Hypercube Graph Decomposition for Boolean Simplification: An Optimization of Business Process Verification

Mohamed NAOUM¹, Outman EL HICHAMI², Mohammed AL ACHHAB³, Badr eddine EL MOHAJR⁴
New Technology Trends Team, Science and Technology Center for Doctoral Studies, Abdelmalek Essaâdi University, Tetouan, Morocco

Abstract—This paper deals with the optimization of business processes (BP) verification by simplifying their equivalent algebraic expressions. Actual approaches of business processes verification use formal methods such as automated theorem proving and model checking to verify the accuracy of the business process design. Those processes are abstracted to mathematical models in order to make the verification task possible. However, the structure of those mathematical models is usually a Boolean expression of the business process variables and gateways. Thus leading to a combinatorial explosion when the number of literals is above a certain threshold. This work aims at optimizing the verification task by managing the problem size. A novel algorithm of Boolean simplification is proposed. It uses hypercube graph decomposition to find the minimal equivalent formula of a business process model given in its disjunctive normal form (DNF). Moreover, the optimization method is totally automated and can be applied to any business process having the same formula due to the independence of the Boolean simplification rules from the studied processes. This new approach has been numerically validated by comparing its performance against the state of the art method Quine-McCluskey (QM) through the optimization of several processes with various types of branching.

Keywords—Business process verification; minimal disjunctive normal form; Boolean reduction; hypercube graph; Karnaugh map; Quine-McCluskey

I. INTRODUCTION

Business processes are key assets of any organization or information system [1], [2]. They are the communication interface and the medium of exchange between the organization stakeholders [3].

BP describe the core business and govern the operation of a system. Business Process Model and Notation (BPNN) is the wide used standard for modeling BP in view of its simplicity and usability [4], [5]. Nevertheless, BP may contain structural flaws [5] due to poor design or human errors. Hence, the verification task is a crucial step between the modeling and the execution phases of any BP. The complexity of real-life BP and the use of automated modeling tools often lead to complex models called “spaghetti” process models [6], [7] where manual verification is difficult to perform [8]. Therefore, automated formal methods are used instead. Automatic verification includes: Model Checking (MC) [5], [9] and Automated Theorem Proving (ATP) [10], [11].

The MC approach uses software called model checker to exhaustively check whether an abstraction equivalent structure of the BP satisfies some properties expressed in temporal logics. Simple Promela Interpreter (SPIN) is a widely used model checker that verifies if a model written in a C-like modeling language called Process Meta Language(Promela), meets properties expressed as Linear Temporal Logic (LTL) formulas [12], [13], [14]. Although this method has the advantage of indicating the counter example violating the checked property, it suffers from the state explosion problem [12] since its complexity is too high and the number of states grows exponentially.

The ATP (or automated deduction) is a subfield of mathematical logic dealing with automatic (or semi-automatic) proving of mathematical theorems. The computer programs allowing this task are called theorem provers [15].

First-order theorem proving is one of the most mature subfields of ATP thanks to its expressivity that allows the specification of arbitrary problems [16]. However, some statements are undecidable [17] in the theory used to describe the model. thereby, current research [18], [17], [19] deal with the challenge of finding subclasses of first-order logic(FOL) that are suitable and decidable in the mapping of such models.

Higher order logics are more expressive and can map wider range of problems than FOL, but theorem proving for these logics is not as developed as in the FOL[20].

Regardless the used approach to verify a BP, its logical structure is deducted as a propositional logic formula written in Disjunctive Normal Form (DNF) [2], [7]. The DNF can be reduced to a minimal form in order for the manipulation and practical implementation to become more efficient. Thus, an optimization of the PB verification is achieved.

Since the simplification of Boolean expressions is extensively used in the analysis and design of algorithms and logical circuits, several methods were developed to perform this task:

- The algebraic manipulation of the Boolean expressions aims at finding an equivalent expression by applying the laws of Boolean algebra. However, for such methods, there is no fixed algorithm to be used to minimize a given expression. Thus, choosing which
This article introduces a novel technique to optimize the verification of a BP by simplifying its equivalent logical formula written in the Disjunctive Normal Form (DNF). This new simplification algorithm searches for the largest hypercubes of lower dimensions (called elements) that are enough to cover all vertices in a partial cube graph mapping of the BP. A minimal equivalent DNF is then expressed as a disjunction of the necessary hypercube abstractions in this elements coverage.

