A Recursive Paradigm to Solve Boolean Relations

David Baneres, Jordi Cortadella, *Member*, *IEEE*, and Mike Kishinevsky, *Senior Member*, *IEEE*

Abstract—A Boolean relation can specify some types of flexibility of a combinational circuit that cannot be expressed with don't cares. Several problems in logic synthesis, such as Boolean decomposition or multilevel minimization, can be modeled with Boolean relations. However, solving Boolean relations is a computationally expensive task. This paper presents a novel recursive algorithm for solving Boolean relations. The algorithm has several features: efficiency, wide exploration of solutions, and customizable cost function. The experimental results show the applicability of the method in logic minimization problems and tangible improvements with regard to previous heuristic approaches.

Index Terms—Boolean relations, logic synthesis, Boolean minimization, decomposition.

1 Introduction

Flexibility in logic synthesis can be expressed using different abstract methods like don't care conditions (DCs), Boolean Relations (BRs), Multiple BRs (MBRs) [30], and sets of pairs of functions to be distinguished (SPFDs) [23], [29].

Don't cares form the basis for the minimization of incompletely specified functions (ISFs) and multilevel networks. BRs allow capturing more flexibility than ISFs. However, while the minimization of ISFs is a unate covering problem, solving BRs is a binate covering problem (BCP) and hence is significantly more difficult [23].

Fig. 1a illustrates an example of a BR with two input and two output variables. It is a subset of $\mathbb{B}^2 \times \mathbb{B}^2$, where $\mathbb{B} = \{0, 1\}$. The input vertex 10 is related to two different output vertices {00, 11}, and 11 is related to another pair {10, 11}. The flexibility for 10 and 11 is different. The latter can be captured by introducing a don't care into the range of output variables ($\{10, 11\} \equiv \{1-\}$).[1] The former, {00, 11}, cannot be expressed with don't cares.

To solve a BR, one needs to find a compatible multiple-output function with minimum cost. Figs. 1b and 1c depict two functions that are compatible with the original BR.

Many problems in logic design can be reduced to BRs: Boolean matching techniques for library binding [4], FSM encoding [21], Boolean decomposition [17], etc. For example, given a cut in the network, the flexibility of the nodes at the cut can be specified with a BR. E.g., if the cut contains two nodes y_1, y_2 that reconverge to an AND gate and for a given primary vector, the output of the AND gate must be 0, then the flexibility at y_1, y_2 is {00, 01, 10}.

This paper presents a novel recursive algorithm for solving BRs. The algorithm has an efficient strategy in exploring the large space of solutions and can be used in exact mode (for small relations) and in heuristic approximate mode for larger relations. The cost function can be tuned for different parameters relating to the area or delay in computing a BR. Moreover, the algorithm can further be adjusted to solve Boolean equations. The experimental results show tangible improvements with regard to previous heuristic approaches. As an application of the solver, this paper describes the use of BRs for the problem of the multiway decomposition of Boolean functions in logic circuits. The experiments show that significant delay and area improvements can be achieved by using our solver in logic circuit implementation.

The rest of the paper is organized as follows: Section 2 gives an overview of the recursive paradigm presented in this paper. Section 3 introduces the previous work in the solvers of BRs. Sections 4 and 5 present the basic definitions on the BR domain and the basis of recursive algorithm, respectively. Details of the solver are explained in Section 6. The major heuristics used to implement the recursive algorithm are presented in Section 7. Section 8 introduces how to solve a system of Boolean equations with a BR. Finally, Section 9 reports experimental results, and Section 10 introduces an application where BRs can be applied.

2 Overview

In this section, we will introduce the basis of the recursive paradigm. The formal details will be presented in Section 5. Consider the BR defined in Fig. 1a. For the sake of simplicity, the BR will be represented with the same notation of sets of

1. The don't care value $\{-\}$ denotes that the output can take all the values from the codomain $\mathbb{B}$.

- *D. Baneres is with the Universitat Oberta de Catalunya, Rambla del Poblenou 156, 08018 Barcelona, Spain. E-mail: dbaneres@uoc.edu.*
- *J. Cortadella is with the Department of Software, Universitat Politècnica de Catalunya, Building Omega, c/ Jordi Girona 1-3, 08034 Barcelona, Spain. E-mail: jordi.cortadella@upc.edu.*
- *M. Kishinevsky is with the Strategic CAD Labs, Intel Corp., MS RA2-451, 2501 NW 229th Ave., Hillsboro, OR 79124. E-mail: michael.kishinevsky@intel.com.*

Manuscript received 15 June 2007; revised 3 Mar. 2008; accepted 27 Mar. 2008; published online 8 Sept. 2008.
Recommended for acceptance by F. Lombardi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-06-0240.
Digital Object Identifier no. 10.1109/TC.2008.165.

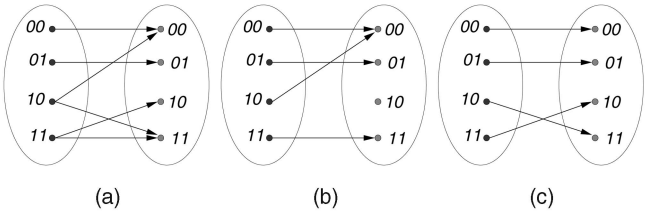


Fig. 1. (a) Example of a BR. (b) and (c) Two compatible functions.

elements. The recursive paradigm illustrated in Fig. 2 is based on the following steps:

a. Overapproximate the BR into a multiple-output ISF (MISF).[2] The input vertices such that their output vertices cannot be captured with conventional don't cares are expanded to cover more vertices of the output set. In the BR presented in Fig. 1a, the output {00, 11} of the input vertex {10} cannot be covered with don't cares. Therefore, it is expanded to $\{--\}$.
b. Use a standard MISF minimization method to obtain a multiple-output function covered by the MISF.
c. If the resulting function has no conflicts with the original relation, then report the result. Otherwise, select one element of the input set where there is an incompatibility with the original BR. In the example, the incompatibility appears in the input vertex {10}, since in the resulting function it maps to the output vertex {10} that was not in the original range ({00, 11}) for this input vertex.
d. Decompose the original BR into two smaller relations by creating a partition in the range of the output vertices of the selected incompatible input vertex.
e. Recursively solve the smaller BRs and select the best compatible solution out of the explored solutions.

Our solver reduces the BCP of solving a BR to a sequence of Boolean minimization problems applied to ISFs (MISFs), each of which is a simpler unate covering problem [12]. There are known techniques that compute the minimal solution for an MISF [1], [15], [26]. Each MISF is an overapproximation of the original BR, and therefore, its solution (a minimized form) may contain conflicts with the original BR, in which case the algorithm continues with a successive refinement of the BR into two simpler BRs that contain fewer output nodes that cannot be captured with don't care flexibility. The objective of this recursive paradigm is in decreasing the number of conflicts during each step. The algorithm may use different completion criteria trading off the quality of the solution and the runtime. For example, the algorithm can stop after one of the minimization problems for the derived MISF returns a function compatible with the BR. In this case, it is guaranteed that the correct solution of the BR is found, but there is no guarantee that it is optimal. To be exact, the algorithm can explore complete branch and bound until no better solution can be found. Multiple heuristic completion criteria may be used in between these two extremes.

2. See Section 4 for the formal definitions of BR and MISF.

3 Previous Work

Several exact and heuristic approaches have been proposed to solve BRs. The exact methods reported in [6] and [7] tackle the problem of solving a BR similarly to the Quine-McCluskey procedure [22]. The definitions of prime and prime implicant in Boolean functions are generalized to candidate prime (c-prime) and c-prime implicant in BRs. Analogous to the Quine-McCluskey procedure, first, all c-primes are generated, and then, the minimization is formulated as a BCP. The covering problem is solved by Integer Linear Programming [28], where the objective is to find the optimum sum-of-product representation of the BR. Other exact methods were presented in [20] and [21] using a Branch-and-Bound algorithm based on the BCP formulation. The major contribution of these approaches was the representation of the constraints of the BCP with Binary Decision Diagrams (BDDs) [5]. This compact representation helped to solve larger relations consuming less memory. However, the exact methods are limited to solve small and medium instances due to the complexity.

Heuristic methods provide approximated solutions with a trade-off between the quality of the solutions and the computational complexity. Herb [18] was the first heuristic method for BRs based on two-level minimization and test pattern generation techniques. The ESPRESSO [8] approach was taken as a reference in the sense that the loop reduce-expand-irredundant is repeatedly applied as long as the cost of the solution decreases. The drawback of this procedure is that the test pattern generation methodology limits the expand operation to one variable at a time. This reduction restricts the search space and increases the overall runtime. gyocro [33] was proposed as another heuristic approach also based on ESPRESSO where some of Herb's weaknesses were amended. Basically, the difference appears in the procedure *expand*, where multiple variables can be taken. The objective cost function in gyocro is slightly different compared with that of the previous approaches. The minimum sum of products with the smallest number of literals per product is searched.

Our experience demonstrates that the number of products is not necessarily a good metric for estimating the quality of the solutions. Sometimes, one needs other objectives, e.g., to balance the functions for delay optimization or to balance the support of the functions for reducing layout congestion. The recursive approach presented in this paper accepts a customizable cost function that allows guiding the search toward a user-defined goal. We also observed that heuristic methods like gyocro often cannot escape from local minima determined by the initial solution, since the reduce-expand-irredundant loop is not always capable of hill climbing. An example of this limitation is presented in Section 9.1.

4 Preliminaries

Definition 4.1. Boolean function. *A Boolean function f is a function $f : \mathbb{B}^n \rightarrow \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. A Boolean function can also be interpreted as the set of vertices $x \in \mathbb{B}^n$ such that $f(x) = 1$.*

Definition 4.2. Literals, minterms, cubes, and covers. *A* literal *is a variable or its complement. The conjunction (or product) of a set of literals is called a* cube. *A cube is called a* minterm *when the number of literals of the cube corresponds*

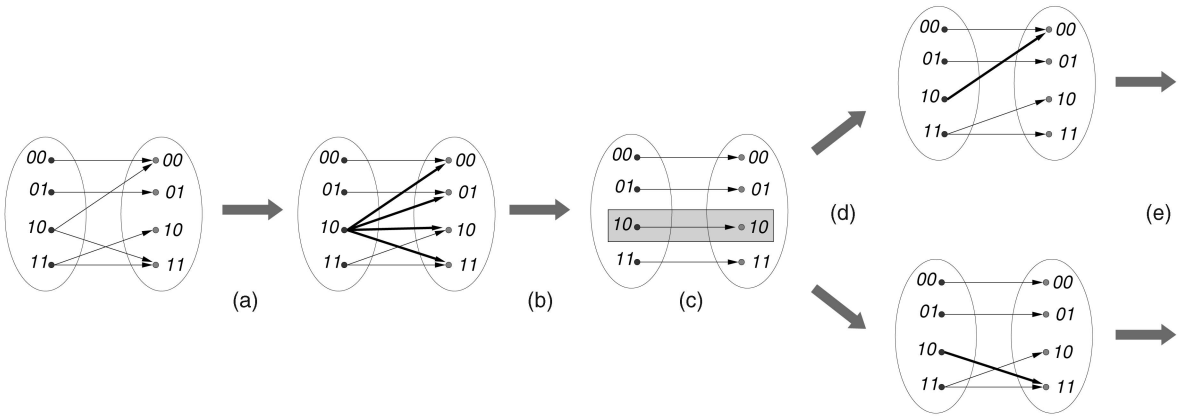


Fig. 2. Steps of the recursive paradigm implemented in BREL. (a) Overapproximate the BR to an MISF. (b) MISF minimization. (c) Selection of one input vertex where there is a conflict. (d) Decomposition in two new BRs. (e) Recursively solve the subrelations.

to the number of variables of the function. A function can be represented by a cover *that is defined as a disjunction (or sum) of cubes.*

Definition 4.3. Multiple-output Boolean function. *A multiple-output Boolean function f is a function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$. It can be also specified as a vector of Boolean functions $f = (f_1, f_2, \ldots, f_m)$.*

Hereafter, we will use $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$ to denote the set of inputs and outputs of a multiple-output Boolean function, respectively.

