Chapter 2
Fast Algorithm of Attribute Reduction Based on the Complementation of Boolean Function

Grzegorz Borowik and Tadeusz Łuba

Abstract. In this chapter we propose a new method of solving the attribute reduction problem. Our method is different to the classical approach using the so-called discernibility function and its CNF into DNF transformation. We have proved that the problem is equivalent to very efficient unate complementation algorithm. That is why we propose new algorithm based on recursive execution of the procedure, which at every step of recursion selects the splitting variable and then calculates the cofactors with respect to the selected variables (Shannon expansion procedure). The recursion continues until at each leaf of the recursion tree the easily computable rules for complement process can be applied. The recursion process creates a binary tree so that the final result is obtained merging the results in the subtrees. The final matrix represents all the minimal reducts of a decision table or all the minimal dependence sets of input variables, respectively. According to the results of computer tests, better results can be achieved by application of our method in combination with the classical method.

2.1 Introduction

Often in the area of machine learning, artificial intelligence, as well as logic synthesis, we deal with some functional dependencies (for example, in the form of decision tables), in which not all attributes are necessary, i.e. some of them could be removed without loss of any information. The problem of removing redundant input attributes is known as the argument reduction problem. Its applications in the area of artificial intelligence have been studied by several researchers [22, 26].

Grzegorz Borowik · Tadeusz Łuba
Institute of Telecommunications, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: {G.Borowik,T.Luba}@tele.pw.edu.pl

R. Klempous et al. (eds.), *Advanced Methods and Applications in Computational Intelligence*, Topics in Intelligent Engineering and Informatics 6,
DOI: 10.1007/978-3-319-01436-4_2,

Methods of attribute reduction have proved very useful in many applications. As an example, a data base containing information about 122 patients with duodenal ulcers, treated by highly selective vagotomy, was analyzed by Słowinski and Sharif [27]. This database contained information needed to classify patients according to the long term results of the operation. From 11 attributes describing the patient's health state, after reduction it turned out that only 5 of them were enough to ensure a satisfactory quality of classification.

Such results indicate that application of argument reduction could also be useful in logic synthesis [20], where circuits performance can be presented as truth tables which are in fact decision tables with two valued attributes, where condition attributes are in fact input variables, and decision ones are to represent output variables of the circuit. In the practical application of Boolean algebra the key problem is to represent Boolean functions by formulas which are as simple as possible. One approach to this simplification is to minimize the number of variables appearing in truth table explicitly. Then, the reduced set of input variables is used in other optimization algorithms, e.g. logic minimization and logic decomposition. Combined with other design techniques, argument reduction allows for great size reduction of implemented circuits [19, 20].

A number of methods for discovering reducts have already been proposed in the literature [1, 2, 7, 8, 10, 11, 12, 13, 14, 15, 17, 21, 23, 26, 28, 29]. The most popular Algorithms for Discovering Rough Set Reducts methods are based on discernibility matrices [26]. Besides mutual information and discernibility matrix based attribute reduction methods, they have developed some efficient reduction algorithms based on CI tools of genetic algorithm, ant colony optimization, simulated annealing, and others [11]. These techniques have been successfully applied to data reduction, text classification and texture analysis [17].

Interestingly, another potentially very promising area of application of arguments reduction algorithm is logic systems designing. This is because the novel hardware building blocks impose limitations on the size of circuits that can be implemented with them. The concept of argument reduction was introduced and effectively applied in the balanced decomposition method [16, 24]. Based on redundant variable analysis of each output of a multi-output Boolean function, parallel decomposition separates F into two or more functions, each of which has as its inputs and outputs a subset of the original inputs and outputs. It was proved that parallel decomposition based on argument reduction process plays a very important role in FPGA based synthesis of digital circuits.

Recently, we have proposed the Arguments/Attributes Minimizer [4] which computes reducts of a set of attributes of knowledge representation in information systems or reduces the number of input variables in logic systems. The algorithm is based on unate complementation concept [5]. This chapter extends the results obtained in [4]. Here, we propose new algorithm based on recursive execution of the procedure, which at every step of recursion selects the splitting variable and then calculates the cofactors with respect to the selected variables (Shannon expansion procedure). The recursion continues until, at each leaf of the recursion tree, the easily computable rules for complement process can be applied. The recursion process

creates a binary tree so that the final result is obtained merging the results obtained in the subtrees (merging procedure). The final matrix represents all the minimal reducts of a decision table or all the minimal dependence sets of input variables, respectively.

The chapter begins with an overview of basic notions of information systems, functional dependencies, decision tables and reducts. In section 2.2 and 2.3 we discuss relations between multi-valued logic and decision table systems with respect to classification of data with missing values. Particularly, it is shown that elimination of attributes can be easily obtained using standard procedures used in logic synthesis. New contribution is presented in section 2.4, where we describe how to apply complementation algorithm and provide new variant of the attribute reduction process.

2.2 Preliminary Notions

The information system contains data about objects characterized by certain attributes, where two classes of attributes are often distinguished: condition and decision attributes (in logic synthesis they are usually called input and output variables). Such an information system is called a decision system and it is usually specified by a decision table (in logic synthesis it is called a truth table). The decision table describes conditions that must be satisfied in order to carry out the decisions specified for them. More formally, an *information system* is a pair $\mathscr{A} = (U, A)$, where U is a nonempty set of objects (in logic synthesis: minterms) called the universe, and A is a nonempty set of attributes (variables). If we distinguish in an information system two disjoint classes of attributes, condition (A) and decision (D) attributes (input and output variables), where $A \cap D = \emptyset$, then the system is called a *decision system* $\mathscr{A} = (U, A \cup D)$. Any information or decision table defines a function ρ that maps the direct product of U and A (U and $A \cup D$, respectively) into the set of all values.

