Integer Pair Representation for Multiple-Output Logic

Rafael A. Arce Nazario
Advisor: Manuel Jimenez
Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
rafael.arce@ece.uprm.edu

Abstract

Extensions allowing the Integer Pair Representation (IPR) format to work with multiple-output binary valued expressions are presented. The structure and semantics of the new format, IPR-M, are discussed as well as algorithms to aid in using this format for the representation and minimization of Boolean functions.

1 Introduction

The representation and manipulation of two-level combinational logic is essential to the design, synthesis and implementation of digital systems. Several formats supporting two-level combinational logic representations and corresponding manipulation algorithms have been proposed. Among these, binary decision diagrams (BDDs) and positional cube notation (PCN) are worth mentioning. As developed as these two models are, their implementations still provide room for improvement in the way they use memory for representation of literals. For example, BDDs use a graph structure to represent a function, where each node is a literal and the vertices represent it's relation to the other literals [3]. PCN uses two characters per binary-valued variable [2]. Memory usage in these methods is dictated by their structure and semantics, and therefore, a mechanism to optimize the memory usage of BDD or PCN would probably render useless the representations and operators.

A more compact representation of Boolean functions, the Integer Pair Representation (IPR), was suggested by Diaz et al. [1]. In this representation, each product term is identified by an ordered pair (c_x, c_y) of positive integers including zero, where the bits that compose each integer are determined by the state of the variable in each product term. Because of the structure of this notation, each literal can be implemented using only two bits. A discussion of the conversion rules, operators and algorithms for an Espresso-like minimization heuristic algorithm is provided in [1].

An important aspect missing from IPR Algebra is its ability to represent (and process) multiple-output Boolean expressions. In the case of logic minimization, it is generally known that when several functions are to be implemented, it is often possible to obtain a multiple-output minimized expression requiring fewer gates than separate circuits implementing the individual optimized functions [4]. So, to be considered a full-fledged 2-value logic representation notation, the IPR Algebra should be expanded to represent and manipulate multiple-output expressions.

We shall call our adaptation of IPR for multiple-output capability the IPR-M. In the rest of this paper, structure, semantics and algorithms for the algebra of IPR-M are discussed. The implementation of a minimization program using this new scheme is presented and its results are compared with Espresso.

2 Definitions

Before we proceed to explain the structure, operators, and algorithms proposed for IPR-M, some terms used throughout the paper are defined. Most of these definitions were adopted from [1], [5], [6] and [7].

An incompletely specified, binary-valued multiple-output Boolean function f is a mapping

$$f : \{0,1\}^n \rightarrow \{0,1,*\}^m \quad (1)$$

$$Y = (y_1, \ldots, y_m) = (f_1(X), \ldots, f_m(X)), \quad (2)$$

where n is the number of inputs, m is the number of outputs, $X = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ is the input vector, and Y is the output vector of f . The **onset** F, **offset** R, and **don't care set** D are sets of minterms which are mapped by Y to 0,1, and *, respectively.

A **literal** is a binary Boolean variable or its complement. A **cube** is a product of one or more literals such that no two literals are complement of each other. A variable missing in a cube is called a **don't care variable**

A cube A is said to **cover** another cube B if the set of minterms represented by B is a subset of that represented by A. A cube A is an **implicant** of f if $A \subseteq F \cup D$. If there is no other cube B such that A is covered by B then A is a **prime implicant**. A prime cube is **essential** if it has at least a single minterm covered by this and only this prime cube. A prime cube is **redundant** if every minterm it covers is also covered by essential prime cubes. A prime cube which is neither essential nor redundant is a **selective prime cube**.

The **distance** between two cubes A and B is the number of variables that appear in both A and B and are uncomplemented in A and complemented in B or viceversa.

3 Structure

3.1 Single Output IPR

In the IPR notation proposed in [1], each cube C of an n-variable Boolean expression is mapped to an ordered pair (c_x, c_y), where c_x and c_y are n-bit binary-valued vectors, and are referred to as the position and expansion parts of the IPR term, respectively. Each of the binary-valued vectors c_x and c_y are constructed as follows:

$$c_x = c_{x_1}, c_{x_2}, c_{x_3}, \ldots, c_{xn} \quad (3)$$

$$c_y = c_{y_1}, c_{y_2}, c_{y_3}, \ldots, c_{yn} \quad (4)$$

where each c_{xi}, c_{yi} are determined according to Table 1.

