Decision tables and OBDDs

Yuen Man Hon

January 31, 2008

Technical University of Braunschweig
Institute of Railway System Engineering and Traffic Safety

This research project is supported by Karl-Vossloh-Stiftung
(Projectno. S047/10005/2006)
Preface

Decision tables are composed of conditions and actions. Rules are used to relate a possible combination of condition and action entries together for decision making. In the case of representing a propositional formula, the propositions of the formula are considered as the conditions. Each of the conditions can be evaluated to be true or false. The action of each rule is the evaluation of the formula.

One of the advantages of using decision tables as a specification method is that the requirements can be expressed in a compact form in a decision table by combining rules. If there are rules with only one difference in a condition and the combinations of other conditions are the same and they lead to the same action, then these rules can be combined together and form a more compact decision table. This is called the optimization of decision tables. An compact decision table summarizes the possible situations that the systems needs to react to and the corresponding actions. It has different kinds of properties. These properties are consistency, exclusivity, inclusiveness, completeness and overlapping.

In order to transform propositional formulas to decision tables, OBDD is used to represent a propositional formula, such that a compact decision table can be automatic generated from the OBDD and the semantics of the formula is persevered. OBDD is a compact representation of a propositional formula and the decision table of the propositional formula can be deduced from the OBDD. A generated decision table satisfies the properties of consistency, exclusivity, completeness and inclusiveness. Based on the properties of OBDD, these properties are guaranteed.

In the formal framework OOLH, users can specify their expected behavior of the system in formulas or decision tables. These expected behaviors can be verified against the specified requirements. Before the verification, the consistency and completeness of this decision table must be checked. This structure analysis and verification are achieved based on the manipulation of boolean functions. OBDDs provides well defined algorithms to support manipulation of boolean expression.

In this note, the decision tables and their properties are formally defined in the section 1. An introduction to OBDD is given in the section 3. The concepts of OBDD that are applied to transform a propositional formula to a decision table are discussed in the section 4. The idea in applying OBDDs to analyze decision tables is further elaborated in section 4. A conclusion is given in the end of this note. Examples are given in each section. This examples are written based on a case study of specifying railway system requirements. The main purpose of the examples is to illustrate the concepts, therefore, definitions of the railway concepts are not discussed in detail.
Contents

1 Decision Tables 1
   1.1 Definition of Decision tables 1
   1.2 Decision Tables of OOLH 4

2 Propositional Logic and Decision Tables 7
   2.1 Propositional formulas and Truth tables 7
   2.2 Propositional formulas and Decision tables 7

3 Introduction to OBDDs 12
   3.1 Definition of BDDs 12
   3.2 Definition of RBDDs 15
   3.3 Definition of OBDDs 17

4 Transformation of OBDDs and Decision tables 20
   4.1 Propositional formulas and OBDD 20
   4.2 OBDD and Decision tables 21

5 Analysis of Decision Tables and OBDDs 24
   5.1 Consistency of Decision tables 24
   5.2 Completeness of Decision tables 25
   5.3 Exclusivity and Inclusiveness 27
   5.4 Verification 28

6 Conclusion 31
List of Figures

1. Decision Table, evaluation of $ZPZ_{DWeg}$ ........................................ 2
2. Overlapping of rules in decision tables ............................................. 4
3. Consistency of rules in decision tables ........................................... 5
4. Exclusivity of rules in decision tables ............................................. 6
5. Inclusiveness of rules in decision tables ......................................... 6
6. OBDDs of $s_7$ .............................................................................. 10
7. Decision tables of $s_7$ .................................................................... 10
8. Transformation from proposition formula to decision table ............ 11
9. A directed graph ............................................................................ 12
10. BDD of $a \land b$, truth table and boolean function .................... 15
11. Reductions on BDD of $a \land b$ ..................................................... 16
12. Reductions on BDD of $\psi$ ............................................................ 17
13. BDD of $a \land b$ where a variable $a$ occurs more than once .......... 17
14. OBDD of $(b \land a) \lor (b \land c)$ ................................................... 22
15. OBDD of überlappen ................................................................. 23
16. Inconsistency and OBDDs ............................................................. 25
17. Completeness and OBDDs .............................................................. 26
18. $AOBDD_{exandin}$ and OBDDs ....................................................... 28
19. Two situations of a point ............................................................... 30
20. Decision table of situation (a) and (b) .......................................... 30
21. Resultant OBDDs of situation (a) and (b) .................................... 30
1 Decision Tables