The rest of this paper is structured as follows: Section II describes how the BP is modeled in BPMN. Section III presents the main Boolean algebra simplification rules as well as the hypercube properties that are used in the developed algorithm. Section IV explains in details the simplification algorithm and goes throw the used speedup tweaks. Our findings are presented and discussed in Section V. Finally, a conclusion is given.

II. BUSINESS PROCESS MODELING AND NOTATION

The most used business process modeling standard is Business Process Model and Notation (BPMN). It is a specification of the Object Management Group (OMG) [25]. The modeling is done by interconnecting standard graphical symbols grouped in five categories:

The Swimlanes and Artifacts categories are used to group objects into lanes and to provide additional descriptions. The Data elements category is used to describe the flow of the data through the process.

The main role of the three categories above is to increase readability of the model without effecting its execution. Therefore the whole BP flow can be described with the remaining two categories: Flow Objects and Connecting Objects [25].

The BPMN 2.0 specifies three Flow Objects: 1) Events, 2) Activities and 3) Gateways (see Fig. 1). These elements are connected using Connecting Objects especially Sequence flows.

The Event elements indicate the various incidents that can occur during the process execution. Three main type of events can be distinguished according to their trigger time: 1) Start Events, 2) End Events, and 3) Intermediate Events. They indicate the beginning or the end of a process or simply any event that may arise in-between.

The Activity elements are used to indicate any performed task in a process. Depending on the level of abstraction, an Activity may be compound or atomic.

The Sequence flows are the arcs connecting related events and activities. They define the chronological order of the elements within a process. If the activation of a sequence flow depends on some condition, then a Boolean variable is defined above it. Thus the immediate successor element is activated only if this condition is considered to be true.

The Gateway elements are used to indicate any divergence or convergence in a Sequence Flow. Depending of their behavior, the five types of Gateways are: Exclusive, Inclusive, Parallel, Event-Based, and Complex. They determine the branching, forking, merging, and joining of paths.

The graph composed of Flow objects and their Sequence Flows connections describes the eventual executions of a BP. Each path of the graph going from the start to the end events indicates a single execution scenario. As an example, Fig. 2 shows a simplified payment/delivery BP.

Once the modeling of the BP is done, the designer must choose which verification method to apply. The structure of the BP model is then extracted as a mathematical expression that depends on the used gateways and the sequence flows branching. The next section will present the necessary elements.
used to map the logical structure of a BP and the main rules used to simplify its equivalent formula.

III. Binary Representation and Reduction Rules

A. Definitions

1) Boolean variable: A Boolean variable is a variable that takes only one of the logical values: either 1 (meaning True) or 0 (meaning False). The complement of a variable $A$ is denoted $\overline{A}$ and has the opposite value of $A$. A literal is either the logic variable $A$ or its complement $\overline{A}$.

2) Minterm: A Minterm is a product (conjunction) of all the variable literals. For instance, for three Boolean variables $A$, $B$, and $C$ the expressions $AB\overline{C}$, $A\overline{B}C$, and $A \land B \land C$ denote the same minterm. It means that $C$ has the value 0 and both $A$ and $B$ have the value 1. By assigning a power of 2 to each variable of a minterm $V_{n-1}...V_2V_1V_0$ composed of $n$ variables $V_i$, the shorthand notation is $m_d$ where $d$ denotes the decimal value of the binary expression $V_{n-1}...V_2V_1V_0$.

For example, $m_6$ is the short hand notation of $ABC$ because $110_2=6$.