The attribute values $\rho_{pi} = \rho(u_p, a_i)$ and $\rho_{qi} = \rho(u_q, a_i)$ are called *compatible* ($\rho_{pi} \sim \rho_{qi}$) if, and only if, $\rho_{pi} = \rho_{qi}$ or $\rho_{pi} = *$ or $\rho_{qi} = *$, where "$*$" represents the case when attribute value is "do not care". On the other hand, if ρ_{pi} and ρ_{qi} are defined and are "different" it is said that ρ_{pi} is *not compatible* with ρ_{qi} and is denoted as $\rho_{pi} \nsim \rho_{qi}$. The consequence of this definition is a COM relation defined as follows:

Let $B \subseteq A$ and $u_p, u_q \in U$. The objects $p, q \in \mathrm{COM}(B)$ if and only if $\rho(u_p, a_i) \sim \rho(u_q, a_i)$ for every $a_i \in B$.

The objects u_p and u_q, which belong to the relation COM(B), are said to be *compatible in the set* B. Compatible objects in the set $B = A$ are simply called *compatible*.

The compatibility relation of objects is a tolerance relation (reflexive and symmetric) and hence it generates compatible classes on the set of objects U.

Compatibility relation allows us to classify objects but the classification classes do not form partitions on the set U, as it is in the case of indiscernibility relation

(IND) [9]. COM(B) classifies objects grouping them into compatibility classes, i.e. $U/\mathrm{COM}(B)$, where $B \subseteq A$.

For the sake of simplicity, collection of subsets $U/\mathrm{COM}(B)$ we call *r-partition* on U and denote as COM(B).

R-partition can be used as a tool to classify objects of a data table description. It can be shown that the r-partition concept is a generalization of the ideas of partitioning a set into consistent classes and partitioning a set into tolerance classes. Therefore, partitioning a set U into consistent classes of certain relation $\mathscr{R}$ and partitioning a set into tolerance classes of a certain relation $\mathscr{T}$ can be treated as special cases of r-partitioning.

R-partition on a set U may be viewed as a collection of non-disjoint subsets of U, where the set union is equal U; and all symbols and operations of partition algebra are applicable to r-partitions. Therefore, convention used for denoting r-partitions and their typical operators are the same as in the case of partitions. We assume the reader's familiarity with these r-partition concepts which are simple extensions of partition algebra [18].

Especially the relation *less than or equal to* holds between two r-partitions Π_1 and Π_2 ($\Pi_1 \leq \Pi_2$) iff for every block of Π_1, in short denoted by $\mathscr{B}_i(\Pi_1)$, there exists a $\mathscr{B}_j(\Pi_2)$, such that $\mathscr{B}_i(\Pi_1) \subseteq \mathscr{B}_j(\Pi_2)$. If Π_1 and Π_2 are partitions, this definition reduces to the conventional ordering relation between two partitions.

The r-partition generated by a set B is the product of r-partitions generated by the attributes $a_i \in B$:

$$\Pi(B) = \bigcap_i \Pi(a_i).$$

If $B = \{a_{i_1}, \ldots, a_{i_k}\}$, the product can be expressed as: $\Pi(B) = \Pi(a_{i_1}) \cdot \ldots \cdot \Pi(a_{i_k})$.

Let Π_1 and Π_2 are r-partitions and $\Pi_1 \leq \Pi_2$. Then a r-partition $\Pi_1|\Pi_2$, whose elements are blocks of Π_2 and whose blocks are those of Π_1, is the *quotient r-partition* of Π_1 over Π_2.

Example. For the decision system shown in Table 2.1

$$\Pi(a_1) = \{\overline{1,3,5,6,7};\ \overline{2,4,5};\ \overline{5,8}\},$$

$$\Pi(a_6) = \{\overline{1,4,7,8};\ \overline{2,3,5,6,7}\},$$

$$\Pi(d) = \{\overline{1,5};\ \overline{2,3,4};\ \overline{6};\ \overline{7,8}\}.$$

Therefore, for the set $B = \{a_1, a_6\}$,

$$\Pi(B) = \Pi(a_1) \cdot \Pi(a_6) = \{\overline{1,7};\ \overline{3,5,6,7};\ \overline{4};\ \overline{2,5};\ \overline{8};\ \overline{5}\},$$

and the quotient r-parttion $\Pi(B)|\Pi(d)$ is

$$\Pi(B)|\Pi(d) = \{\overline{(1),(7)};\ \overline{(3),(5),(6),(7)};\ \overline{(4)};\ \overline{(2),(5)};\ \overline{(8)};\ \overline{(5)}\}.$$

Table 2.1. Example of a decision system

	a_1	a_2	a_3	a_4	a_5	a_6	d
1	0	0	1	1	0	0	1
2	1	$*$	2	0	1	1	2
3	0	1	1	0	0	1	2
4	1	2	2	$*$	2	0	2
5	$*$	2	2	2	0	1	1
6	0	0	1	1	0	1	3
7	0	1	0	3	2	$*$	4
8	2	2	2	3	2	0	4

2.3 Elimination of Input Variables

In this section the process of searching and elimination of redundant arguments is described using concepts of logic systems, however appropriate simplifications caused by functional dependency features as well as the generalization to the case of r-partition have been efficiently applied.