Table 1. IPR values for the position (c_{xi}) and expansion (c_{yi}) terms.

$Condition$	c_{xi}	c_{yi}	Meaning that in C
$\bar{x}_1 \in C$	0	0	x_1 appears inverted
$x_1 \in C$	1	0	x_1 appears non inverted
$x_1 \notin C$	0	1	x_1 is a don't care variable
$--$	1	1	Cube is not valid

3.2 IPR-M

To account for the representation of multiple-output functions, we propose the addition of a third term for each cube (we shall call this term the *function term*). Thus, for an m-output expression, each cube would now be represented by an ordered triplet (c_x, $c_{y,}c_z$), where c_x, $c_y$will be assigned according to the IPR rules, and c_z is an n-tuple where each bit c_{zi} indicates if the cube belongs (c_{zi}=1) or does not belong (c_{zi}=0) to function f_i. A cube with every $f_i = 0$ is not a valid cube.

To maintain compatibility with the original IPR scheme, single-output expressions can be expressed IPR-M by converting each ordered pair to an ordered triplet with the same position, and expansion terms, and a single '1' in the function term.

4 Cube operators

The implementation in IPR of two common cube operators: coverage and orthogonality was described in [1]. We proceed to expand the original implementation to handle multi-output expressions.

4.1 Coverage

Let A= (a_x, a_y, a_z) and B= (b_x, b_y, b_z) be a pair of cubes and their representation in IPR-M. Cube A covers cube B, written $B \subseteq A$, if every literal in A also appears in B and A appears in all the functions where B appears.

In IPR-M, coverage is guaranteed if and only if the following is satisfied:

$$b_x \wedge \bar{a}_y = a_x \quad (5)$$

$$and \quad b_y \vee a_y = a_y \quad (6)$$

$$and \quad b_z \wedge a_z = b_z \quad (7)$$

4.2 Orthogonality

Two cubes A and B are said to be orthogonal (written $A \perp B$) if their intersection is an empty set (they do not have any minterms in common). In IPR-M, two multi-output cubes A and B are orthogonal if and only if:

$$a_x \wedge \bar{b}_y \neq b_x \wedge \bar{a}_y \quad (8)$$

$$or \quad a_z \wedge b_z = 0 \quad (9)$$

4.3 Multi-Output Shared Cube

Let A and B be cubes that that belong to different functions ($a_z \wedge b_z = 0$). The multi-output shared cube (MOSC) of A and B (written $A \mp B$) is the cube that covers the minterms common to A and B and nothing more. We obtain the MOSC of two cubes ($C = A \mp B$) in IPR-M by:

$$c_x = a_x \vee b_x \quad (10)$$

$$c_y = a_y \wedge b_y \quad (11)$$

$$c_z = a_z \vee b_z \quad (12)$$

5 Algorithms

Virtually any Boolean manipulation algorithm can be implemented using IPR-M. Below we present several frequently used algorithms for Boolean manipulation.

5.1 Single Output Expansion

This operation, represented $\zeta(F, D, f_i)$, expands every cube in F that appears in function f_i considering the on minterms in f_i and the don't care minterms in f_i.

Algorithm 1 *Single Output Expansion*

Input: Input: Function onset F, don't care set D, and single output function f_i.
Output: Expanded function for single output.

1. $C \leftarrow \emptyset$.
2. $F' = \{A \in F, a_{z_i} = 1\}$
3. $D' = \{B \in D, b_{z_i} = 1\}$
4. While $F' \neq \emptyset$ do:
   - 4.1. Select a cube $Y \in F'$.
   - 4.2. $X \leftarrow Y \diamondsuit ((F' \cup C), D')$.
   - 4.3. $C \leftarrow C \cup X$.
   - 4.4. $F' \leftarrow F' \circ X$.
5. Return C.

The operation $X = Y \diamondsuit (F, D)$ invoked in step 4.2 is called cube expansion and is performed exactly as in [1]. It expands every cube in the offset F of a function considering the minterms in the don't care set D. The operation $F' \circ X$ invoked in step 4.4 is called cube removal (defined in [1]). It removes the cubes in set F that are covered by X.

5.2 Border Cubes

The border of a cube C is to be defined as the set of cubes that exactly cover the cube C and the minterms that are at a distance of 1 from C.