Definition 4.4. Incompletely specified Boolean function. *An incompletely specified Boolean function (ISF) is a function $f : \mathbb{B}^n \rightarrow \mathbb{B} \cup \{-\}$, where $-$ is called the* don't care *value of the function. An ISF can be specified by three Boolean functions, $\mathrm{OFF}(f)$, $\mathrm{ON}(f)$, and $\mathrm{DC}(f)$, that characterize the vertices in $\mathbb{B}^n$ with images 0, 1, and $-$, respectively.*

An ISF defines an interval of Boolean functions between $\mathrm{ON}(f)$ and $\mathrm{ON}(f) \cup \mathrm{DC}(f)$. An implementation of an ISF f is a Boolean function $\hat{f}$ such that

$$\mathrm{ON}(f) \subseteq \hat{f} \subseteq \mathrm{ON}(f) \cup \mathrm{DC}(f).$$

Definition 4.5. MISF. *An* MISF *is a function $f : \mathbb{B}^n \rightarrow (\mathbb{B} \cup \{-\})^m$. It can be also specified as a vector of* ISFs *$f = (f_1, f_2, \ldots, f_m)$.*

An MISF also defines an interval of multiple-output functions. The objective of a two-level minimizer is to find a function with the minimum (or minimal) sum-of-product representation that covers the MISF. Efficient methods for computing minimal sum-of-product representations are well known [1], [15], [26].

Definition 4.6. BR. *A BR R is a subset of $\mathbb{B}^n \times \mathbb{B}^m$, where $\mathbb{B}^n$ and $\mathbb{B}^m$ are called the input and output sets of R, respectively. A BR is* left-total *if for all $x \in \mathbb{B}^n$, there is $y \in \mathbb{B}^m$ such that $(x, y) \in R$. We will also refer to the left-total BRs as* well defined, *following the nomenclature in [33]. A BR is* functional *if every input vertex is associated with a single output vertex.*

Reusing the notation for the multiple-output functions, $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$ denote the set of inputs and outputs of a relation. Hereafter, we will indistinctively use the terms BR and relation.

Definition 4.7. Natural join [11]. *The natural join over the input set X between two relations R and S is defined as*

$$R(X, Y) \bowtie_X S(X, Z) = \{(x, y, z) \mid (x, y) \in R \wedge (x, z) \in S\}.$$

Note that when the relations R and S have the same input and output set and the natural join is applied over all the variables, the natural join is equivalent to the intersection operator.

The next two definitions describe how functions ISFs and MISFs can be represented with the notation of BRs.

Definition 4.8. Relationship between an incomplete function and a BR. *The* don't care *value $\{-\}$ assigned to the output of a minterm of an* ISF *denotes all the permissible values that the minterm can take from $\mathbb{B}$. Therefore, an* ISF *f_y can be also interpreted as a BR $F_y \subseteq \mathbb{B}^n \times \mathbb{B}$ such that*

- *$(x, 0) \in F_y$ iff $f_y(x) \in \{0, -\}$ and*
- *$(x, 1) \in F_y$ iff $f_y(x) \in \{1, -\}$,*

so that $f_y(x) = \{-\}$ implies that both $(x, 0)$ and $(x, 1)$ belong to the relation. This definition implies a mutual relationship between left-total BRs $F_y \subseteq \mathbb{B}^n \times \mathbb{B}$ and ISFs.

An MISF *f can be also defined as a BR $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$ such that*

$$R(X, y_1, y_2, \ldots, y_m) = \bowtie_{X \atop i \in \{1,\ldots,m\}} F_{y_i}(X, y_i).$$

Example 4.1. Consider the next two ISF functions in tabular representation:

x_1x_2	y_1
00	0
01	0
10	–
11	1

x_1x_2	y_2
00	0
01	1
10	–
11	–

They can be represented as BRs from Definition 4.8 where the don't care $\{-\}$ takes all the permissible values from $\mathbb{B}$. Similarly, the defined MISF from the conjunction of these ISFs can be also described as a BR from Definition 4.8. The resulting BRs of the ISFs and the MISF are given as follows:

ISFs		
x_1x_2	y_1	y_2
00	$\{0\}$	$\{0\}$
01	$\{0\}$	$\{1\}$
10	$\{0,1\}$	$\{0,1\}$
11	$\{1\}$	$\{0,1\}$

MISF	
x_1x_2	y_1y_2
00	$\{00\}$
01	$\{01\}$
10	$\{00,01,10,11\}$
11	$\{10,11\}$

Definition 4.9. Compatible functions. *Given a BR R, the set of multiple-output functions compatible with R is defined as*

$$\mathbb{F}(R) = \{F \mid F \subseteq R \wedge F \text{ is a multiple-output function}\}.$$

Note that $\mathbb{F}(R) = \emptyset$ if R is not well defined.

We next define a solution and an optimal solution to a BR.

Definition 4.10. Solving a BR. *Given a well-defined BR R, solving the BR R is finding a multiple-output function $F \in \mathbb{F}(R)$. We also say that F is a solution of a BR R. If in addition a cost function $c(F)$ is provided, we say that solution F is an optimal solution of a BR R if*

$$\forall F' \in \mathbb{F}(R) : c(F) \leq c(F').$$

In practical applications, it is often too costly or impossible to find an optimal solution for a given BR, and a solution of a "reasonably good cost" is searched for. The solver is said to be exact if it guarantees to find an optimal solution; otherwise, the solver is heuristic. The advantage of the algorithm described in this paper is that it can be used in both exact and heuristics modes, depending on the selected completion function.

Example 4.2. This example below shows a tabular representation of the BR that corresponds to Fig. 1a:

x_1x_2	y_1y_2
00	$\{00\}$
01	$\{01\}$
10	$\{00,11\}$
11	$\{10,11\}$

The two Boolean functions below illustrate examples of a compatible and an incompatible function for the previously defined BR:

x_1x_2	y_1y_2
00	00
01	01
10	11
11	11

Compatible function

x_1x_2	y_1y_2
00	00
01	01
10	10
11	11

Incompatible function

We will next discuss the basic principles of solving BRs (Section 5), the details of the BR solver (Section 6), and the lower level implementation aspects (Section 7).

5 Basics of Solving a Boolean Relation

5.1 Semilattice of Well-Defined Boolean Relations

In this section, the boundaries of the search space are defined. As we will show, the search space of the well-defined BRs is a semilattice. To demonstrate the existence of the semilattice, first, let us prove that there is a lattice over the set of BRs in $\mathbb{B}^n \times \mathbb{B}^m$.

Property 5.1. Lattice of BRs. *$(R, \subseteq)$ is a lattice with the top element $\mathbb{B}^n \times \mathbb{B}^m$ and the bottom element $\emptyset$. The join and meet operations are the intersection and the union of relations, respectively.*

The proof of this property follows directly from the properties of the union and the intersection on sets of finite Boolean vectors.

Lemma 5.1. *If R is a functional BR and $R' \subset R$, then R' is not well defined.*

Proof. By definition, there is only one output vertex for each input vertex in a functional BR. A relation R' such that $R' \subset R$ has at least one input vertex without image on $\mathbb{B}^m$. Thus, R' is not well defined. □

Finally, the definition of the semilattice is straightforward from the previous lemmas.

Theorem 5.1. Semilattice of well-defined BRs. *The set of well-defined BRs with the partial order $\subseteq$ is a semilattice with one greatest element $\mathbb{B}^n \times \mathbb{B}^m$ and 2^{m2^n} least elements that correspond to the elements of $\mathbb{F}(\mathbb{B}^n \times \mathbb{B}^m)$.*

Proof. Two statements have to be proved: the existence of the semilattice and the upper and lower bounds of this semilattice. First, the semilattice of well-defined BRs can be easily derived from Property 5.1. A lattice implicitly defines two semilattices, one for each operator (union and intersection). Therefore, the semilattice over the operator union exists and, therefore, over the partial order $\subseteq$. Second, let us demonstrate the greatest and the least elements:

- The supremum element is $\mathbb{B}^n \times \mathbb{B}^m$. It is clearly well defined. There is no other relation that subsumes it.
- The least lower bound elements are all the compatible multiple-output functions with n inputs and m outputs from the set $\mathbb{F}(\mathbb{B}^n \times \mathbb{B}^m)$ defined by Definition 4.9. Lemma 5.1 defines that there is no relation R' such that $R' \subset \mathbb{F}(\mathbb{B}^n \times \mathbb{B}^m)$ and R' is well defined. The number of elements in $\mathbb{F}(\mathbb{B}^n \times \mathbb{B}^m)$ is equal to 2^{m2^n}, that is, the product of m single output functions from a set of 2^{2^n} possible Boolean functions. □

5.2 Projection of a Boolean Relation to a Multiple-Output ISF

This section introduces a method to obtain an MISF from a BR. This approximation is useful to derive a fast solution for a BR.

Definition 5.1. Projection of a BR. *The projection of a relation $R(X,Y)$ onto the output y_i is another relation $(R \downarrow y_i)$ such that*

$$(R \downarrow y_i) = \{(X,z) \mid \exists y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_m \text{ such that } (X,y_1,\ldots,y_{i-1},z,y_{i+1},\ldots,y_m) \in R\}.$$

The projection of a well-defined relation R onto one output y_i implicitly defines an ISF for that output. Note that the projection can be extended to multiple outputs.

Example 5.1. From the relation presented in Example 4.2, the following projections can be derived:

x_1x_2	$R \downarrow y_1$
00	{0}
01	{0}
10	{0,1}
11	{1}

x_1x_2	$R \downarrow y_2$
00	{0}
01	{1}
10	{0,1}
11	{0,1}

Definition 5.2 MISF covering a BR. *Given a BR R, an MISF covering R can be obtained as follows:*

$$\mathrm{MISF}_R(X,Y) = \bowtie_{i\in\{1,\ldots,m\}}^{X} (R \downarrow y_i).$$

The relation $\mathrm{MISF}_R(X,Y)$ is a vector of ISFs, and hence, it is an MISF.