An argument $x \in X$ is called *dispensable* in a logic specification of function F iff $\Pi(X - \{x\}) \leq \Pi(F)$, otherwise, i.e. $\Pi(X - \{x\}) \not\leq \Pi(F)$, an argument is called *indispensable* (i.e. an essential variable).

The meaning of an indispensable variable is similar to that of a core attribute, i.e. these are the most important variables. In other words, no indispensable variable can be removed without destroying the consistency of the function specification. Thus, the set of all indispensable arguments is called a *core* of X and is denoted as $\mathrm{CORE}(X)$.

In order to find the core set of arguments we have to eliminate an input variable and then verify whether the corresponding partition inequality holds. A key theorem is stated below to make this procedure more efficient.

First of all we reformulate this problem to apply more useful tools which are efficiently used in switching theory [18, 19, 20].

A set $B = \{b_1, \ldots, b_k\} \subseteq X$ is called a *minimal dependences set*, i.e. *reduct*, of a Boolean function F iff $\Pi(B) \leq \Pi(F)$, and there is no proper subset B' of B, such that $\Pi(B') \leq \Pi(F)$.

It is evident that an indispensable input variable of function F is an argument of every minimal dependence set of F.

Now we introduce two basic notions, namely discernibility set and discernibility function, which help us to construct an efficient algorithm for attribute reduction process.

Let $\mathscr{A} = (U, A \cup D)$ be a decision system. Let u_p, u_q $(u_p \neq u_q)$ are objects of U, such that $d \in D$ is a decision attribute and $\rho(u_p, d) \nsim \rho(u_q, d)$. By $\{c_{pq}\}$, we denote a set of attributes called a *discernibility set* which is defined as follows:

$$c_{pq} = \{a_i \in A : \rho(u_p, a_i) \nsim \rho(u_q, a_i)\}. \tag{2.1}$$

A *discernibility function* $f_{\mathscr{A}}$ for a decision system $\mathscr{A}$ is a Boolean function of m Boolean attributes $\widehat{a}_1,\ldots,\widehat{a}_m$ corresponding to the attributes $a_1,\ldots,a_m$, defined by the conjunction of all expressions $\vee(\widehat{c}_{pq})$, where $\vee(\widehat{c}_{pq})$ is the disjunction of all attributes $\widehat{c}_{pq} = \{\widehat{a}:\ a \in c_{pq}\}$, $1 \leq p < q \leq n$.

A strong relation between the notion of a reduct of function $F(RED(X))$ and prime implicant of the monotonic Boolean function f_F:

$$\{x_{i_1},\ldots,x_{i_k}\} \in RED(X) \ \text{ iff } \ x_{i_1} \wedge \cdots \wedge x_{i_k} \text{ is a prime implicant of } f_F.$$

was investigated among others by Skowron and Kryszkiewicz [14, 26]:

Example. Using the previously calculated quotient r-parttion $\Pi(B)|\Pi(d)$:

$$\Pi(B)|\Pi(d) = \{\overline{(1),(7)};\ \overline{(3),(5),(6),(7)};\ \overline{(4)};\ \overline{(2),(5)};\ \overline{(8)};\ \overline{(5)}\}$$

we can conclude that the only pairs of objects which should be separated by condition attributes are shown in the Table 2.2. For each pair u_p, u_q the set of all distinguishing attributes is calculated. It is easy to observe that the discerniblity function expressed in CNF is as

$$f_M = (a_2 + a_4)(a_4 + a_5).$$

Table 2.2. Pairs of objects and corresponding separations

p,q	attributes
1,7	$\{a_2,a_3,a_4,a_5\}$
3,5	$\{a_2,a_3,a_4\}$
3,6	$\{a_2,a_4\}$
3,7	$\{a_3,a_4,a_5\}$
5,6	$\{a_2,a_3,a_4\}$
5,7	$\{a_2,a_3,a_4,a_5\}$
6,7	$\{a_2,a_3,a_4,a_5\}$
2,5	$\{a_4,a_5\}$

Transforming CNF into DNF

$$f_M = a_4 + a_2a_5$$

we conlude that all reducts for DT from Table 2.1 are

- $\{a_1,a_4,a_6\}$,
- $\{a_1,a_2,a_5,a_6\}$.

Noticeably, minimization of the discernibility function is carried out by transforming the conjunctive normal form (in which it is originally constructed) into the disjunctive normal form and finding a minimum implicant. Such a transformation is usually time-consuming. Therefore, it is important to look for efficient algorithms which can handle this task.

2.4 Computing Minimal Sets of Attributes Using COMPLEMENT Algorithm

A proposed approach is based on the fact that transformation of a conjunctive normal form into the disjunctive normal form can be reduced to the task of complementation of monotone/unate Boolean function which is intimately related to the concept of a column cover of the binary matrix. The unate complementation was proposed by Brayton [5], and is used in logic minimization algorithms as unate recursive paradigm. The unate recursive paradigm exploits properties of unate functions, while performing recursive decomposition. Therefore, to obtain discernibility function in the minimal DNF we apply the fast complementation algorithm adopted from *Espresso* system [5].

To this end, we describe the collection $\{c_{pq}\}$ from (2.1), of all c_{pq} sets in the form of the binary matrix M for which an element M_{ij} is defined as follows:

$$M_{ij} = \begin{cases} 1, \text{ if } a_j \in c_{pq_i}, \\ 0, \text{ otherwise,} \end{cases} \tag{2.2}$$

$i = 1,\ldots,t = \text{CARD}(\{c_{pq}\})$, $j = 1,\ldots,m = \text{CARD}(A)$. Then,

Theorem [5]. Each row i of $\overline{M}$, the binary matrix complement of M, corresponds to a column cover L of M, where $j \in L$ if and only if $\overline{M}_{ij} = 1$.