3) Disjunctive Normal Form (DNF): A logical formula is considered to be in Disjunctive Normal Form (DNF) if and only if it is a disjunction (sum) of one or more conjunctions (products) of one or more literals [26]. A DNF formula is in full disjunctive normal form if each of its variables appears exactly once in every conjunction (minterm). The only propositional operators in DNF are and (denoted with $\land$, or $\lor$, or (denoted with $+$ or $\vee$), and not (denoted with $\neg A$ or $\overline{A}$). The not operator can only be used as part of a literal, which means that it can only precede a propositional variable. The following formula of three variables $A$, $B$, and $C$ is in DNF:

$$f = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \quad (1)$$

It can be written in shorthand notation as follow:

$$f = m_3 + m_4 + m_6 + m_7 \quad (2)$$

B. Boolean Algebra

1) Boolean algebra identities: In Boolean algebra, there are four basic identities for addition (logical $\lor$) and four for multiplication (logical $\land$) that holds true for all possible values of a Boolean statement variables. Table I gives a summary of those identities:

2) Boolean algebra properties: In Boolean algebra, there are three basic properties: commutative, associative, and distributive. Table II gives a summary of those properties:

### Table I. Boolean Algebraic Identities

<table>
<thead>
<tr>
<th>Operation</th>
<th>Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$A \lor 0 = A$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$A \land 0 = 0$</td>
</tr>
</tbody>
</table>

### Table II. Boolean Algebra Properties

<table>
<thead>
<tr>
<th>Operation</th>
<th>Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$A \lor B = B \lor A$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$A \land B = B \land A$</td>
</tr>
</tbody>
</table>

A. The Hypercube Graph Representation

A Boolean statement of $n$ variables can be written in DNF with at most $2^n$ minterms of $n$ literals. By creating a vertex for each minterm $m_i$ and linking each two vertices when their binary representations differ in a single digit (the Hamming distance of their minterms is one), a hypercube graph (noted $n$-cube or $Q_n$) is created[27]. Fig. 3 gives a flat representation of the hypercube graph $Q_4$.

A hypercube graph of $n$ vertices can be viewed as the disjoint union of two hypercubes $Q_{n-1}$ if an edge is added from each vertex/minterm in one copy of $Q_{n-1}$ to the corresponding minterm/vertex of the other copy. As shown in Fig. 4, the joining edges form a perfect matching between the blue and black vertices.
In fact, every hypercube \( Q_n \) of \( n > 0 \) is composed of elements, or \( n \)-cubes of a lower dimension, on the \((n-1)\)-dimensional surface on the parent hypercube. The smallest elements are the vertices (points). There is \( 2^n \) of them.

In general, the number of \( m \)-cubes on the boundary of a given \( n \)-cube is \( E_{m,n} = 2^n - m^m \) where \( \binom{n}{m} = \frac{n!}{m!(n-m)!} \) is the binomial coefficient.

A partial cube is an isometric subgraph of a hypercube. The distance between any two vertices in the subgraph is the same as the distance between those vertices in the hypercube.

**Lemma III.1** Let \( Q_n \) be a hypercube graph with \( n > 0 \) minterms \( m_i \) where \( i \in [0, 2^n] \). Let \( f \) be a DNF formula given by the disjunction of all \( Q_n \) minterms. Then \( n \) variables of \( f \) can be simplified. The abstracted equivalent formula is easily obtained by identifying the common literals between the minterm with maximum shorthand notation value (denoted \( m_{\text{max}} \)) and the one with the minimum shorthand notation value (denoted \( m_{\text{min}} \)). This abstraction is chosen to be called: abstraction \( m_{\text{max}} \) with filter \( m_{\text{min}} \).

**Proof:** For instance, if \( n = 1 \) then \( Q_1 \) is composed of two minterms \( m_0 \) and \( m_1 \) of one variable \( v_0 \). By applying the identity \( \overline{v_0} + \overline{v_0} = 1 \), an abstraction of the variable \( v_0 \) is given (abstraction \( m_1 \) with the filter \( m_0 \)).