Example 5.2. From the projections of the relation presented in Example 4.2, the tabular representation of the original relation and the MISF_R is shown next:

x_1x_2	y_1y_2
00	{00}
01	{01}
10	{00,11}
11	{10,11}

Boolean relation R

x_1x_2	y_1y_2
00	{00}
01	{01}
10	{00,01,10,11}
11	{10,11}

MISF_R

In this example, we can observe that the image of the input vertex 10 in MISF_R covers the output vertices {01, 10} that are not included in $R(10)$. This is because MISF_R effectively expands the output set {00, 11} to the smallest covering cube $\{--\} = \{00,01,10,11\}$.

Property 5.2. *The following property holds between a well-defined BR R and MISF_R:*

$$R(X,Y) \subseteq \mathrm{MISF}_R(X,Y).$$

Proof. Let us assume that only two output variables are involved:

$$R = (R \downarrow y_1y_2) \subseteq (R \downarrow y_1) \bowtie_x (R \downarrow y_2).$$

This assumption can be proved as follows:

$$\forall (x,y_1,y_2) \in R \Longrightarrow (x,y_1) \in (R \downarrow y_1) \wedge (x,y_2) \in (R \downarrow y_2) \Longrightarrow (x,y_1,y_2) \in (R \downarrow y_1) \bowtie_x (R \downarrow y_2).$$

The previous statement can be generalized to multiple outputs:

$$(R \downarrow y_i \ldots y_j) \subseteq (R \downarrow y_i) \bowtie_x (R \downarrow y_{i+1} \ldots y_j).$$

Finally, the property $R(X,Y) \subseteq \mathrm{MISF}_R(X,Y)$ can be proved based on the previous statement and the next formula:

$$\begin{aligned} R &\subseteq (R \downarrow y_1) \bowtie_x (R \downarrow y_2 \ldots y_m) \subseteq \\ &\subseteq (R \downarrow y_1) \bowtie_x (R \downarrow y_2) \bowtie_x (R \downarrow y_3 \ldots y_m) \subseteq \ldots \subseteq \\ &\subseteq (R \downarrow y_1) \bowtie_x \ldots \bowtie_x (R \downarrow y_{m-2}) \bowtie_x (R \downarrow y_{m-1}y_m) \subseteq \\ &\subseteq \mathrm{MISF}_R. \end{aligned}$$

□

The next property is important for the presented method, since the MISF_R obtained by projection to the outputs is the smallest MISF that still covers all the compatible functions of R.

Property 5.3. *Given a BR R and the MISF_R obtained from the projection onto the outputs, there is no other MISF f' such that $R \subseteq f' \subset \mathrm{MISF}_R$.*

Proof. The proof is by contradiction. Let us assume that f' exists. This statement implies that $R \subseteq f' \subset \mathrm{MISF}_R$. Taking into account that an MISF is a vector of ISFs

$$\bowtie_{i\in\{1,\ldots,m\}}^{X} (f' \downarrow y_i) \subset \bowtie_{i\in\{1,\ldots,m\}}^{X} (\mathrm{MISF}_R \downarrow y_i).$$

The previous statement implies that there is an output $y_i \in \{y_1,\ldots,y_m\}$ that generates an ISF such that $(R \downarrow y_i) \subseteq (f' \downarrow y_i) \subset (\mathrm{MISF}_R \downarrow y_i)$. However, this ISF does not exist since $\forall y_i \in Y$, $(R \downarrow y_i) = (\mathrm{MISF}_R \downarrow y_i)$ by Definition 5.2, and hence, $(f' \downarrow y_i) \subset (R \downarrow y_i)$. □

5.3 Solution of a Multiple-Output ISF

This section describes the method to obtain a fast solution from the BR MISF_R that covers the relation R. Note that we cannot guarantee the compatibility of the solution with R.

The MISF generated from the projection onto the single outputs is solved by performing an individual minimization of the outputs with an ISF minimizer [3], [16], [27].

Example 5.3. A possible solution for the ISFs $(R \downarrow y_1)$ and $(R \downarrow y_2)$ in Example 4.2 is the following:

x_1x_2	y_1
00	0
01	0
10	1
11	1

x_1x_2	y_2
00	0
01	1
10	0
11	1

The multiple-output function for the MISF_R is shown next:

x_1x_2	y_1y_2
00	00
01	01
10	10
11	11

As shown in Figs. 3a and 3b, a BR R can be projected to MISF_R, where an MISF minimizer can be used. If the BR R is an MISF, then $R = \mathrm{MISF}_R$, and the solution is always compatible with the relation. However, the compatibility of

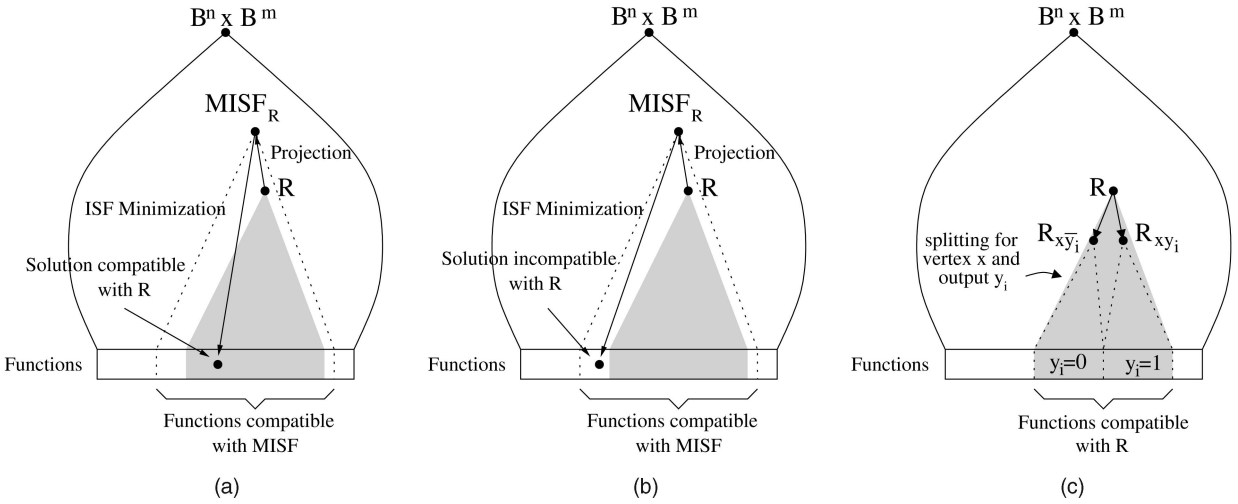


Fig. 3. Solving a BR R in the semilattice of well-defined BRs. (a) The BR R is projected to MISF_R, and MISF_R is solved with an MISF minimizer. In this case, the obtained solution is compatible with the original relation, and the process is stopped. (b) The final solution is compatible with MISF_R but incompatible with the BR. (c) The split operation is applied on the BR R. The set of all the compatible functions of R is still contained in the set of compatible functions for $R_{xy_i} \cup R_{x\overline{y_i}}$.

the solution is not guaranteed when the BR R is not an MISF since $R \subset \mathrm{MISF}_R$.

Definition 5.3. Compatibility of a function with a BR. *Given a multiple-output function F and a relation $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$, F is* compatible *with R if $F \subseteq R$. In general, we define the set of pairs of inputs and output vertices of F incompatible with R as*

$$\mathrm{Incomp}(F, R) = F \setminus R.$$

Example 5.4. The multiple-output function presented in Example 5.3 is an incompatible solution of the relation in Example 4.2. The incompatible pair is $\mathrm{Incomp}(F, R) = \{(10, 10)\}$.

5.4 Divide and Conquer

Let us next discuss the basis of the divide-and-conquer approach presented in this paper for dealing with relations in which the solution of the projected MISF_R is incompatible with the original relation.

Definition 5.4. Splitting of a BR. *Let $R \subseteq \mathbb{B}^n \times \mathbb{B}^m$ be a well-defined relation, $x \in \mathbb{B}^n$, and y_i be one of the outputs of the relation. The following two relations can be defined:*

$$R_{xy_i}(X, Y) = R - \{(x, y_1, \ldots, y_{i-1}, 0, y_{i+1}, \ldots, y_m)\},$$
$$R_{x\overline{y_i}}(X, Y) = R - \{(x, y_1, \ldots, y_{i-1}, 1, y_{i+1}, \ldots, y_m)\}.$$

We denote the previous operation by

$$\left(R_{xy_i}, R_{x\overline{y_i}}\right) = \mathrm{Split}(R, x, y_i).$$

The Split operation is graphically illustrated in Fig. 3c. Intuitively, given an input vertex x of the input set and one output y_i, the relation can be split into two relations such that one of them takes the value $y_i = 0$ and the other takes the value $y_i = 1$ for the vertex x. The two relations induce a partition over the functions compatible with R. The relations R_{xy_i} and $R_{x\overline{y_i}}$ still cover all the compatible solutions of R and no other functions.

Example 5.5. Let us take the input vertex {10} and the output y_1 from the relation in Example 4.2. Note that the selected input vertex {10} is included in $\mathrm{Incomp}(F, R)$. The objective of this selection is to remove this incompatibility in R_{xy_1} and $R_{x\overline{y_1}}$. Then, R_{xy_1} and $R_{x\overline{y_1}}$ are defined by the following tables:

R_{xy_1}	
x_1x_2	y_1y_2
00	{00}
01	{01}
10	{11}
11	{10, 11}

$R_{x\overline{y_1}}$	
x_1x_2	y_1y_2
00	{00}
01	{01}
10	{00}
11	{10, 11}

R_{xy_1} and $R_{x\overline{y_1}}$ are smaller relations. We can now recursively solve each one of them and choose the best solutions. After minimizing each one, the following multiple-output functions are obtained:

F_1	
x_1x_2	y_1y_2
00	00
01	01
10	11
11	11

F_2	
x_1x_2	y_1y_2
00	00
01	01
10	00
11	11

These functions are compatible with R_{xy_1} and $R_{x\overline{y_1}}$, respectively, and, therefore, compatible with R.

Property 5.4. *Given R, R_{xy_i}, and $R_{x\overline{y_i}}$ as defined above, the sets of compatible functions $\mathbb{F}(R_{xy_i})$ and $\mathbb{F}(R_{x\overline{y_i}})$ are a partition of $\mathbb{F}(R)$.*

Proof. A partition has the following properties:

$$\mathbb{F}(R) = \mathbb{F}\left(R_{xy_i}\right) \cup \mathbb{F}\left(R_{x\overline{y_i}}\right);\ \mathbb{F}\left(R_{xy_i}\right) \cap \mathbb{F}\left(R_{x\overline{y_i}}\right) = \emptyset.$$

```
QuickSolver (R)
{Input: A well-defined relation R(X,Y)}
{Output: A multi-output function compatible with R}
    S := R;
    for each output y_i do
        F_{y_i} := (y_i ⇔ Minimize(S ↓ y_i));
        S := S ∧ F_{y_i};
    return S;
end;
```

Fig. 4. A naive algorithm to solve a BR.

