Big(F \triangle C_1\Big) \cdots\Big) \triangle C_n \tag{6}$$

THEOREM 2 (Simplification Theorem on Exclusion)

$$F \bigtriangledown \Big(\bigcup_i C_i\Big) = \bigcap_i \Big(F \bigtriangledown C_i\Big) \tag{7}$$

$$= \Big(\Big(F \bigtriangledown C_1\Big) \cdots\Big) \bigtriangledown C_n \tag{8}$$

$$\Big(\bigcup_i X_i\Big) \bigtriangledown C = \bigcup_i \Big(X_i \bigtriangledown C\Big) \tag{9}$$

$$\Big(\bigotimes_i X_i\Big) \bigtriangledown C = \Big(\bigotimes_i \Big(X_i \bigtriangledown C\Big)\Big) \bigtriangledown C \tag{10}$$

$$F \bigtriangledown \Big(\bigotimes_i C_i\Big) = F \bigtriangledown \Big(\bigcup_i C_i\Big), \qquad \textit{if } \forall i\, \forall j\ i \neq j, C_i \cap C_j = \phi \tag{11}$$

Formula (1) can be converted by Theorem 1(6) into

$$F = (((X1 \otimes X2 \otimes X3) \triangle C12) \triangle C13) \triangle C23, \tag{12}$$

which is much more efficient than formula (1).

We prove a simple case of Theorem 1(3),

$$A \triangle (B \cup C) = (A \triangle B) \cup (A \triangle C). \tag{13}$$

– "Prove $A \triangle (B \cup C) \supseteq (A \triangle B) \cup (A \triangle C)$."
  Assume that $\gamma \in (A \triangle B) \cup (A \triangle C)$, then $\gamma \in (A \triangle B)$ or $\gamma \in (A \triangle C)$. If $\gamma \in (A \triangle B)$, then $\gamma \in A$ and $\exists \delta \in B$ such that $\gamma/\delta \neq 0$. Since $\delta \in B \cup C$, $\gamma \in A \triangle (B \cup C)$. The remaining case is proved similarly.

– "Prove $A \triangle (B \cup C) \subseteq (A \triangle B) \cup (A \triangle C)$."
  Assume that $\gamma \in A \triangle (B \cup C)$, then $\gamma \in A$ and $\exists \delta \in (B \cup C)$ such that $\gamma/\delta \neq 0$. That is, $\delta \in B$ or $\delta \in C$. If $\delta \in B$, then $\gamma \in (A \triangle B)$. If $\delta \in C$, then $\gamma \in (A \triangle C)$. Therefore, $\gamma \in (A \triangle B) \cup (A \triangle C)$.

The proof of the other theorems is easy.

We present some meanings of these theorems. Consider a simple divide-and-conquer approach: A problem is divided into a set of subproblems. Exclusion operations are applied to each subproblem and all the results are collected. Theorem 2(10) indicates that this approach may not compute the correct answer, and that a further application of the exclusion operation should be required.

The reason why a special care is needed because of Theorem 2(10), is that in restriction operations ($F \triangle C$) or exclusion operations ($F \bigtriangledown C$), the elements of C that are not included in any element of F are ignored. Let us define $var(F)$ for a combination set, F, as $\cup_{\alpha \in F} var(\alpha)$. Note that $\Lambda \notin var(F)$. A set, $(C \triangle var(F))$, is obtained by removing elements including none of $var(F)$ from C. Then the following theorem holds.

THEOREM 3 (Trimming Theorem on Constraint)

$$F \triangle C \equiv F \triangle (C \triangle var(F)), \text{ unless } \Lambda \in C \tag{14}$$
$$F \bigtriangledown C \equiv F \bigtriangledown (C \triangle var(F)), \text{ unless } \Lambda \in C \tag{15}$$

In other words, even if C is given as a general constraint, elements irrelevant to F are ignored. E.g., $P \triangle \Big(Q \bigcup \{a \times c \times e, a \times f\}\Big) = P \triangle Q$.