Column cover means that every row of M contains a "1" in some column which appears in L. The rows of $\overline{M}$ include the set of all minimal column covers of M. If $\overline{M}$ was minimal with respect to containment, then $\overline{M}$ would precisely represent the set of all minimal column covers of M.

More precisely, a *column cover* of binary matrix M is defined as a set L of columns such that for every i

$$\sum_{j \in L} M_{ij} \geq 1. \tag{2.3}$$

Our goal is to select an optimal set L of arguments corresponding to columns of M. Covers L of M are in one-to-one correspondence with the reduced subsets of arguments, i.e. reducts.

The task of searching the minimal column cover is based on the Shannon expansion of Boolean function f:

$$f = \overline{x}_j f_{\overline{x}_j} + x_j f_{x_j}, \tag{2.4}$$

where $f_{x_j}, f_{\overline{x}_j}$ are *cofactors* of f with respect to splitting variable x_j, i.e. the results of substituting "1" and "0" for x_j in f. It is the key to a fast recursive complementation process.

On the basis of [5] a logic function f is *monotone increasing* (*monotone decreasing*) in a variable x_j if changing x_j from 0 to 1 causes all the outputs of f that change, to increase also from 0 to 1 (from 1 to 0). A function that is either monotone increasing or monotone decreasing in x_j is said to be *monotone* or *unate* in x_j.

A function is *unate* (*monotone*) if it is unate in all its variables. For example, the function $f = x_1\overline{x}_2 + \overline{x}_2 x_3$ is unate since it is increasing in x_1 and x_3, and decreasing in x_2.

An important aspect of the algorithm is that properties of unate functions are exploited to simplify or terminate the recursion. Unate functions have special properties which make them especially useful.

Applying the property of unateness the equation (2.4) can be expressed as simplified formulas, i.e. when function is monotone decreasing, then $f_{x_j} \subseteq f_{\overline{x}_j}$, and therefore

$$f = \overline{x}_j f_{\overline{x}_j} + f_{x_j} \tag{2.5}$$

and when function is monotone increasing, then $f_{x_j} \supseteq f_{\overline{x}_j}$, and therefore

$$f = f_{\overline{x}_j} + x_j f_{x_j} \tag{2.6}$$

Stating that discernibility function f_M is in CNF, proposed approach benefits from the transformation (2.7), i.e. double complementation of a Boolean function.

$$\prod_k \sum_l x_{kl} = \overline{\overline{\prod_k \sum_l x_{kl}}} = \overline{\sum_k \prod_l \overline{x_{kl}}} \tag{2.7}$$

Given that the discernibility function f_M representing the CNF is unate, then by applying De Morgan's law it can be transformed into $F = \overline{f}_M$ (first complementation) and then considered as a binary matrix M. Then, each row of M corresponds to the conjunction of negative literals $\overline{x}_j$ of F, i.e. then $M_{ij} = 1$. In fact, the task of searching the complement of function F, i.e. $\overline{F}$, can be reduced to the concept of searching of a column covers (represented by matrix C) of the binary matrix M (second complementation).

Given an initial matrix M – the initial cover of a unate Boolean function F, for simplicity we call the *cover of function* F – the algorithm recursively splits the matrix into smaller pieces until, at each leaf of the recursion tree, the easily computable termination rules (described in Subsection 2.4.1) can be applied. If a basic termination rule applies, then the appropriate cover is returned immediately. If no basic termination rule applies, the initial cover of F is cofactored by both $\overline{x}_j$ and x_j, and the algorithm is recursively called. Each recursive call returns a complemented cover, i.e. $\overline{F}_{\overline{x}_j}$ and $\overline{F}_{x_j}$.

Then, the results are reassembled into a final solution. For the complementation algorithm, the result is the complement of the initial cover M.

To assemble the final result, the complemented covers are merged using following formulas, i.e. for monotone decreasing function

$$\overline{f} = \overline{f}_{\overline{x}_j} + x_j \overline{f}_{x_j} \tag{2.8}$$

and for monotone increasing function

$$\overline{f} = \overline{x}_j \overline{f}_{\overline{x}_j} + \overline{f}_{x_j} \tag{2.9}$$

Noticeably, function F is monotone decreasing for all literals, therefore only equations (2.5) and (2.8) are considered in the presented approach.

2.4.1 Unate Complementation

In the unate complementation algorithm calculations are organized in a top-down synthesis process to obtain the required final complement of function F. At each node of the binary recursion tree the splitting variable x_j is chosen and both cofactors F_{x_j}, $F_{\overline{x}_j}$ are calculated.

In order to identify the splitting variables we choose them among the shortest terms in F, i.e. the object with the maximum number of "do not care" values. Having identified the splitting variables, we have to decide the order in which these variables are processed. This order is made by choosing first the variables that appear most often in the other terms of F. Such procedure eliminates the largest number of rows of matrix M in one of the branches of the recursion.

The cofactor with respect to $\overline{x}_j$ is obtained by setting up the j-th column to "0", and the cofactor with respect to x_j, is obtained by excluding all the rows for which the j-th element is equal to 1.