Let us prove each one independently:

- $\mathbb{F}(R) = \mathbb{F}(R_{xy_i}) \cup \mathbb{F}(R_{x\overline{y_i}})$. By the definition of the Split operation, $R = R_{xy_i} \cup R_{x\overline{y_i}}$. Therefore

$$\begin{aligned}(\mathbb{F}(R) = \{F \mid F \subseteq R\}) \wedge \left(R = R_{xy_i} \cup R_{x\overline{y_i}}\right) \Longrightarrow \\ \mathbb{F}(R) = \left\{F \mid F \subseteq R_{xy_i} \cup R_{x\overline{y_i}}\right\} \Longrightarrow \\ \mathbb{F}(R) = \left\{F \mid F \subseteq R_{xy_i}\right\} \cup \left\{F \mid F \subseteq R_{x\overline{y_i}}\right\} \\ = \mathbb{F}\left(R_{xy_i}\right) \cup \mathbb{F}\left(R_{x\overline{y_i}}\right).\end{aligned}$$

- $\mathbb{F}(R_{xy_i}) \cap \mathbb{F}(R_{x\overline{y_i}}) = \emptyset$. The BRs R_{xy_i} and $R_{x\overline{y_i}}$ differ on the output vertices of the input vertex x such that $R_{xy_i}(x) \cap R_{x\overline{y_i}}(x) = \emptyset$. Therefore, there is no Boolean function F such that $F \subseteq R_{xy_i}$ and $F \subseteq R_{x\overline{y_i}}$. □

The next theorem defines the conditions for R_{xy_i} and $R_{x\overline{y_i}}$ to be well defined and strictly smaller than R.

Theorem 5.2. *Consider an input vertex x and an output y_i of the relation R. R_{xy_i} and $R_{x\overline{y_i}}$ obtained from* $\mathrm{Split}(R, x, y_i)$ *are both well defined and strict subsets of R (i.e., $R_{xy_i} \subset R$ and $R_{x\overline{y_i}} \subset R$) iff R is well defined and $(R \downarrow y_i)(x) = \{0, 1\}$.*

Proof. When the Split operation is performed on an input vertex x such that $(R \downarrow y_i)(x) = \{0, 1\}$, it can be easily proven that R_{xy_i} and $R_{x\overline{y_i}}$ are well defined. By Definition 5.4, $\forall x' \neq x$, R_{xy_i} and $R_{x\overline{y_i}}$ have the same output vertices, and for the input vertex x, the output vertices are split in such a way that $(x, y_{1,\ldots}, y_{i-1}, 1, y_{i+1,\ldots}, y_m) \in R_{xy_i}$ and $(x, y_{1,\ldots}, y_{i-1}, 0, y_{i+1,\ldots}, y_m) \in R_{x\overline{y_i}}$. Therefore, both R_{xy_i} and $R_{x\overline{y_i}}$ are still well defined, and moreover, both are strict subsets of R since at least one of the output vertices is dropped for both subrelations. Let us assume for the contrary that $(R \downarrow y_i)(x) \neq \{0, 1\}$, e.g., $(R \downarrow y_i)(x) = \{0\}$. Then, $R_{x\overline{y_i}} = R$ is well defined but not a strict subset of R, while R_{xy_i} is not left-total and, hence, not well defined. □

Example 5.6. In Example 5.5, the input vertex {10} and the output y_1 are used in the Split operation. Note that if the vertex to split were {11}, then $R_{x\overline{y_1}}$ would not be well defined, since y_1 cannot take the value 0 for this input vertex.

6 Details of the Boolean Relation Solver

This section describes first a naive BR solver and then the recursive algorithm based on a branch-and-bound strategy. Let us start with introducing the representation of BRs with characteristic functions that is used in our implementation.

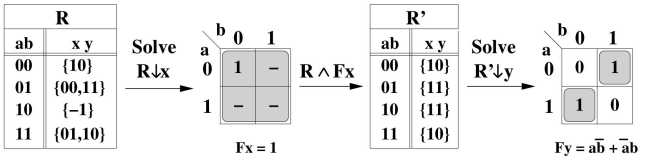


Fig. 5. Example of solving a BR with QuickSolver.

6.1 Characteristic Functions

A BR can be represented by its characteristic function.

Definition 6.1. Characteristic functions. *A BR R can be specified by a characteristic function*[3] *$\mathcal{R}$: $\mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{B}$ such that $(x, y) \in R$ iff $\mathcal{R}(x, y) = 1$.*

Characteristic functions are convenient for the automation of solving BRs since it enables reusability of algorithms and tools developed for Boolean functions.

Definition 6.2. Cofactor and existential abstraction. *The cofactors f_{x_i} and $f_{\overline{x_i}}$ of a Boolean function $f(x_1, \ldots, x_n)$ are defined as $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. The existential abstraction $\exists_{x_i} f$ is defined as $\exists_{x_i} f = f_{x_i} + f_{\overline{x_i}}$. Cofactors and existential abstraction can be extended to multiple variables.*

6.2 Quick Solver

The algorithm presented in Fig. 4 allows us to obtain a solution of the BR quickly. It was used in gyocro [33] to obtain the initial solution before applying the reduce-expand-irredundant iterations. The quick solver minimizes each output in order using the maximum flexibility provided by the relation. As long as the outputs are calculated, the constraints of the previous solutions are propagated to the rest of the outputs. The core of the algorithm is the function Minimize that performs the ISF minimization. Although this algorithm is fast, it has two drawbacks:

- The solution depends on the order in which the outputs are minimized.
- The first outputs tend to take advantage of the flexibility of the relation, whereas the last outputs inherit little flexibility. This leads to highly unbalanced and suboptimal solutions.

Example 6.1. Consider the example in Fig. 5. Initially, the flexibility of the output x is used, and the ISF for x is solved. Based on the solution F_x, the original BR is constrained, and R' is obtained such that $(R' \downarrow x) = F_x$. The ISF for the output y is extracted from the relation R' and is solved. The solution is $f(a, b, x, y) = (x \Leftrightarrow 1)(y \Leftrightarrow a\overline{b} + \overline{a}b)$. Note that the best function with the smallest number of product terms, $f(a, b, x, y) = (x \Leftrightarrow \overline{b})(y \Leftrightarrow a)$, is not found by the quick solver.

The goal of this paper is to propose a method that performs a better exploration of the space of solutions while having an affordable computational complexity.

3. We will refer the characteristic function of a relation with the symbol $\mathcal{R}$.

```
BREL (R, cost, BestF)
Input: A well-defined relation R(X,Y) and the best
       function to estimate the cost of the solution (cost)
       found compatible function (BestF)
Output: BestF returns the minimum-cost function
        compatible with R
    // Check for R to be a function
1:  if R is a function then
2:      if cost(R) < cost(BestF) then BestF := R;
3:      return;

    // R is not a function
    // Compute MISF_R and minimize
4:  MISF_R := Compute_MISF_R(R);
5:  F := Minimize(MISF_R);

    // The solution cannot be better
6:  if cost(F) ≥ cost(BestF) then return;

    // The solution is better, but it may not be compatible
7:  I := Incomp(F, R);
8:  if I = 0 then BestF := F; return;

    // Incompatible solution: split and call recursively
9:  (x, y_i) := Pick (vertex, output signal) pair from I
                such that (R ↓ y_i)(x) = {0,1};
10: (R_1, R_2) := Split(R, x, y_i);
11: BREL(R_1,cost,BestF);
12: BREL(R_2,cost,BestF);

    return;
end;
```

Fig. 6. A recursive algorithm for solving BRs.

6.3 The Recursive Approach

The approach proposed in this paper is based on the Split operation presented in Definition 5.4. The intuitive basis of this approach can be informally described as follows:

> Minimize each output independently with the maximum flexibility provided by the relation.
> If the solution is incompatible:
> Select a conflicting input vertex and an output.
> Generate two sub-relations.
> Branch-and-bound through the tree of BRs in which the leaves are the compatible functions.
> Select the best solution.

The recursive algorithm is shown in more detail in Fig. 6, where the cost of the best explored solution is used to prune the search space.

BREL is initially called with a null BR with infinite cost. Only the best solution is preserved in BestF. The algorithm checks if $\mathcal{R}$ is a function (the terminal case) in lines 1-3. In case $\mathcal{R}$ is not a function, the minimization of the $\mathrm{MISF}_{\mathcal{R}}$ is performed in lines 4 and 5 with BDD-based optimization methods (as further explained in Section 7.5). The solution, even if it is incompatible, is rejected if its cost is greater than the cost of the best previously obtained function (line 6). In case of an incompatible solution, constraining the relation further for solving the conflicts cannot improve the cost of a solution obtained for the problem with higher flexibility. If the new cost is smaller than the previous one, the compatibility of the solution is checked (line 7). If there is no conflict, the best solution is stored in the variable BestF (line 8). In case the solution is incompatible (lines 9 and 10), an input vertex and an output are selected from the incompatible points to perform the Split operation based on Theorem 5.2. The largest input cube within the characteristic function of all input conflicting vertices and an output such that $(R \downarrow y_i)(x) = \{0,1\}$ are selected to apply the Split operation (further discussed in Section 7.4). Finally, the recursive calls are done (lines 11 and 12) for each of the smaller subrelations $\mathcal{R}_1$ and $\mathcal{R}_2$.

This algorithm uses two additional parameters:

- The `cost` function can be customized by the user and is a parameter of the recursive algorithm. Previous algorithms such as the exact or heuristic solvers in [6] and [33] aim at minimizing the number of cubes of the solutions.
- The algorithm can trade-off between the quality of the solution and the runtime spent in the search. As in any branch-and-bound algorithm, the search can be stopped as soon as some resources (e.g., the CPU time) have been exhausted.

Note that incompatibilities may occur in this algorithm only at an input vertex x for which the output set cannot be precisely captured with don't cares. Consider the example in Fig. 1. BREL can potentially find an incompatibility for the input vertex 10, since its output set {00, 11} cannot be captured with don't cares, but it would not consider the input vertex 11 as a potential incompatible vertex, since its output set {10, 11} can be described as $1-$.

Example 6.2. Fig. 7 depicts how the relation $\mathcal{R}(a,b,c,x,y)$ is solved with BREL. In the first recursion, the same solution b is found for both outputs x and y. The minimization after the projection steps is represented using Karnaugh maps. After the individual minimization, a multiple-output function is composed from the individual solutions. Two conflicts are found between this function and the original BR on input vertices 010 and 101. In order to reduce the conflicts, the vertex 010 and the output y are selected to split the relation. The solver will find a compatible solution for each of the new subrelations in the second recursive iteration:

$$f(a,b,c,x,y) = \begin{cases} (x \Leftrightarrow ac)(y \Leftrightarrow b) & \text{for } y = 1, \\ (x \Leftrightarrow b)(y \Leftrightarrow a + c) & \text{for } y = 0. \end{cases}$$

For this example, the conflict for the input vertex 101 is also solved in the second recursive call. However, in general, the number of required recursive calls to solve different conflicting vertices may differ.