If \( n = 2 \) then \( Q_2 \) is composed of four minterms \( \{m_0, m_1, m_2, m_3\} \) each one is composed of two variables \( v_0 \) and \( v_1 \). By applying the same identity to two opposite sides of \( Q_2 \) an abstraction of the variables \( v_0 \) and \( v_1 \) is given (the abstraction \( m_3 \) with the filter \( m_0 \)). In fact:

\[
f = m_0 + m_1 + m_2 + m_3 = \overline{v_0} \overline{v_1} \lor \overline{v_0} v_1 \lor v_0 \overline{v_1} \lor v_0 v_1 \lor v_0 v_1 \overline{v_0} \lor \overline{v_0} v_0 \overline{v_1} = \overline{v_0} \lor v_1 = 1
\]

Let us assume that the lemma III.1 is correct for any \( n > 0 \). Let \( Q_1 \) and \( Q_2 \) be two hypercubes that their disjoint union form the hypercube \( Q_{n+1} \). Each minterm \( m_x = m_x \lor \lor v_{n-1} \ldots v_0 v_1 v_0 \) in \( Q_1 \) forms a perfect matching with another minterm \( m_y = m_y \lor \lor v_{n-1} \ldots v_0 v_1 v_0 \) in \( Q_2 \). \( m_x \) and \( m_y \) can be abstracted to \( m_z \) because they differ by the value of a single variable \( v_n \). Each minterm:

\[
f = m_x + m_y = \lor v_{n-1} \ldots v_0 v_1 v_0 \lor v_{n-1} \ldots v_0 v_1 v_0
\]

which gives an abstraction of the variable \( v_n \). As a result, the hypercube \( Q_{n+1} \) gives an abstraction of \( n + 1 \) variables: \( n \) variables with the hypercube \( Q_1 \) plus that of \( v_n \).

In the next section, an explanation of how the lemma III.1 can be used as a key stone to perform the simplification of any formula written in DNF is given.

**IV. SIMPLIFICATION ALGORITHM**

In order to simplify a Boolean expression written in DNF, its expression is represented as a partial cube \( PQ_n \) of the hypercube graph \( Q_n \), with \( n \) the number of variables in the DNF formula. The developed algorithm consists in finding the largest elements (hypercubes) \( Q_m \), with \( m \leq n \), so that their disjoint union covers all vertices of the partial cube \( PQ_n \). The fewer is the number of necessary hypercubes \( Q_m \), the more abstract is the equivalent formula. As an example, the following DNF formula can be considered:

\[
f(A, B, C, D) = \overline{A} \overline{B} \overline{C} D + \overline{A} B C D + \overline{A} B \overline{C} D + A B \overline{C} D + A B C D + A B C D + AB \overline{C} D + A B \overline{C} D + A B C D + A B C D
\]

This formula is represented as a partial cube \( PQ_4 \) with vertices \( m_1, m_3, m_4, m_5, m_7, m_9, m_{11}, m_{12}, m_{13}, \) and \( m_{15} \). Fig. 5 shows that the vertices of \( PQ_4 \) (green and yellow vertices) can be covered with the disjoint union of two hypercubes \( Q_2 \) and \( Q_3 \).

Using lemma III.1, three variables \( A, B, \) and \( C \) can be reduced with the hypercube \( Q_3 \) composed of vertices \( \{m_1, m_3, m_5, m_7, m_9, m_{11}, m_{13}, m_{15}\} \). Thus \( Q_3 \) is reduced to \( m_{15} \) with the filter \( m_1 \) which is equivalent to the expression \( D \) since it is the only variable that remains with the same value in all minterms of \( Q_3 \) (we have \( m_{\text{max}} = m_{15} = m_{111111} \) and \( m_{\text{min}} = m_{1} = m_{000000} \) the abstraction is \( - - - 1 \) ).

The hypercube \( Q_2 \), composed of \( \{m_4, m_5, m_{12}, m_{13}\} \), gives an abstraction of two variables \( A \) and \( D \). Thus \( Q_2 \) is reduced to \( m_{13} \) with the filter \( m_4 \) which is equivalent to the expression \( \overline{B} C \) (we have \( m_{\text{max}} = m_{13} = m_{1110} \) and \( m_{\text{min}} = m_4 = m_{0100} \) the abstraction is \( 10 \) ).
Finally, the disjunction of this two abstractions gives the minimal formula:

$$(A, B, C, D) = D + B \overline{C}.$$  

If a vertex of the partial cube is covered by multiple hypercubes, the largest one has to be considered. That way, each vertex is surely covered with the most abstract expression.