Stating that matrix M is the cover function F, the recursion continues until at each leaf of the recursion tree one of the basic termination rules is encountered:

- The cover of F is empty. A cover is empty if F contains no terms. In this case, its complement is a tautology. Hence, a cover containing the universal cube [5] is returned. Then, the resulting cover contains one row of all 0's.
- F includes the universal cube, i.e. M contains a row of all 0's. Here, F is a tautology, so its complement is empty. The empty cover is returned (no terms).
- F contains a single term. Here, the complement of F can be computed directly using De Morgan's law. After complementation, $\overline{F}$ contains term(s) of one variable only. Then, corresponding cover is returned.

In each step of the recursive complementation algorithm, termination conditions are checked; if they are not satisfied, recursion is performed. The complementation algorithm returns an actual cover (the complement of the initial cover).

Example. The influence of the unate recursive algorithm on the final result of the attribute reduction process is explained with function f_M represented by a following CNF:

$$f_M = x_4x_6(x_1+x_2)(x_3+x_5+x_7)(x_2+x_3)(x_2+x_7).$$

Hence, omitting indispensable variables:

$$f'_M = (x_1+x_2)(x_3+x_5+x_7)(x_2+x_3)(x_2+x_7),$$

and complementing

$$F = \overline{f'}_M = \overline{x}_1\overline{x}_2 + \overline{x}_3\overline{x}_5\overline{x}_7 + \overline{x}_2\overline{x}_3 + \overline{x}_2\overline{x}_7.$$

Then, the initial cover

$$M = \begin{array}{c} x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7 \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

In order to identify the splitting variables we choose them among the shortest terms in F. Here we select the first term, yielding variables x_1 and x_2. Since the variable that appears most often in the other terms of F is x_2, we decide to choose this one.

Now we compute the cofactors of F with respect to the variable x_2. This is illustrated in Fig. 2.1 by two arrows with the common starting point going to different directions. The current tree has two leaves: the larger represents cofactor $F_{\overline{x}_2}$ and the smaller one represents F_{x_2}.

The smaller cofactor is the subject to special case resulting in three objects of the complement, because F_{x_2} cofactor has only one row. The larger cofactor is again decomposed yielding cofactor for $\overline{x}_1$ (left hand side matrix), and for x_1 (right hand side matrix). For the left hand side component we again can apply special case, now resulting in empty cover – there is a row of all 0's. Thus, the next step deals with the right hand side function matrix, which can be expanded onto two matrices, with easily calculated complements.

To illustrate the merging operation of the unate recursive process, consider the actions taken at the nodes x_7, x_1, x_2. Applying formula (2.8) at the nodes x_7, x_1, x_2 we calculate following complements, denoted C_7, C_1, C_2:

$$C_7 = x_7 \cdot [0\,0\,1\,0\,0\,0\,0] + \emptyset = [0\,0\,1\,0\,0\,0\,1]$$

$$C_1 = x_1 \cdot [0\,0\,1\,0\,0\,0\,1] + \emptyset = [1\,0\,1\,0\,0\,0\,1]$$

$$C_2 = x_2 \cdot \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} + [1\,0\,1\,0\,0\,0\,1] = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Resulted complement C_2 together with indispensable variables x_4 and x_6 represents the following reducts:

- $\{x_2, x_3, x_4, x_6\}$,
- $\{x_2, x_4, x_5, x_6\}$,
- $\{x_2, x_4, x_6, x_7\}$,
- $\{x_1, x_3, x_4, x_6, x_7\}$.

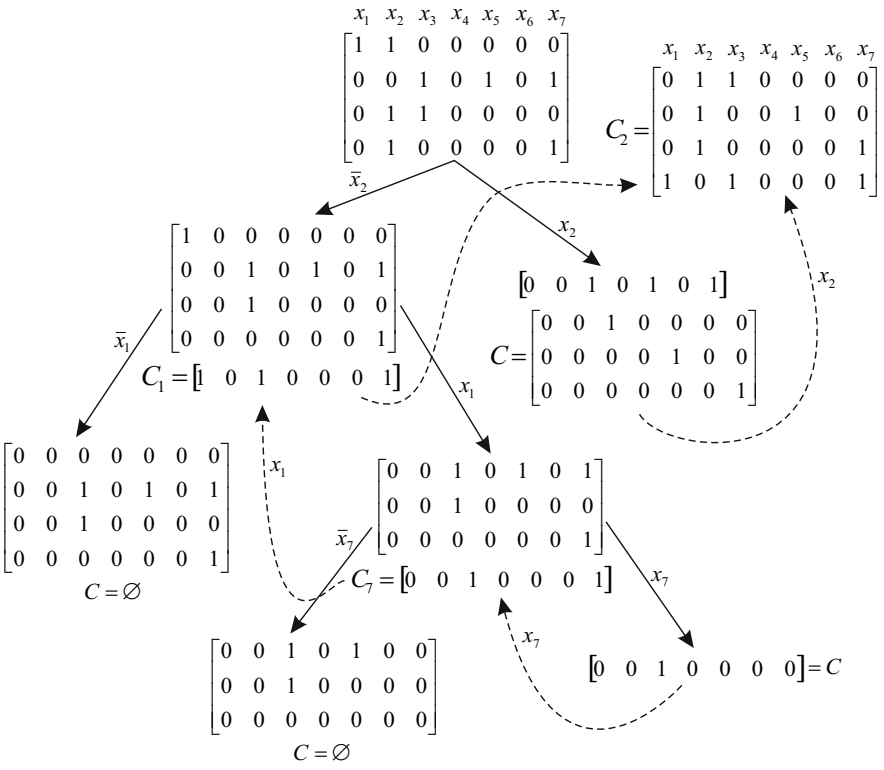

Fig. 2.1. Complementation scheme of $F=\bar{x}_1\bar{x}_2+\bar{x}_3\bar{x}_5\bar{x}_7+\bar{x}_2\bar{x}_3+\bar{x}_2\bar{x}_7$