7 Further Implementation Details

The general branch-and-bound approach presented in Fig. 6 can be implemented in different ways. There are multiple degrees of freedom in the implementation: selecting a data structure for representing relations, a strategy to explore the branch-and-bound tree, particular cost functions, algorithms for ISF minimization, etc.

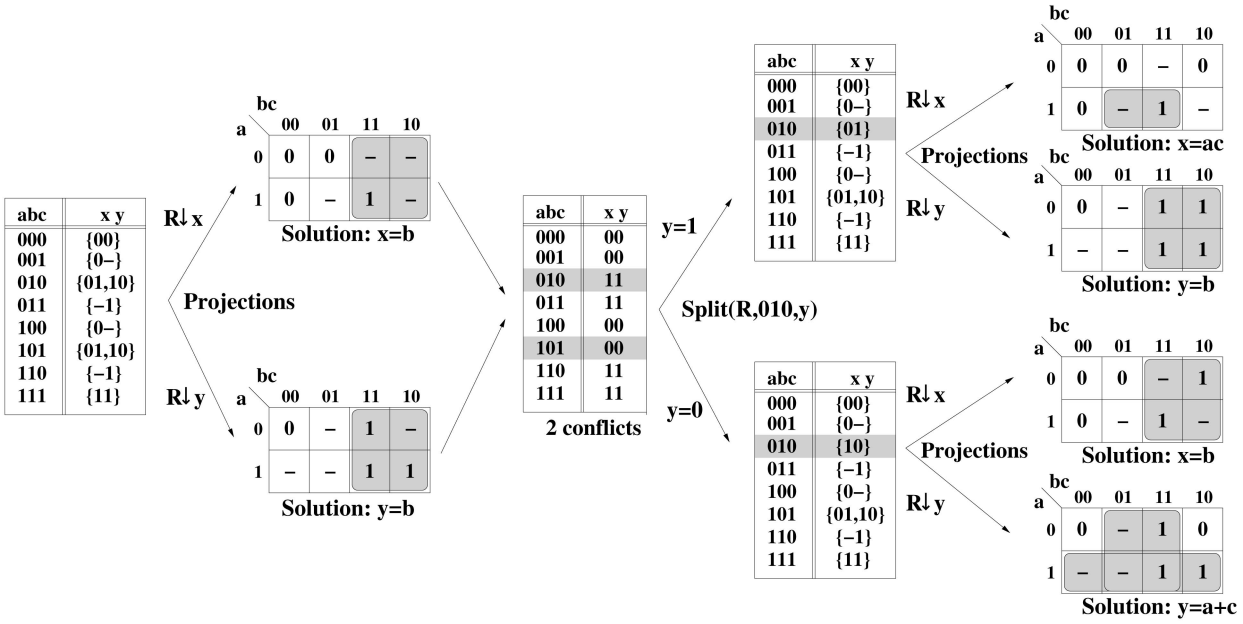


Fig. 7. Example of solving a BR.

We will next present several implementation details of our solver, BREL, that lead to an efficient trade-off between the quality of the solutions and the computational complexity of the search. Many of the implementation decisions have been taken after experimenting with different strategies and choosing the most effective ones.

7.1 Representation of Relations

BDDs [5] are used to represent and manipulate the characteristic functions of the relations. All the transformations, the evaluation of cost functions, and the ISF minimization are implemented using BDD operations.

Since all the relations generated by the solver come from a single original relation, there is a lot of sharing in the BDD data representation. The solver invokes many similar low-level BDD operations that are cached and calculated only once. This has an important impact on the performance of the solver.

7.2 Exploration of Solutions

The branch-and-bound tree of solutions is explored using a partial breadth-first search (BFS). This requires a slight modification of the algorithm in Fig. 6. All the relations generated by splitting are stored in a bounded FIFO implemented as a list.

The size of the FIFO of solutions is a parameter of the solver. Due to the bound on the number of unresolved intermediate relations, not all of the potentially generated relations are resolved into compatible functions and reach the functional leaves of the semilattice of relations. Therefore, the `QuickSolver` (Fig. 4) is used as the first step to guarantee that at least one compatible function is found during the exploration.

The BFS enables a larger diversity in the exploration of solutions and prevents the solver from spending all the resources in only one corner of the tree in searching for a local optimum.

7.3 Cost Function

A cost function is another parameter of the solver. For efficiency reasons, BDD-based cost functions are desirable since they are easy to compute. Even though the size of BDDs is not always the best estimation of complexity for a Boolean function, typically, there is a correlation between both. In the experiments, we have used different cost functions, depending on the minimization goal: the sum of BDD sizes when targeting area minimization and the sum of the squares of BDD sizes when targeting delay. The latter cost function biases the exploration toward solutions in which the complexity of the functions is balanced, and hence, the delay is more evenly distributed along all paths. The former tends to minimize the overall size regardless of the relative complexity of the subfunctions.

The experimental results, demonstrating application of the BREL to logic decomposition (presented in Section 10.2), show that these cost functions lead to significant area and delay optimization.

7.4 Split Strategy

When conflicts appear after the minimization of the $\mathrm{MISF}_{\mathcal{R}}$, an input vertex x and an output y_i must be selected for splitting (line 9 in Fig. 6). Intuitively, the solver selects the largest input cube within the characteristic function of all input conflicting vertices.

More precisely, given the characteristic function of the conflicts, Incomp, the outputs are existentially abstracted ($C = \exists_Y \mathrm{Incomp}$). Next, the shortest path in the BDD representing C is extracted. The shortest path represents the largest set of adjacent conflicting input vertices. Constraining the value of the relation in one of the vertices of this set forces many other adjacent vertices to acquire the same output value during the minimization.

The input vertex x is obtained from the incompatible input cube by assigning the value 1 to the variables with a don't care value ($\{-\}$). The selection of the output y_i must fulfill Theorem 5.2. Therefore, an output such that

TABLE 1
Normalized Comparison between Several ISF Minimization Based on BDDs

	ISOP		Constrain		LICompact	
	LIT	CPU	LIT	CPU	LIT	CPU
Eliminate non-essential	1.00	1.00	1.09	1.03	1.02	1.02
Keep non-essential	1.16	1.59	1.16	1.57	1.17	1.57

$(R \downarrow y_i)(x) = \{0, 1\}$ is selected following the variable order in the BDD manager.

7.5 Minimization of ISFs

We will next explain the details of the Minimize operation in the solver. Each ISF of the MISF_R is individually minimized with BDD-based optimization methods. A BDD-based approach contributes to speed up the solver. ISFs are defined by a pair of functions that represent the interval of flexibility [Min, Max] (or $[\mathrm{On}, \mathrm{On} \cup \mathrm{Dc}]$). There are different methods to minimize the ISF implementation using the flexibility within the specification interval. Versions of generalized cofactors, such as *constrain* and *restrict* [13], [14], have been often used to reduce the size of BDDs. A BDD operation to find irredundant SOPs is also possible by using Minato-Morreale's algorithm [24], even though the obtained solutions can be far from the optimum.

Another way to reduce the complexity is to reduce the support by eliminating nonessential variables. A variable z is called not essential if the interval $[\exists_z \mathrm{Min}, \forall_z \mathrm{Max}]$ is not empty (cf. [9, pp. 107-112]).

Our solver first reduces the support of the ISF by greedily eliminating nonessential variables from the top to the bottom of the BDD representation. After that, an irredundant SOP is calculated using Minato-Morreale's algorithm. We found this combined approach to be more efficient, in terms of the performance and quality of the solutions, than other tested techniques. Three techniques have been tested: a minimization of irredundant SOPs (ISOP) based on Minato-Morreale's approach [24], a constrain-restrict minimization (Constrain) [13], [14], and a BDD safe minimization (LICompact) [19]. Table 1 shows the normalized comparison of these ISF minimization approaches with regard to the selected ISOP minimization with the elimination of nonessential variables. The table reports the increment of the number of literals in SOP representation of the final solution (LIT) and the required CPU time (CPU) for the benchmarks used in the experimental results on Boolean Relations. The elimination of the nonessential variables contributes to significantly reduce the runtime and improves the quality of the solutions of the ISF minimization. The table also demonstrates that the irredundant SOP minimization, on the average, provides slightly better solutions than other methods as measured by the literal count in the SOP form.

7.6 The Heuristic Approach

As previously discussed, the algorithm presented in this paper can be used in both exact and heuristic modes, depending on the complexity of the problem: reasonably small BRs can be solved exactly, while for large BRs, a heuristic mode can be run, pruning the solution tree more aggressively and stopping at runtime limits.

The presented method is exact under the two following conditions:

- The ISF minimizer used by the solver is exact.
- The exploration in the semilattice of BRs is complete, i.e., not constrained by the heuristic pruning strategies.

As in any branch-and-bound approach, heuristic approaches can be used to trade-off the computational cost at the expense of sacrificing optimality. For example, it is possible to stop the search after a runtime time-out or after exploring only several subrelations and selecting the best compatible function out of those found so far. It is crucial in this process to apply the technique explained in Section 7.2 in order to guarantee that the exploration cannot stop before obtaining at least one solution. Running the `QuickSolver` guarantees that one solution is obtained for every subrelation (including the original BR). The BFS order diversifies the solutions obtained by the `QuickSolver`. The heuristic mode of our solver has been used for the experimental results reported in this paper.

7.7 Symmetries in Boolean Relations

Symmetries in Boolean functions are often used for speeding up equivalence checking between two functions by analyzing when functions (or their subfunctions) are structurally equivalent after the permutation of some variables. Symmetries can be also exploited in the characteristic functions of BRs.

Fig. 8b depicts the first and second recursions of BREL solving a two-input and two-output BR shown in Fig. 8a. Initially, BREL finds the solution $(x \Leftrightarrow 1)(y \Leftrightarrow 1)$ with three incompatible vertices $\{\overline{a}\overline{b}, \overline{a}b, a\overline{b}\}$. Let us assume that the input vertex $\overline{a}\overline{b}$ and the output x are selected to perform the split. In the second iteration of the recursion, depending on the value of the output x for the input vertex $\overline{a}\overline{b}$, the solutions are computed as follows:

$$f(a, b, x, y) = \begin{cases} (x \Leftrightarrow a)(y \Leftrightarrow 1), & \text{for } x = 1, \\ (x \Leftrightarrow 1)(y \Leftrightarrow a), & \text{for } x = 0. \end{cases}$$

Note that these solutions are fully symmetric with respect to the permutation of variables x and y. The two symmetric relations lead to solutions with equal cost, as calculated by a BDD-based cost function. Therefore, the exploration for a relation can be stopped if a symmetric relation has already been processed by the solver.

BREL has a cache of processed relations. Symmetries can be checked for every new relation from the *Split* process to identify if a symmetric relation is stored in the cache. If it is found, the exploration for this branch is stopped.