Algorithm 2 *Border Cubes*
Input: Cube C.
Output: Border cube set H.
1. $H \leftarrow \emptyset$.
2. $\forall w_{x_i} \in c_x, w_{x_i} = 0$ do:
   2.1. Make $D \leftarrow C$.
   2.2. Make $d_{x_i} = 0, d_{y_i} = 1$.
   2.3. $H \leftarrow H \cup D$.
3. Return H.

5.3 Adjacent and Intersecting Cubes

Rao and Jacob [7] propose a method for determining if a cube is essential by using it's Adjacent and Intersecting Cube (AIC) set. The AIC of a prime cube A are those cubes:

1. In the onset or the dcset which are logically adjacent to A.
2. In the don't care set which intersect A.
3. In the onset which intersect A without covering A.

One way of determining which cubes are adjacent to cube A is by obtaining which cubes are at a distance of exactly 1 from cube A. However, the distance operation is costly when implemented in software as it involves counting the number of ones on a binary word. The algorithm described below utilizes the border set of cube A to determine its AIC.

Algorithm 3 *Adjacent and Intersecting Cubes*
Input: Cube A, function onset F, and don't care set D
Output: Adjacent and intersecting cube set G.
1. $G \leftarrow \emptyset$.
2. $H \leftarrow Border(A)$.
3. $\forall C \in (F \cup D)$ do:
   3.1. If $C \not\perp H$ Then
      3.1.1. If $C \not\in F$ or $A \not\subseteq C$ Then $G \leftarrow G \cup C$
4. Return G.

5.4 Essential Primes Cubes

Rao and Jacob [7] establish that, given a prime cube $A \in F$, it can be determined if A is an essential prime cube (EPC) using the following procedure:

1. For every literal a_i that appears in cube A, change the corresponding literal in each of the Adjancent and Intersecting Cubes $AIC(A)$.
2. If the new modified $AIC(A)$ does not cover A, then A is an essential prime cube.

Algorithm 4 *Essential Primes Cube*
Input: Cube A, function onset F, and don't care set D
Output: True if cube A is an EPC.
1. $B \leftarrow AIC(A)$
2. $\forall a_{y_i} \in a_y, a_{y_i} = 0$ do:
   2.1. $\forall C \in B$ do:
      2.1.1. Make $c_{x_i} = a_{x_i}, c_{y_i} = 0$.
3. Return $A \Subset C$

The operation $A \Subset C$ is called tautology test and is described in [1]. It verifies that all minters in A are covered by function C.

6 Multiple output minimization

To illustrate IPR-M's ability to support more complex operations we present the implementation of a heuristic algorithm, previously proposed by Guranath et al. [5], for minimizing multiple-output logic functions, using the algorithms described in Section 5.

The chosen heuristic algorithm uses a divide and conquer approach wherein the minimization is carried out by four main procedures:

1. *Selection of essential prime cubes* : Each cube is expanded within a single function and any redundant cubes are removed. If it is determined that the expanded cube is essential and it is shared among two or more functions or that it is orthogonal to all other functions, it will be included in the solution.

   Whenever a cube is added to the solution, it will be added to the don't care function. This will allow remaining cubes to expand, if needed, using the solution terms.

   Algorithm 5 *Select essential prime cubes*
   Input: Function onset F and don't care set D.
   Output: Set of essential cubes E and preliminary solution set S with shared and exclusive essential cubes.
   1. $E = \emptyset, S = \emptyset$.
   2. $\forall f_i \in F$ do:
      2.1. $G = \rho(\zeta(F, D, f_i))$.
      2.2. $\forall B \in G$ do:
         2.2.1. If $EPC(B)$ Then $E \leftarrow E \cup B$
   3. $\forall f_i \in F$ do:
      3.1. $\forall C \in E$ do:
         3.1.1. If $C \Subset f_i$ Or $C \perp f_i$ Then
            $S \leftarrow S \cup C$, $E \leftarrow E \circ C$,
            $D = D \cup C$.
   4. Return S, E.

2. *Selection of valid selective primes*: Selective prime cubes are detected. If a selective cube is shared by two or more functions, or if it is orthogonal to all other functions then it is included in the solution.

   To determine selective cubes we use the fact that prime cubes are either essential, selective, or redundant. The expand and irredundancy step of the previous algorithm eliminates all redundant primes. Thus, after the algorithm for essential cubes is executed, F contains only selective prime cubes.