This agrees with the result that can be obtained applying fundamental transformations of Boolean algebra:

$$f_M=(x_1+x_2)(x_3+x_5+x_7)(x_2+x_3)(x_2+x_7),$$

hence, performing the multiplication and applying absorption law

$$(x_1+x_2)(x_3+x_5+x_7)(x_2+x_3)(x_2+x_7)=$$

$$(x_2+x_1)(x_2+x_3)(x_2+x_7)(x_3+x_5+x_7)=$$

$$(x_2+x_1x_3x_7)(x_3+x_5+x_7)=$$

$$x_2x_3+x_2x_5+x_2x_7+x_1x_3x_7$$

we obtain the same set of reducts.

2.5 Experimental Results

Many computer data mining systems were developed. In particular, the well-known Rough Set Exploration System elaborated at the University of Warsaw. The system implements many advanced procedures. Some algorithms of RSES have also been embedded in the even more famous ROSETTA system located in the Biomedical Center in Sweden [31, 32].

ROSE2 (Rough Sets Data Explorer) is a software implementing basic elements of the rough set theory and rule discovery techniques. It has been developed at the Laboratory of Intelligent Decision Support Systems of the Institute of Computing Science in Poznań, basing on fourteen-year experience in rough set based knowledge discovery and decision analysis. All computations are based on rough set fundamentals introduced by Z. Pawlak [22] with some modifications proposed by Słowiński and Ziarko [30].

These tools were used to compare them with the presented synthesis metod (Table 2.3). Experiments performed show that despite many efforts directed to the designing of an effective tools for attribute reduction, existing tools are not efficient.

The confirmation of this supposition are experiments with the system RSES. Our research has shown that this system cannot process data tables with large number of indeterminacy. An important example from the literature is *trains* database. Applying the new method we generate 689 reducts, however RSES 333 only (not selecting the option "Do not discern with missing values"). While running the system with the option "Do not discern with missing values" selected it ends up after several hours of computation yielding "Not enough memory" message. Note, that not selecting the option the tool takes into account missing values when calculating the discernibility matrix, which yields a result that is different from the set of all minimal sets of attributes.

Another example confirming the absolute superiority of the proposed method is *kaz* function. It is the binary function of 21 arguments, used when testing advanced logic synthesis tools. RSES calculates all the 5574 reducts within 70 minutes. In

Table 2.3. Results of analysis the proposed method in comparison to RSES data mining system

database	attributes	instances	RSES	compl. method
house	17	232	1s	187ms
breast-cancer-wisconsin	10	699	2s	823ms
kaz	22	31	70min	234ms
trains	33	10	out of memory (5h 38min)	6ms
agaricus-lepiota-mushroom	23	8124	29min	4min 47s
urology	36	500	out of memory (12h)	42s 741ms
audiology	71	200	out of memory (1h 17min)	14s 508ms
dermatology	35	366	out of memory (3h 27min)	3min 32s
lung-cancer	57	32	out of memory (5h 20min)	111h 57min

comparison, the new procedure developed and implemented by the authors calculates the set of all reducts in 234ms.

Presented method was additionally proved on the typical databases of medicine, i.e. audiology database, dermatology database, urology database, breast cancer database and lung cancer database. Table 2.3 shows the computation time for all the minimum sets of attributes.

The experiments performed confirm that logic synthesis algorithms developed for the design of digital systems are much more effective than currently used algorithms in data mining systems.

Undoubtedly, in logic synthesis systems and hardware realizations we are almost always looking for these sets of arguments (reducts) which are both: minimal and least. However, the decision systems depend on all the minimal sets of attributes. For example, when considering a reduct of the least cardinality, it can include an attribute that its implementation is actually expensive. In particular, when we consider calculations for a medical diagnosis, it may be a parameter which express complicated or expensive examination, or a test which may have a negative impact on the health of the patient and it is not possible to carry out. Therefore, the reducts of higher cardinality could be sometimes easier to be applied/used in practice.

The importance of calculating the minimum reducts is explained on simple decision table representing the results of medical examination and diagnosis for the seven patients (Tab. 2.4).

Table 2.4. Example of a decision system

	exam_1	exam_2	exam_3	exam_4	exam_5	exam_6	exam_7	exam_8	diagnosis
p_1	1	1	1	1	1	0	0	0	d_3
p_2	0	1	1	1	1	1	0	1	d_3
p_3	1	1	2	1	0	0	0	0	d_3
p_4	0	1	1	0	0	0	0	0	d_1
p_5	0	0	0	1	0	0	1	0	d_2
p_6	0	1	1	1	0	1	0	0	d_3
p_7	0	1	2	1	0	0	0	1	d_3

The use of such data involves the induction of decision rules. Based on the rules induced the initial decisions of patient's health status are made. For Table 2.4 decision rules induced with Rough Set Exploration System [32] are as follows:

- $(\text{exam}_2 = 1)\&(\text{exam}_4 = 1) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_4 = 1)\&(\text{exam}_7 = 0) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_3 = 1)\&(\text{exam}_4 = 1) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_1 = 1) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_5 = 1) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_6 = 1) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_8 = 1) \Rightarrow (\text{diagnosis} = d_3)$,