There are efficient methods for identifying the first-order [34], [35] and the second-order symmetries [10] in Boolean functions that can be applied to BRs as well. However, the analysis of BR symmetries has a high complexity, especially for large BRs. Therefore, the application of symmetry detection has to be limited in order to reduce the runtime impact. For BREL, we have made a few implementation decisions regarding the use of symmetries:

- Symmetries are only supported for output variables. This implementation decision was based on experiments in the application domains (like logic decom-

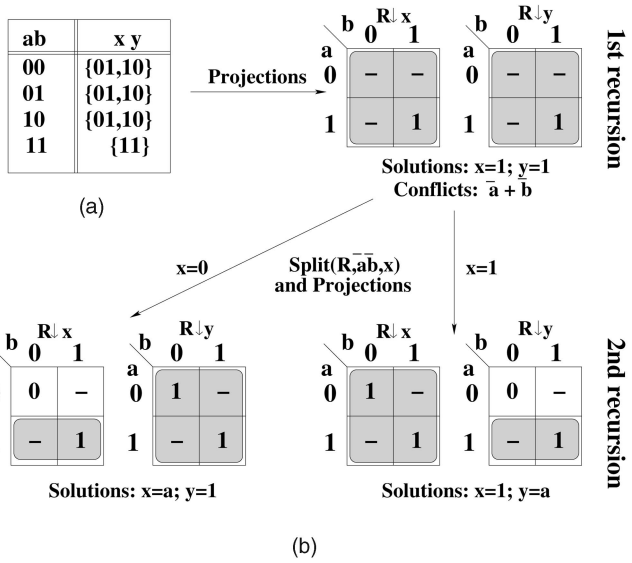


Fig. 8. Example of a symmetry in BRs.

position) where BREL is currently used. In this domain, output variable symmetries appear very often: e.g., if the large stage of logic is a symmetric gate (such as AND, OR, NAND, NOR, etc.), the permutation of two functions that feed this gate leads to a symmetric implementation of the same function.

- The solver supports all types of the first-order symmetries and the nonskew nonequivalence second-order symmetries [10].
- Symmetries are only explored during the initial recursions on the exploration tree for subrelations that are close to the original relation. Here, significant cuts on the exploration branches can be expected. Later, the symmetry check is turned off to avoid spending significant CPU time in the search for symmetries in the smaller relations.

An experiment has been done to identify the impact of the symmetry detection on the quality of results and runtime in the logic decomposition problem (the experiments on logic decomposition will be described in detail in Section 10.2). When the symmetry check is turned on, the results are, on the average, improved by 1.6 percent in delay and 1.2 percent in area after technology mapping at the cost of a runtime increase of 10.6 percent. The literal count in the SOP representation also decreases (on the average, by 1.30 percent). However, the improvement on some particular examples is significant. In the small netlist, s208, there is an improvement of 16 percent on delay and 11 percent on area with similar runtimes. In a larger netlist, s641, the delay is improved by 13 percent and the area is reduced by 17 percent. Nevertheless, the cost of the symmetry check increases the runtime by 15 percent.

The reason for the improvement of the quality of results is given as follows: For large BRs, BREL runs in a nonexact mode, since only a subset of solutions can be explored within the limited time and memory resources. Without symmetry detection, the solver can explore more subrelations within the given runtime. However, many of these subrelations are symmetric and, hence, useless. With symmetry detection, the solver spends slightly less time in solving relations (due to the penalty of symmetry detection) but actually solves more relations from different equivalence classes and hence explores more different solutions.

8 Solving Boolean Equations

Many problems in Boolean algebra with a finite number of elements can be reduced to solving a system of Boolean equations (cf. [9, pp. 153-154]). In this section, we illustrate how to solve a system of Boolean equations by solving the corresponding BR. We use characteristic functions to represent BRs. The overall strategy for solving a system of Boolean equations is given as follows:

Reduce the system of Boolean equations to a single equation of the form $\mathbb{E}(\mathtt{X},\mathtt{Y}) = 1$.
Check the consistency of the equation.
If inconsistent:
    The system has no solution.
Otherwise:
    Interpret the $\mathbb{E}(\mathtt{X},\mathtt{Y})$ as a Boolean relation with output variables corresponding to the unknowns of the equation
    Solve the corresponding BR using BREL to obtain a solution for the system of equations.

Definition 8.1. Boolean equation. *A Boolean equation is defined as*

$$P(\mathtt{X},\mathtt{Y}) \odot Q(\mathtt{X},\mathtt{Y}),$$

where P and Q are multiple-output Boolean functions of independent variables X and dependent variables Y, and $\odot$ is the equivalence $(=)$ or the inclusion-relation operator $(\leq)$.

Definition 8.2. *A* particular solution *(or, simply, a* solution*) of a Boolean equation is a multiple-output function $Y(X)$ such that $P(\mathtt{X},\mathtt{Y}(\mathtt{X})) \odot Q(\mathtt{X},\mathtt{Y}(\mathtt{X}))$ is a tautology. A Boolean equation is* consistent *if it has at least one solution. A* general solution *of a Boolean equation is a representation of the set of all its particular solutions [9]. A parametric general solution can be formed from any particular solution using the Löwenheim formula [9].*

Here, we will focus of finding particular solutions for the system of Boolean equations.

Definition 8.3. Boolean system. *A Boolean system is a set of Boolean equations:*

$$\begin{aligned} P_1(\mathtt{X},\mathtt{Y}) &\odot Q_1(\mathtt{X},\mathtt{Y}) \\ &\vdots \\ P_k(\mathtt{X},\mathtt{Y}) &\odot Q_k(\mathtt{Y},\mathtt{Y}). \end{aligned}$$

Property 8.1. *A Boolean equation $P(\mathtt{X},\mathtt{Y}) \odot Q(\mathtt{X},\mathtt{Y})$ can be transformed to the form $T(\mathtt{X},\mathtt{Y}) = 1$ using the following equivalence properties [9]:*

$$P = Q \Leftrightarrow \overline{P \oplus Q} = 1,$$
$$P \leq Q \Leftrightarrow \overline{P} + Q = 1.$$

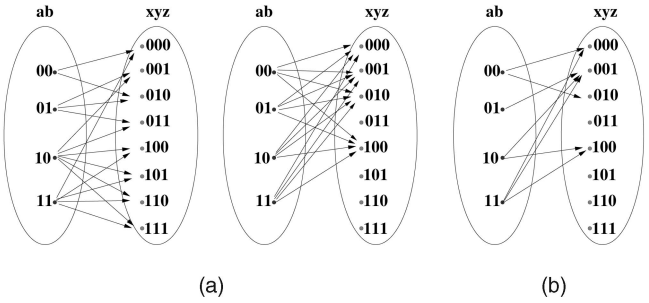


Fig. 9. Representing a Boolean system of equations as BRs.

Example 8.1. The following system of Boolean equations with a set of independent variables $\{a, b\}$ and a set of dependent variables $\{x, y, z\}$

$$x + b\overline{y}\,\overline{z} + \overline{b}z = a,$$
$$xy + xz + yz = 0,$$

can be transformed using the previous equivalence properties to

$$a\overline{b}z + ax + ab\overline{y}\,\overline{z} + \overline{a}b\overline{x}\,\overline{z} + \overline{a}b\overline{x}y + \overline{a}b\overline{x}z = 1,$$
$$\overline{x}\,\overline{y} + \overline{x}\,\overline{z} + \overline{y}\,\overline{z} = 1.$$

Note that the characteristic function of a Boolean equation matches with the definition of the characteristic function of a BR. Fig. 9a depicts the two Boolean equations of the system as sets of vertices of BRs with $\{a, b\}$ as input variables and $\{x, y, z\}$ as output variables of the BRs.

Theorem 8.1. Reduction. *A Boolean system*

$$\begin{aligned} T_1(\mathtt{X}, \mathtt{Y}) &= 1 \\ &\vdots \\ T_k(\mathtt{X}, \mathtt{Y}) &= 1 \end{aligned}$$

can be reduced to a single equation $\mathbb{E}(\mathtt{X}, \mathtt{Y}) = 1$, *where* $\mathbb{E}$ *is the characteristic function*

$$\mathbb{E}(\mathtt{X}, \mathtt{Y}) = \bigwedge_{i=1}^{k} T_i(\mathtt{X}, \mathtt{Y}).$$

The characteristic function $\mathbb{E}(\mathtt{X}, \mathtt{Y})$ only contains the feasible solutions of the system. Note that $\mathbb{E}(\mathtt{X}, \mathtt{Y})$ can be also represented as a BR.

Example 8.2. The Boolean system in Example 8.1 is reduced to the following single equation:

$$ab\overline{y}\,\overline{z} + a\overline{b}\,\overline{x}\,\overline{y}z + ax\overline{y}\,\overline{z} + \overline{a}b\overline{x}y\overline{z} + \overline{a}\,b\,\overline{x}\,\overline{y}\,z + \overline{a}\,\overline{b}\,\overline{x}\,\overline{z} = 1.$$

Fig. 9b shows the single BR associated with the characteristic function used in the above equation. It is easy to see from the figure that this BR only covers the solutions that are feasible in both BRs represented in Fig. 9a.

Property 8.2. Consistency of a Boolean system. *A Boolean system is consistent if for all* $x \in X$, *there exists a* $y \in Y$ *such that* $(x, y) \in \mathbb{E}(\mathtt{X}, \mathtt{Y})$.

As shown in [9],[4] Boolean equation $\mathbb{E}(\mathtt{X}) = 1$ is consistent iff the existential quantification of all variables (also called smoothing of all variables) gives constant 1:

$$\exists_X \mathbb{E}(\mathtt{X}) = 1.$$

Given a Boolean system, we convert it to a single Boolean equation and then check its consistency using quantification. If the system is inconsistent, then it has no solutions, and therefore, there is no corresponding well-formed BR. If the system is consistent, an equivalent well-defined BR exists, and an optimized particular solution is obtained by solving the corresponding BR using the BREL solver.

Example 8.3. The set of functions $x = ab$, $y = \overline{a}\overline{b}$, and $z = \overline{a}b + a\overline{b}$ forms a particular solution of the Boolean system in Example 8.1. This can be checked by substitution into the equation in Example 8.2 and checking that the equations simplifies to a tautology $1 = 1$. Hence, this system is consistent.

9 Efficiency of the Method

9.1 Comparison with the Expand-Reduce-Irredundant Paradigm

In this section, we illustrate the limitations of the expand-reduce-irredundant paradigm used in the Herb [18] and gyocro [33] BR solvers. Let us consider the BR depicted in Fig. 10. The best compatible function with the smallest number of product terms is $f(a, b, x, y) = (x \Leftrightarrow \overline{b})(y \Leftrightarrow a)$. Although this relation covers a small set of only eight compatible functions, the expand-reduce-irredundant local search technique is not able to explore the whole search space and find the best one.