Algorithm 6 *Selective Cubes*
Input: Function onset F, don't care set D, and preliminary solution set S from previous algorithm.
Output: Preliminary solution set S with shared and exclusive selective cubes.

1. $\forall f_i \in F$ do:
   1.1. $\forall C \in F$ do:
      1.1.1. If $C \Subset f_i$ Or $C \perp f_i$ Then
      $S \leftarrow S \cup C$, $E \leftarrow E \circ C$,
      $D = D \cup C$.
2. Return S, E.

3. *Selection of intersecting cubes*: Find the intersection among functions of the remaining essential and selective primes. The process of selecting intersecting cubes simply involves determining if a given cube $A \in f_i$ has an intersection with a cube $B \in f_j$ that covers the uncovered minterms in A and B.

Algorithm 7 *Intersecting Cubes*
Input: Function onset F, don't care set D, and preliminary solution set S from previous algorithm.
Output: Preliminary solution set S with intersecting cubes.

1. $F = F \cup E$
   1.1. $\forall X \in f_i, Y \in f_j, i \neq j$
      1.1.1. $Z = X \mp Y$
      1.1.2. If $Z \neq \emptyset$ And $(X \backslash Y) \Subset D$ And
      $(Y \backslash X) \Subset D$ Then
      $S = S \cup Z$, $F \leftarrow F \circ X$,
      $F \leftarrow F \circ Y$, $D = D \cup X$,
      $D = D \cup Y$.
2. Return S, F.

The operation $X \backslash Y$ is called cube difference [1] and it obtains the set of cubes that cover the minterms covered by X but not by Y, and nothing more.

4. *Select exclusive cubes*: The terms not yet chosen by previous steps are exclusive cubes (either essential or selective). This step expands them in their particular function.

Algorithm 8 *Select Exclusive Cubes*
Input: Function onset F, don't care set D.
Output: Solution set S.

1. $\forall f_i \in F$
   1.1. $\zeta(F, D, f_i)$
2. Return F.

7 Preliminary Results

To verify the correctness of the presented algorithms, we implemented them in C++ as part of a logic minimization program. Table 2 shows preliminary results obtained for several multiple-output examples provided with Espresso Version 2.3, when minimized with the IPR-M-based program and Espresso. For each test, the table shows the number of cubes c, input variables (In), and output variables (Out) of the multi-output function, as well as the number of cubes c and literals l in the minimized result.

As evidenced by the results, the algorithm implementation still needs some validation because, although for some of the examples the results were the same, for others Espressos' results contain considerably less literals and cubes.

Table 2. Minimization results for the IPR-based and standard Espresso algorithms.

Circuit	Input set			IPR		Espresso	
Name	c	In	Out	c	l	c	l
a2pair	16	4	3	11	32	11	32
mlp4	256	8	8	163	899	128	735
Z5xp1	128	7	10	79	312	65	287
stial	22	14	8	22	140	22	140

8 Conclusions and Future Work

IPR-M's ability to represent and manipulate multiple-output logic functions has been discussed and justified through the presentation of common multiple-output minimization operators and algorithms. To establish more meaningful comparisons with other representations (such as BDD and PCN) our future work will be directed towards analyzing the time and space complexity of IPR-M algorithms and comparing them with established methods.

References

[1] A. Diaz, M. Jimenez, E. Strangas, and M. Shanblatt. Integer pair representation of binary terms and equations. In *1998 Midwest Symposium on Circuits and Systems*, pages 172–175. IEEE, August 1998.

[2] G. D.Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, Inc., New York, NY 02061, 1994.

[3] R. Drechsler and D. Sieling. Binary decision diagrams in theory and practice. *Internal Journal of STTT*, pages 112–136, 2001.

[4] A. Friedman and P. Menon. *Theory and Design of Switching Circuits*. Computer Science Press, Inc., Maryland, USA 20850, 1975.

[5] B. Gurunath and N. Biswas. An algorithm for multiple output minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1007–1013, September 1989.

[6] H. J. Mathony. Universal logic design algorithm and its application to the synthesis of two-level switching circuits. In *Computers and Digital Techniques, IEE Proceedings*, volume 136, pages 171–177, May 1989.

[7] P. Rao and J. Jacob. A fast two-level logic minimizer. In *Proceedings Eleventh International Conference on VLSI Design, 1998*, pages 528–533, January 1998.