- $(\text{exam}_3 = 2) \Rightarrow (\text{diagnosis} = d_3)$,
- $(\text{exam}_4 = 0) \Rightarrow (\text{diagnosis} = d_1)$,
- $(\text{exam}_1 = 0)\&(\text{exam}_3 = 1)\&(\text{exam}_6 = 0) \Rightarrow (\text{diagnosis} = d_1)$,
- $(\text{exam}_1 = 0)\&(\text{exam}_2 = 1)\&(\text{exam}_6 = 0)\&(\text{exam}_8 = 0) \Rightarrow (\text{diagnosis} = d_1)$,
- $(\text{exam}_1 = 0)\&(\text{exam}_6 = 0)\&(\text{exam}_7 = 0)\&(\text{exam}_8 = 0) \Rightarrow (\text{diagnosis} = d_1)$,
- $(\text{exam}_3 = 1)\&(\text{exam}_5 = 0)\&(\text{exam}_6 = 0) \Rightarrow (\text{diagnosis} = d_1)$,
- $(\text{exam}_2 = 0) \Rightarrow (\text{diagnosis} = d_2)$,
- $(\text{exam}_3 = 0) \Rightarrow (\text{diagnosis} = d_2)$,
- $(\text{exam}_7 = 1) \Rightarrow (\text{diagnosis} = d_2)$,
- $(\text{exam}_1 = 0)\&(\text{exam}_4 = 1)\&(\text{exam}_6 = 0)\&(\text{exam}_8 = 0) \Rightarrow (\text{diagnosis} = d_2)$.

On the basis of the rule $(\text{exam}_2 = 1)\&(\text{exam}_4 = 1) \Rightarrow (\text{diagnosis} = d_3)$ it can be concluded that the patient whose all eight examinations are "1" should be diagnosed with d_3. However, the rule $(\text{exam}_7 = 1) \Rightarrow (\text{diagnosis} = d_2)$ suggests the diagnosis d_2. This situation means that in order to make a correct diagnosis additional examination should be performed.

In fact, we may find that some of the tests are expensive or difficult to be carried out. However, reduction of attributes can enable the elimination of troublesome examinations. For example, after reducing the attributes in this example decision system we get following minimum sets of attributes:

- $\{\text{exam}_1, \text{exam}_4, \text{exam}_6, \text{exam}_8\}$,
- $\{\text{exam}_1, \text{exam}_6, \text{exam}_7, \text{exam}_8\}$,
- $\{\text{exam}_3, \text{exam}_4\}$,
- $\{\text{exam}_3, \text{exam}_5, \text{exam}_6\}$,
- $\{\text{exam}_4, \text{exam}_7\}$,
- $\{\text{exam}_2, \text{exam}_4\}$,
- $\{\text{exam}_1, \text{exam}_3, \text{exam}_6\}$,
- $\{\text{exam}_1, \text{exam}_2, \text{exam}_6, \text{exam}_8\}$.

For example, if exam_4 is difficult to be performed, we can choose a set (reduct) that does not include this examination, i.e. $\{\text{exam}_3, \text{exam}_5, \text{exam}_6\}$. It is worth noting that the choice of the smallest set of attributes is not always a good option and it is important to calculate all the minimal sets of attributes. Selection of the set with greater number of attributes can cause that elements included are easier to be put into practice.

2.6 Conclusion

The argument reduction problem is of a great importance in logic synthesis. It is the basis of some functional transformations, such as parallel decomposition [24]. Combined with some other design techniques it allows us to reduce the size of implemented circuits.

In this chapter we have described an important problem of attribute reduction. This concept, originating from artificial intelligence (namely the theory of rough

sets), helps to deal with functional dependencies having redundant input attributes. We have presented a new exact algorithm for attribute reduction which is based on the unate complementation task. Experimental results which have been obtained using this approach proved that it is a valuable method of processing the databases.

Acknowledgements. This work was partly supported by the Foundation for the Development of Radiocommunications and Multimedia Technologies.

References

1. Abdullah, S., Golafshan, L., Nazri, M.Z.A.: Re-heat simulated annealing algorithm for rough set attribute reduction. International Journal of the Physical Sciences 6(8), 2083–2089 (2011), doi:10.5897/IJPS11.218
2. Bazan, J., Nguyen, H.S., Nguyen, S.H., Synak, P., Wróblewski, J.: Rough set algorithms in classification problem. In: Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems, vol. 56, pp. 49–88. Physica-Verlag, Heidelberg (2000), doi:10.1007/978-3-7908-1840-6_3
3. Borowik, G., Łuba, T.: Attribute reduction based on the complementation of boolean functions. In: 1st Australian Conference on the Applications of Systems Engineering, ACASE 2012, Sydney, Australia, pp. 58–59 (2012) (electronic document)
4. Borowik, G., Łuba, T., Zydek, D.: Features reduction using logic minimization techniques. International Journal of Electronics and Telecommunications 58(1), 71–76 (2012), doi:10.2478/v10177-012-0010-x
5. Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers (1984)
6. Brzozowski, J.A., Łuba, T.: Decomposition of boolean functions specified by cubes. Journal of Multi-Valued Logic & Soft Computing 9, 377–417 (2003)
7. Dash, R., Dash, R., Mishra, D.: A hybridized rough-PCA approach of attribute reduction for high dimensional data set. European Journal of Scientific Research 44(1), 29–38 (2010)
8. Feixiang, Z., Yingjun, Z., Li, Z.: An efficient attribute reduction in decision information systems. In: International Conference on Computer Science and Software Engineering, Wuhan, Hubei, pp. 466–469 (2008), doi:10.1109/CSSE.2008.1090
9. Grzenda, M.: Prediction-oriented dimensionality reduction of industrial data sets. In: Mehrotra, K.G., Mohan, C.K., Oh, J.C., Varshney, P.K., Ali, M. (eds.) IEA/AIE 2011, Part I. LNCS, vol. 6703, pp. 232–241. Springer, Heidelberg (2011)
10. Hedar, A.R., Wang, J., Fukushima, M.: Tabu search for attribute reduction in rough set theory. Journal of Soft Computing – A Fusion of Foundations, Methodologies and Applications 12(9), 909–918 (2008), doi:10.1007/s00500-007-0260-1
11. Jensen, R., Shen, Q.: Semantics-preserving dimensionality reduction: Rough and fuzzy-rough based approaches. IEEE Transactions on Knowledge and Data Engineering 16, 1457–1471 (2004), doi:10.1109/TKDE.2004.96
12. Jing, S., She, K.: Heterogeneous attribute reduction in noisy system based on a generalized neighborhood rough sets model. World Academy of Science, Engineering and Technology 75, 1067–1072 (2011)