The initial solution $f(a, b, x, y) = (x \Leftrightarrow 1)(y \Leftrightarrow a\overline{b} + \overline{a}b)$ is obtained using the procedure QuickSolver. This solution is a local minimum, and therefore, gyocro gets trapped and cannot explore the complete set of compatible functions. From the initial solution, the *reduce* procedure cannot simplify the function any further. The *expand* procedure can only be applied to the input vertex {10} to reach another MISF compatible with the original relation with the output vertices $\{-1\} = \{01, 11\}$. This expansion results in two possible solutions: the initial compatible function and the function $f(a, b, x, y) = (x \Leftrightarrow \overline{a} + b)(y \Leftrightarrow a\overline{b} + \overline{a}b)$ that has a higher cost. The expansion of the other input vertices {00, 01, 11} produces MISFs incompatible with the original relation. At this point, the exploration is stopped. Therefore, there is no feasible cube expansion that leads to the optimal solution. The reason for this limitation is that this local search exploration is not capable of exploring the range of output vertices since they cannot be covered with a set of cubes.

9.2 Experimental Results

Table 2 presents comparative results with gyocro. In [33], a similar analysis is done between gyocro and the exact minimizer in [6] and Herb [18]. The cost function used by BREL in these runs is the sum of BDD sizes for each output, aiming at area minimization. The tree of solutions has been limited to the partial exploration of 10 BRs. During this exploration, the QuickSolver procedure is applied on each

4. The consistency check presented in this paper is a modification of [9, Theorem 6.1.1], which gives conditions for a complemented form of a Boolean equation.

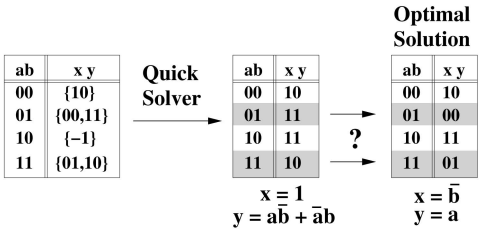


Fig. 10. Example of the expand-reduce-irredundant approach.

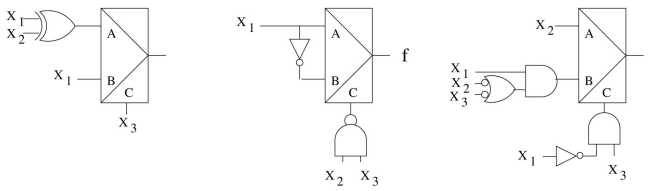


Fig. 11. Some decompositions of $f(x_1, x_2, x_3)$ using a multiplexor.

smaller BR to obtain a solution, as explained in Section 7.2. Exploring more solutions did not significantly contribute to improving the results. The table summarizes the number of input (PI) and output (PO) variables for all BR examples and reports the number of cubes (CB) and literals (LIT) in the sum-of-product representation obtained by each solver.

When comparing cubes and literals, gyocro obtains better results in several examples, since its objective cost function aims at reducing these parameters.

For more practical comparisons, we also performed two more experiments to check the quality of solutions after multilevel logic synthesis and technology mapping in SIS [31]. First, the multiple-output Boolean function in the sum-of-product representation obtained from the BR solvers is transformed to a multilevel Boolean network in SIS by applying the *algebraic* script. This script reduces the size of the network by sharing common subexpressions. BREL obtains an 11 percent improvement on the average in literal count after *algebraic* (ALG column). gyocro obtains better results only in two cases: int1 and int10.

The improvement is also observed after technology mapping. For this comparison, we used the technology mapper *map* [25], [32] and the library *lib2* of SIS for mapping a multilevel logic network into the library of logic gates. The area results obtained by BREL are better than that by gyocro in all cases except for she1 (see the column labeled with AREA). BREL obtains, on the average, a 14 percent area reduction. Although gyocro aims at minimizing the number of cubes, while BREL minimizes the number of BDD sizes, there are cases in which the solution obtained by BREL is significantly better (b9 and vtx) in the number of cubes as well. We attribute this phenomenon to the fact that gyocro could be trapped in a local minimum (e.g., after generating the initial solution), from which it cannot easily escape by simply reducing and expanding cubes. On the other hand, the BFS strategy used by BREL allows it to perform hill climbing and explore a larger set of solutions.

Finally, the runtime (columns CPU) is usually better for BREL, with a tangible speedup for two examples (b9 and vtx). The runtime of gyocro is significantly better than BREL's only for gr.

10 Application of Boolean Relations

In this section, we present an application to the problem of a multiway logic decomposition and report experimental results.

10.1 Logic Decomposition

The multiway logic decomposition problem can be formulated as follows:

Definition 10.1. *Let us assume a function $F(X)$ with the set of variables $X = \{x_1, x_2, \cdots, x_m\}$ and a gate $G(Y)$ with the set $Y = \{y_1, y_2, \cdots, y_n\}$. The decomposition of the function $F(X)$ with the gate $G(Y)$ is $F(X) = G(F_1(X), F_2(X), \ldots, F_n(X))$. The BR that specifies all possible decompositions of the function $F(X)$ with the gate $G(Y)$ is defined as follows:*

$$R(X, Y) = F(X) \Leftrightarrow G(Y).$$

TABLE 2
Comparison with gyocro [33]

			gyocro					BREL				
	PI	PO	CB	LIT	ALG	AREA	CPU	CB	LIT	ALG	AREA	CPU
int1	4	3	5	8	8	9280	0.03	7	12	9	8352	0.01
int5	4	3	7	14	11	11136	0.02	7	14	11	11136	0.00
int10	6	4	25	88	32	44544	0.08	29	102	34	41296	0.02
c17b	5	3	7	12	12	10208	0.03	7	12	12	10208	0.00
c17i	5	3	15	37	34	34336	0.04	13	32	30	32016	0.01
she1	5	3	6	20	16	16240	0.04	9	26	15	17632	0.00
she2	5	5	10	33	31	30624	0.09	12	30	24	26448	0.01
she3	6	4	9	26	23	24592	0.08	9	27	21	21344	0.01
she4	5	6	20	91	56	62176	0.14	27	120	40	46864	0.03
gr	14	11	54	455	318	346608	3.43	86	590	313	322016	6.79
b9	16	5	270	2833	321	382336	4.68	137	1174	256	306240	0.19
int15	22	14	131	1083	506	525248	21.94	166	1062	459	472352	19.14
vtx	22	6	424	4460	117	151728	30.10	244	1809	101	94656	0.58
Normalized sum			**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.77**	**0.55**	**0.89**	**0.86**	**0.44**

TABLE 3
Logic Decomposition for Mux Latches

				Delay Minimization					Area Minimization				
				ORIGINAL		Decomp. mux-latch			ORIGINAL		Decomp. mux-latch		
	PI	PO	FF	Area	Delay	Area	Delay	CPU	Area	Delay	Area	Delay	CPU
daio	2	3	4	18096	3.47	11136	3.12	0.1	18560	4.39	12064	3.68	0.1
s27	4	1	3	15312	4.47	18096	4.01	0.2	13456	4.74	15312	3.81	0.2
s208	10	1	8	88160	7.58	77952	4.79	0.8	80272	9.91	35264	10.37	0.7
s298	3	6	14	140128	6.72	91408	4.18	1.4	125744	8.37	62176	4.72	1.4
s349	9	11	15	233856	9.45	333616	9.54	5.2	157760	10.81	199984	13.33	5.0
s382	3	6	21	223648	9.63	192560	6.68	4.0	154512	8.87	127136	6.94	3.9
s386	7	7	6	182352	7.84	155904	5.87	2.4	144304	7.96	115072	6.97	2.3
s420	18	1	16	207408	9.67	108112	6.74	41.3	160544	18.08	61248	19.77	41.3
s444	3	6	21	213904	8.54	182816	6.07	4.0	181424	10.23	75632	8.57	3.9
s510	19	7	6	300208	8.42	316448	7.91	297.0	287216	9.40	245456	9.17	303.6
s526	3	6	21	225504	8.75	166112	5.60	3.7	188848	9.02	97904	8.90	3.5
s641	35	23	19	522464	12.16	376304	8.18	9.5	191632	22.48	214480	10.69	9.2
s832	18	19	5	338256	10.28	327584	7.81	27.3	337328	11.15	270976	8.85	27.4
s953	17	24	29	503904	9.82	528032	8.31	32.3	423632	12.89	380944	9.66	27.3
s1196	14	14	18	1062560	11.91	1220784	12.62	5.7	558192	19.20	642384	13.77	5.6
s1488	8	19	6	741472	9.98	802720	9.73	7.6	723840	12.25	660272	11.85	7.4
s1494	8	19	6	729872	10.14	759104	10.00	7.7	688112	12.46	611088	13.03	7.4
sbc	40	56	28	920112	8.88	979504	8.73	21.0	775344	14.50	756320	11.18	21.2
Normalized sum				**1.00**	**1.00**	**0.98**	**0.82**		**1.00**	**1.00**	**0.87**	**0.85**	

We next present an example to clarify the decomposition problem. Consider the following Boolean function:

$$f(x_1, x_2, x_3) = x_1(\overline{x_2} + \overline{x_3}) + \overline{x_1}x_2x_3.$$

The goal is to decompose this function using a multiplexor and, therefore, to absorb part of the original function f within the multiplexor with the function $Q(A, B, C) = A \cdot C + B \cdot \overline{C}$. A BR will enclose all possible decompositions than can be performed using the multiplexor. The next tabular representations show the original function $f(x_1, x_2, x_3)$ and the corresponding BR for the multiplexor:

$x_1x_2x_3$	f
000	0
001	0
010	0
011	1
100	1
101	1
110	1
111	0

$x_1x_2x_3$	ABC
000	$\{-00, 0-1\}$
001	$\{-00, 0-1\}$
010	$\{-00, 0-1\}$
011	$\{1-1, -10\}$
100	$\{1-1, -10\}$
101	$\{1-1, -10\}$
110	$\{1-1, -10\}$
111	$\{-00, 0-1\}$

The construction of the BR can be done intuitively. The relation is built by finding all the possible values of the inputs of the multiplexor that yield the desired output value in the function. For instance, the multiplexor produces the output value $Q(A, B, C) = 0$ regardless of whether the value of (A, B, C) is -00 or $0-1$. Therefore, the output of the relation for the minterms where $f(x_1, x_2, x_3) = 0$ is $\{-00, 0-1\}$. The same reasoning can be followed to find the output set of the remaining minterms of the relation. Note that the solution of one output of the BR is conditioned to the values of the other outputs. For instance, the output A can only achieve the value 1 for the minterm $x_1x_2x_3$ if BC obtains the value 00.

Many decompositions can be found using the BR. Fig. 11 depicts some of these solutions. A solver of BRs will explore the set of solutions and will return one of them based on the minimization objective.