A simplified version of the reduction algorithm is summarized in Fig. 6. The algorithm starts with identifying the vertices of the partial cube $PQ$ that maps all the minterms of the formula to minimize. Then, it tries to find, for each vertex $m_i$ of the $PQ$, the largest hypercube (or hypercubes if there are many with the same size) that contains $m_i$. Finally, the algorithm gives priority to external vertices then holds only the necessary hypercubes to cover them all. The abstraction given by those hypercubes is the minimal equivalent expression of the DNF to minimize.

In the next section, the performance of the proposed algorithm will be compared with the Quine-McCluskey method (QM).

All abstractions were performed using a Python implementation of the developed algorithm. They were then compared to the optimized Python implementation of the standard Quine-McCluskey algorithm (This implementation is included in the digital electronics simulation library BinPy).

The experiments were carried out on a conventional laptop computer equipped with an Intel i5 processor and 8GB of RAM.

For each dimension $n$, with $n \leq 4$, all formulas were tested since there is only 65812 possible ones ($2^{24} + 2^{24} + 2^{24} + 2^{24} = 65812$). For $n > 4$, the formulas to minimize were chosen up to $x = 2^{24}$.

For each test, the running times, for both methods, were recorded starting from the feeding of the formula to minimize until the reception of the minimal equivalent DNF. The integer representing the input formula is then incremented for the next test. Since the execution time can vary significantly depending on the input size, we choose to plot the relative percent difference of the two algorithms runtimes. Each scatter in Fig. 7 represents the result of one test that is given by the formula:

$$100 \times \frac{QM's\ runtime - Our\ algorithm's\ runtime}{minimum\ of\ both\ runtimes}$$

A blue scatter indicates a result in favor of the proposed algorithm while a red scatter indicates a result in favor of the QM algorithm.

The plot was generated using the python data visualization library Seaborn based on matplotlib.

V. RESULTS AND DISCUSSION

From Fig. 7 we can conclude that our algorithm has better performances than the QM Method since it has better results in 89.40% cases of the $2^{24}$ conducted tests. Moreover, the proposed algorithm is over 400% faster in 1380450 cases while the QM method is over 400% faster only in 746 cases. Also this percent difference can reach over 2000% in 3829 cases in favor of the developed algorithm and in no case in favor of the QM method.

One advantage of the new algorithm introduced in this work, is that it follows a top-down approach: it searches first for the largest hypercube that covers a minterm which means that the algorithm does not waste time on smaller hypercubes with less abstraction. In the counterpart, the Quine-McCluskey algorithm follows a down-top approach: it tries to find all prime implicants of size 2 then size 4 and so on, which means that it wastes time on multiple partial prime implicants before reaching the optimum formula.

A second advantage of the developed algorithm is that, unlike for the tabular method, there is no need to use the Pitrick’s method to solve the problem of cyclic covering. It is simply solved by holding first the coverage of the external vertices of the decomposition hypercubes.

Finally, another advantage is the use of binary operations that are directly supported by the microprocessor; it applies a simple binary and/or filters to find the successors of a given vertex or to store the previous found coverage. For instance, if there are six variables then there are $2^6 = 64$ minterms, instead of using a loop of 64 iterations, a single microprocessor operation can be used to filter the active minterms in the partial cube.

VI. CONCLUSION

Business Processes are indubitable tools for the modern business planning, but those models can include structural flaws that are hard to detect with manual verification, which gives extreme importance to automatic verification.
methods verification algorithms suffer from the high complexity since the problem they try to solve is NP-hard, hence the necessity to reduce the problem size by minimizing the number of literals.

In this paper, a novel technique of business processes simplification has been presented. A simplification tool that performs literals reduction using hypercube decomposition has been built. Moreover, the simplification algorithm was entirely automated which makes the optimization task accessible to the regular BP designers. Promising subject of research can be explored in further depth, such as how machine learning algorithms could be used to accelerate the simplification algorithm, how the algorithm can be modified to reduce the spatial complexity, and finally, the possibility of adapting the algorithm, view its characteristics, for quantum computing.

REFERENCES