13. Kalyani, P., Karnan, M.: A new implementation of attribute reduction using quick relative reduct algorithm. International Journal of Internet Computing 1(1), 99–102 (2011)
14. Kryszkiewicz, M., Cichoń, K.: Towards scalable algorithms for discovering rough set reducts. In: Peters, J.F., Skowron, A., Grzymała-Busse, J.W., Kostek, B.Z., Swiniarski, R.W., Szczuka, M.S. (eds.) Transactions on Rough Sets I. LNCS, vol. 3100, pp. 120–143. Springer, Heidelberg (2004)
15. Kryszkiewicz, M., Lasek, P.: FUN: Fast discovery of minimal sets of attributes functionally determining a decision attribute. In: Peters, J.F., Skowron, A., Rybiński, H. (eds.) Transactions on Rough Sets IX. LNCS, vol. 5390, pp. 76–95. Springer, Heidelberg (2008)
16. Lewandowski, J., Rawski, M., Rybiński, H.: Application of parallel decomposition for creation of reduced feed-forward neural networks. In: Kryszkiewicz, M., Peters, J.F., Rybiński, H., Skowron, A. (eds.) RSEISP 2007. LNCS (LNAI), vol. 4585, pp. 564–573. Springer, Heidelberg (2007)
17. Lin, T.Y., Yao, Y.Y., Zadeh, L.A. (eds.): Data mining, rough sets and granular computing. Physica-Verlag GmbH, Heidelberg (2002)
18. Łuba, T., Lasocki, R.: On unknown attribute values in functional dependencies. In: Proceedings of The Third International Workshop on Rough Sets and Soft Computing, San Jose, pp. 490–497 (1994)
19. Łuba, T., Lasocki, R., Rybnik, J.: An implementation of decomposition algorithm and its application in information systems analysis and logic synthesis. In: Ziarko, W. (ed.) Rough Sets, Fuzzy Sets and Knowledge Discovery. Workshops in Computing Series, pp. 458–465. Springer (1994)
20. Łuba, T., Rybnik, J.: Rough sets and some aspects in logic synthesis. In: Słowiński, R. (ed.) Intelligent Decision Support – Handbook of Application and Advances of the Rough Sets Theory. Kluwer Academic Publishers (1992)
21. Nguyen, D., Nguyen, X.: A new method to attribute reduction of decision systems with covering rough sets. Georgian Electronic Scientific Journal: Computer Science and Telecommunications 1(24), 24–31 (2010)
22. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers (1991)
23. Pei, X., Wang, Y.: An approximate approach to attribute reduction. International Journal of Information Technology 12(4), 128–135 (2006)
24. Rawski, M., Borowik, G., Łuba, T., Tomaszewicz, P., Falkowski, B.J.: Logic synthesis strategy for FPGAs with embedded memory blocks. Electrical Review 86(11a), 94–101 (2010)
25. Selvaraj, H., Sapiecha, P., Rawski, M., Łuba, T.: Functional decomposition – the value and implication for both neural networks and digital designing. International Journal of Computational Intelligence and Applications 6(1), 123–138 (2006), doi:10.1142/S1469026806001782
26. Skowron, A., Rauszer, C.: The discernibility matrices and functions in information systems. In: Słowiński, R. (ed.) Intelligent Decision Support – Handbook of Application and Advances of the Rough Sets Theory. Kluwer Academic Publishers (1992)
27. Słowiński, R., Sharif, E.: Rough sets analysis of experience in surgical practice. In: Rough Sets: State of The Art and Perspectives, Poznań-Kiekrz (1992)
28. Wang, C., Ou, F.: An attribute reduction algorithm based on conditional entropy and frequency of attributes. In: Proceedings of the 2008 International Conference on Intelligent Computation Technology and Automation, ICICTA 2008, vol. 1, pp. 752–756. IEEE Computer Society, Washington, DC (2008), doi:10.1109/ICICTA.2008.95

29. Yao, Y., Zhao, Y.: Attribute reduction in decision-theoretic rough set models. Information Sciences 178(17), 3356–3373 (2008), doi:10.1016/j.ins.2008.05.010
30. ROSE2 – Rough Sets Data Explorer, http://idss.cs.put.poznan.pl/site/rose.html
31. ROSETTA – A Rough Set Toolkit for Analysis of Data, http://www.lcb.uu.se/tools/rosetta/
32. RSES – Rough Set Exploration System, http://logic.mimuw.edu.pl/~rses/