10.2 Experimental Results

Table 3 reports the results of an experiment designed to illustrate the applicability of BREL and the customization of its cost function. We consider the existence of a flip-flop with an embedded mux in the library, and the next-state equation $Q^+ = A \cdot C + B \cdot \overline{C}$. This three-input flip-flop (typically available in the industrial gate libraries) enables the implementation of the next-state function $F(X)$ as the composition of three functions: $A(X)$, $B(X)$, and $C(X)$. The BR specifying this flexibility is $F(X) \Leftrightarrow (A \cdot C + B \cdot \overline{C})$, where A, B, and C are the output variables. The table summarizes the number of primary inputs (PI), the number of primary outputs (PO), and the number of flip-flops of the network (FF). The last row of the table summarizes the results and shows the global improvement obtained by the mux-based decomposition with BRs. The table reports the results for two different cost functions. First, the cost function has been defined as the sum of the squares of the BDD sizes for the three functions. The squaring favors a tendency to balance the complexity of the function and, therefore, reduce the delay of the circuit. In the second part of the table, the sum of BDD sizes has been used, aiming at minimizing the total area. In this table, BREL is limited to explore up to 200 BRs for each next-state function.

The table reports the area and delay of the combinational part of the circuit for each cost function. Only the area and the delay of the combinational logic are considered. We make an optimistic assumption considering that the mux is embedded in the flip-flop without any extra area and delay overhead. In the delay optimization, the results have been obtained by collapsing the next-state functions, running the algebraic script, speed_up,[5] and technology mapping in SIS.

5. This command in SIS produces more balanced solutions and contributes to improve the global delay of the network.

For the mux latch, the decomposition is done before running the algebraic script. In general, the results manifest several features of the approach:

1. The delay is usually reduced (sometimes significantly, e.g., s382, s641, and s832).
2. In many cases, the area is also reduced due to the power of Boolean decomposition (e.g., s420, s526, and s641).
3. In some cases, the delay is reduced at the expense of increasing the area due to the balancing tendency of the cost function (e.g., s953 and sbc).
4. The CPU time is affordable.

In two cases (s349 and s1196), both the area and the delay became worse with the mux-based decomposition.

For area optimization, the process of minimization is the same as the previous one without the speed_up command. The behavior of the results is similar:

1. The area is also reduced considerably (e.g., s298, s420, s444).
2. In many cases, the delay is reduced as well.
3. Only in a few cases did the delay increase (e.g., s208, s420, and s1494).
4. The cases in item 3 are sometimes related to circuits where the area is also worse (e.g., s349).
5. The CPU time is similar to the delay decomposition.

There are four cases (s27, s349, s641, and 1196) where the results were worse. Some of these circuits (s349 and 1196) are worse in both area and delay minimization. The heuristic methods applied in BREL and the limitation on the number of explored BRs sometimes lose some of the good solutions.

11 Conclusions

This paper describes a new algorithm for solving BRs and Boolean equations. Experimental results demonstrate that this approach is capable of finding better solutions in shorter runtimes than the previously known techniques. The reason for this advantage is that our exploration technique is more immune to the danger of being trapped in local minima and better explores the solution space. Depending on the complexity of the original BR, our solver can work in the exact or in the approximate mode. In this paper, we also demonstrated a successful application of our solver to the problem of decomposing Boolean functions.

Acknowledgments

This work was supported in part by CICYT TIN2007-66523 and by a grant from Intel Corp. This paper is an extended version of [2].

References

[1] P. Agrawal, V.D. Agrawal, and N.N. Biswas, "Multiple Output Minimization," *Proc. 22nd ACM/IEEE Design Automation Conf. (DAC '85),* pp. 674-680, 1985.

[2] D. Baneres, J. Cortadella, and M. Kishinevsky, "A Recursive Paradigm to Solve Boolean Relations," *Proc. 41st ACM/IEEE Design Automation Conf. (DAC '04),* pp. 416-421, June 2004.

[3] T. Bartee, "Computer Design of Multiple-Output Logical Networks," *IRE Trans. Electronic Computers,* vol. 10, pp. 21-30, 1961.

[4] L. Benini and G.D. Micheli, "A Survey of Boolean Matching Techniques for Library Binding," *ACM Trans. Design Automation of Electronic Systems,* vol. 2, no. 3, pp. 193-226, July 1997.

[5] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient Implementation of a BDD Package," *Proc. 27th ACM/IEEE Design Automation Conf. (DAC '90),* pp. 40-45, 1990.

[6] R. Brayton and F. Somenzi, "An Exact Minimizer for Boolean Relations," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '89),* pp. 316-319, Nov. 1989.

[7] R. Brayton and F. Somenzi, "Minimization of Boolean Relations," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '89),* pp. 738-743, 1989.

[8] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[9] F. Brown, *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[10] M. Chrzanowska-Jeske, "Generalized Symmetric Variables," *Proc. Eighth IEEE Int'l Conf. Electronics, Circuits and Systems (ICECS '01),* pp. 1147-1150, 2001.

[11] E.F. Codd, "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia 6, "Data Base Systems".* Prentice Hall, May 1971.

[12] O. Coudert, "Two-Level Logic Minimization: An Overview," *Integration, the VLSI J.,* vol. 17, no. 2, pp. 97-140, 1994.

[13] O. Coudert, C. Berthet, and J. Madre, "Verification of Synchronous Sequential Machines Using Boolean Functional Vectors," *Proc. IFIP Int'l Workshop Applied Formal Methods for Correct VLSI Design,* pp. 111-128, Nov. 1989.

[14] O. Coudert and J. Madre, "A Unified Framework for the Formal Verification of Circuits," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '90),* pp. 126-129, Nov. 1990.

[15] O. Coudert and J.C. Madre, "New Ideas for Solving Covering Problems," *Proc. 32nd ACM/IEEE Design Automation Conf. (DAC '95),* pp. 641-646, 1995.

[16] R. Cutler and S. Muroga, "Useless Prime Implicants of Incompletely Specified Multiple-Output Switching Functions," *Int'l J. Parallel Programming,* vol. 9, no. 4, 1980.

[17] M. Damiani, J. Yang, and G.D. Micheli, "Optimization of Combinational Logic Circuits Based on Compatible Gates," *IEEE Trans. Computer-Aided Design,* vol. 14, no. 11, pp. 1316-1327, Nov. 1995.

[18] A. Ghosh, S. Devadas, and A. Newton, "Heuristic Minimization of Boolean Relations Using Testing Techniques," *Proc. IEEE Int'l Conf. Computer Design (ICCD '90),* Sept. 1990.

[19] Y. Hong, P.A. Beerel, J.R. Burch, and K.L. McMillan, "Safe BDD Minimization Using Don't Cares," *Proc. 34th ACM/IEEE Design Automation Conf. (DAC '97),* pp. 208-213, 1997.

[20] S. Jeong and F. Somenzi, "A New Algorithm for the Binate Covering Problem and Its Application to the Minimization of Boolean Relations," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '92),* pp. 417-420, Nov. 1992.

[21] B. Lin and F. Somenzi, "Minimization of Symbolic Relations," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '90),* pp. 88-91, Nov. 1990.

[22] E. McCluskey, "Minimization of Boolean Functions," *Bell System Technical J.,* vol. 35, pp. 1417-1444, 1956.

[23] G.D. Micheli, *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.

[24] S. Minato, "Fast Generation of Prime-Irredundant Covers from Binary Decision Diagrams," *IEICE Trans. Fundamentals of Electronics, Comm. and Computer Sciences,* vol. E76-A, no. 6, pp. 967-973, June 1993.

[25] R. Rudell, "Logic Synthesis for VLSI Design," PhD dissertation, Univ. of California, Berkeley, Apr. 1989.

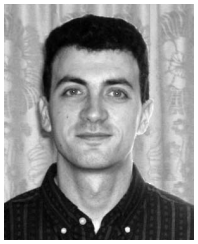
[26] R.L. Rudell and A. Sangiovanni-Vincentelli, "Multi-Valued Minimisation for PLA Optimisation," *IEEE Trans. Computer-Aided Design,* pp. 727-750, 1987.

[27] T. Sasao, "An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays," *Proc. Eighth IEEE Int'l Symp. Multiple-Valued Logic (ISMVL '78),* pp. 65-72, 1978.

[28] A. Schrijver, *Efficient Parallel Algorithms.* John Wiley & Sons, 1998.

[29] E. Sentovich and D. Brand, "Flexibility in Logic," *Logic Synthesis and Verification,* S. Hassoun and T. Sasao, eds., chapter 3, pp. 65-88, Kluwer Academic Publishers, 2002.

[30] E. Sentovich, V. Singhal, and R. Brayton, "Multiple Boolean Relations," *Proc. Int'l Workshop Logic Synthesis (IWLS '93),* May 1993.
[31] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," technical report, Univ. of California, Berkeley, May 1992.
[32] H.J. Touati, *Performance-Oriented Technology Mapping,* Memorandum number ucb/erl m90/109, Electronics Research Laboratory, College of Eng., Univ. of California, Berkeley, Nov. 1990.
[33] Y. Watanabe and R. Brayton, "Heuristic Minimization of Multiple-Valued Relations," *IEEE Trans. Computer-Aided Design,* vol. 12, no. 10, pp. 1458-1472, Oct. 1993.
[34] J.S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J.R. Burch, "Generalized Symmetries in Boolean Functions: Fast Computation and Application to Boolean Matching," *Proc. Int'l Workshop Logic Synthesis (IWLS '04),* pp. 424-430, 2004.
[35] J.S. Zhang, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske, "Symmetry Detection for Large Boolean Functions Using Circuit Representation, Simulation, and Satisfiability," *Proc. 43rd ACM/IEEE Design Automation Conf. (DAC '06),* pp. 510-515, 2006.



David Baneres received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, in 2002 and 2008, respectively. He was an intern student at Strategic CAD Labs, Intel Corp., in 2004. He is currently a lecturer in the Universitat Oberta de Catalunya, Barcelona. His research interests include computer-aided design of very large-scale integration systems, with special emphasis on logic synthesis.

Jordi Cortadella received the MS and PhD degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, in 1985 and 1987, respectively. He is a professor in the Department of Software, Universitat Politècnica de Catalunya. In 1988, he was a visiting scholar at the University of California, Berkeley. His research interests include formal methods and computer-aided design of VLSI systems, with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis. He has coauthored numerous research papers and has been invited to present tutorials at various conferences. He has served on the technical committees of several international conferences in the field of design automation and concurrent systems. He received the best paper awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems and at the Design Automation Conference, both in 2004. In 2003, he was the recipient of a Distinction for the Promotion of the University Research by the Generalitat de Catalunya. He is a member of the IEEE.

Mike Kishinevsky received the PhD degree from the Electrotechnical University of St. Petersburg, Russia. He is a principal engineer at Strategic CAD Labs, Intel Corp., Hillsboro, Oregon. Prior to joining Intel in 1998, he was a research fellow at the Russian Academy of Science, a senior researcher at R&D Coop TRASSA, a visiting associate professor at the Technical University of Denmark, and a professor at the University of Aizu, Japan. He coauthored three books in asynchronous design and has published more than 70 journal and conference papers. He has served on the technical program committee of several conferences and workshops. He is a senior member of the IEEE.