4 Implementation of Restriction Operation

The restriction operation can be defined as $F \triangle C = F \cap (F \otimes C)$. An implementation of restriction that follows this definition rigidly is inefficient, because a lot of intermediate combinations are created by the cross product and most of them become garbage immediately. There are, however, two efficient implementations of restriction: by an iterative algorithm and by a recursive one. The iterative algorithm is defined as follows:

$$F \triangle C = \bigcup_{\alpha \in C} \Big((F \,/\, \alpha) \times \alpha \Big). \tag{16}$$

The recursive algorithm is explained by introducing a new notation. A set, P, can be decomposed into two parts: $P = \Big(x_k \times (P \,/\, x_k) \Big) \bigcup \Big(P \,\%\, x_k \Big)$, where $(P \,\%\, x_k)$ is a set consisting of combinations that do not contain x_k and $(P \,/\, x_k)$ is obtained by eliminating x_k from the remaining combinations. The following algorithm computes $F \triangle C$ recursively on variables:

$$F \triangle C = \begin{cases} \Big(\{x_k\} \times \Big(((F \,/\, x_k) \triangle (C \,/\, x_k)) \cup ((F \,/\, x_k) \triangle (C \,\%\, x_k)) \Big) \Big) \\ \qquad\qquad \bigcup \Big((F \,\%\, x_k) \triangle (C \,\%\, x_k) \Big) \\ \phi \quad \text{if} \quad F = \phi \ \text{ or } \ C = \phi \\ F \quad \text{if} \quad F = C \end{cases} \tag{17}$$

Let's consider the complexity of the restriction operation, $F \triangle C$. We adopt a ZBDD to represent a combination set. Since a ZBDD is canonical for a fixed order of variables, recursive implementations of most combination set operations are quite efficient. Suppose that the numbers of nodes for F and C are represented by m and n, respectively. Combination set operations such as multiplication, $F \times C$, division, F/α ($\alpha \in C$), and union, $F \cup C$, are executed in the order of the multiple of m and n, that is, $O(mn)$ [5].

Therefore, the complexity of the iterative algorithm defined by (16) is $O(mn^2)$, because the number of nodes in F/α and $(F/\alpha) \times \alpha$ are at most m. The complexity of the recursive algorithm defined by (17) is $O(m^2 n)$, because the numbers of nodes in F/x_k and in (C/x_k) are at most m and n, respectively.

Usually a ZBDD system uses a hash table to cache the results of combination set operations. Some combination set operations may be avoided if the cache hits. Since the recursive algorithm computes similar operations more, the cache works more effectively. In fact, the recursive algorithm is faster than the iterative one [6].

Table 1
Execution times for Magic Square Problems.

order of Magic Square	Recursion		Iteration	
	Max. no. of nodes	Time (s)	Max. no. of nodes	Time (s)
3	155	0.45	622	0.88
4	83852	345	489098	702

5 Evaluations of Implementations

We use two benchmarks, N-Queens problem and Magic Square to evaluate the two implemetations.

For an N-Queens problem, a position of (i, j) on a chess board is represented by a variable $x_{i,j}$. Let X_i be a set of positions of the i-th column, that is, $X_i\{x_{i,1}, \cdots, x_{i,n}\}$, and C be a set of prohibited combinations each of which is a pair of capturing positions. N-Queens problem is encoded as $G = (\bigotimes_i X_i) \bigtriangledown C$. To avoid combinatorial explosions, ZBDDs are created incrementally by placing a queen column by column. Let G_{i-1} be a set of queens placed on each column from the first to the *i-1*-th each of which satisfies the constraints applied so far. The following recurrence relation is obtained:

$$G_1 = X1$$
$$G_i = (G_{i-1} \otimes X_i) \bigtriangledown C$$

G_i is transformed by using Theorems 1 and 2 to

$$\begin{aligned} G_i &= (G_{i-1} \otimes X_i) \bigtriangledown (C \bigtriangleup (var(G_{i-1} \otimes X_i))) \\ &= (G_{i-1} \otimes X_i) \bigtriangledown (C \bigtriangleup (var(G_{i-1}) \cup X_i)) && \text{(by } var(X_i) = X_i\text{)} \\ &= \bigcup_j ((G_{i-1} \times x_{i,j}) \bigtriangledown ((C \,/\, x_{i,j}) \cap var(G_{i-1})) && (C_{i,j} \text{ has only 2-variable} \\ &= \bigcup_j (x_{i,j} \times (G_{i-1} \bigtriangledown ((C \,/\, x_{i,j}) \cap var(G_{i-1})))) && \text{combinations)} \end{aligned}$$

In other words, first compute F_j defined as below:

$$F_j = x_{i,j} \times (G_{i-1} \bigtriangledown \{ \overbrace{x_{1,j}, \cdots, x_{i-1,j}}^{\text{the same column}}, \overbrace{x_{i-1,j-1}, x_{i-2,j-2}, \cdots}^{\text{the same diagonal}}, \overbrace{x_{i-1,j+1}, x_{i-2,j+2}, \cdots}^{\text{the same diagonal}} \})$$

and then G_i is calculated by $G_i = F_1 \cup \cdots \cup F_n$.

Two implementations of the restriction operation, by iteration (16) and by recursion (17), are evaluated. First, the speed ratio of the recursive implemen-

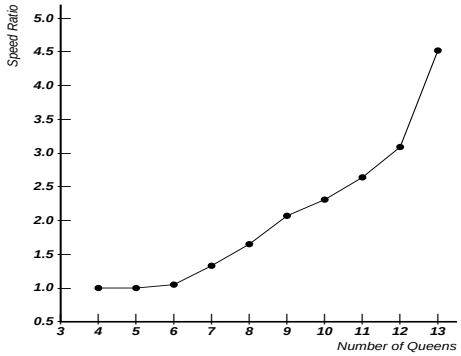

Fig. 1. Speed ratio of recursive to iterative implementations in N-Queens.

tation to the iterative one for the N-Queens problems is shown in Figure 1. For the 13-Queens problem, the recursive implementation is about 4.5 times faster than the iterative one. Space usage is also more efficient in recursive implementation. For the magic square of order 4, the recursive implementation is about 2.1 times faster than the iterative one. These benchmarks show that the hash table works well in computing subproblems in the recursive implementation.

6 Concluding remarks

In this paper, we presented two new operations on combination sets: restriction and exclusion. Theorems on these operations can be used to decompose a problem and to reduce constraints, and make it easy to pursue local consistency to avoid combinatorial explosions in solving a CSP with combination sets. We presented recursive and iterative implementations of the restriction operation and showed that the recursive one is more efficient.

Acknowledgements

We thank Dr. Hirofumi Katsuno for his comments and Dr. Ken'ichiro Ishii and Dr. Hiroshi Murase for their support.

References

[1] R. E. Bryant, Graph-based algorithm for Boolean function manipulation, *IEEE Trans. Computer*, IEEE, **C-35** (5) (1986) 677–691.

[2] O. Coudert, J. C. Madre and H. Fraisse, A New Viewpoint on Two-Level Logic Minimization, in: *Proc. of 30th ACM/ IEEE Design Automation Conference*, (ACM and IEEE, 1993) 625–630.

[3] P. N. Johnson-Laird, *Mental models* (Harvard Univ. Press, 1983).

[4] V. Kumar, Algorithms for Constraint-Satisfaction Problems: A Survey, *AI MAGAZINE*, AAAI, **13**(1) (1992) 32–44.

[5] S. Minato, Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, in: *Proc. of 30th ACM/IEEE Design Automation Conference*, (ACM and IEEE, 1993) 272–277.

[6] H. G. Okuno and S. Minato, Solving Constraint-Satisfaction Problems by Binary Decision Diagrams (*in Japanese*), *Trans. of Information Processing Society of Japan*, IPSJ, **36**(8) (1995) 1789–1799.