As mentioned above, decision tables describe the system’s logic based on the conditions and the corresponding actions of the system. In this section, the type of decision table $D_{OOLH}$ that is used to express the propositional formulas of the infrastructure objects is first introduced informally and the formal definition of decision tables $D_{OOLH}$ is then given.

1.1 Definition of Decision tables

In this section, decision tables are discussed informally based on the following example. Table 1 is a decision table. The first column of the decision table is composed of two parts: the conditions of the system and the actions that are triggered based on the conditions. The first part is called condition stub and the second part is called action stub. The action stub contains the actions of the system. The condition stub contains all the possible conditions that will trigger the actions. In table 1, row 1 - 5 of the first column are the conditions of infrastructure elements and row 6 of the first column is the action. The action of this table is the evaluation of $Z_{PZ_{DW \text{eg}}}$ of infrastructure elements.

The other columns show the possible combinations of the condition entries and the corresponding action entry. These columns are called the rules of a decision table. In this example, the condition and action entries are specified by either $Y$ or $N$. They indicate Yes and No, respectively. The condition entry is described by the truth values. It is called a limited condition entry. The condition can be formulated as a yes-no question. For example, in table 1, the first condition is $\text{solllage}$, this can be formulated as "Is the point set to a proper position for the route?" If a condition is specified by different attributes, it is called extended condition entry. For example, a condition like "The aspect of a main signal" can be $\text{Hp0, ks1, ks2}$. There is another symbol in the condition entry ‘-’. This entry means do not care. This entry is called ‘Don’t-care’ entry. The action of this rule will not be affected by this condition. In other words, no matter whether this condition is evaluated as $Y$ or $N$, it will lead to the same result in the corresponding rule.

In table 1, the action entry contains $Y$ and $N$ which indicates Yes and No respectively. $Y$ means the action would be triggered based on the combination of conditions. If the corresponding action will not take place, $N$ is specified in the action entry. For example, in the second column (R1), if the point is set in a proper position, then $Z_{PZ_{DW \text{eg}}}$ of the point is positive. The third column (R2) indicates that the point is set in an improper position for the $D_{Weg_{\text{current}}}$ and it is blocked, then $Z_{PZ_{DW \text{eg}}}$ of the point is negative. Similar to the condition entry, if the action entry is specified by truth values, $Y$ or $N$, it is called limited action entry and if it is specified by different attributes, it is called extended action entry.

In propositional logic, the meaning of a proposition is given by the assignment of truth values. There are two truth values which are true and false and they are identical to $Y$ and $N$, respectively. As a result, decision tables $D_{OOLH}$ with limited condition and action entries are used to the semantics
of propositional formulas and DT_{OOLH} are defined formally as follows:

- **Definition 1** *Condition set*
  
  \[ C = \{C_1, C_2, \ldots, C_c\} \]

  The condition set indicates the set of condition symbols. In the railway domain, \( C \) contains all the possible attribute of infrastructure objects. For example, \( C_{ESTW-R} = \{\text{solllage, FWEL(b), DWEL(b), FLEL(b), überlappen, spitz}\} \).

- **Definition 2** *Condition domain*
  
  \[ CD = \{Y, N\} \]

  The condition domain \( CD \) indicates the possible values of the condition symbol. As mentioned above, propositions that are used to describe the properties of each infrastructure object has only two truth values. As a result, the condition domain \( CD \) has only one subset which has only two elements, \( Y \) (Yes) and \( N \) (No).

- **Definition 3** *Condition subject*
  
  \[ CS = \{CS_1, CS_2, \ldots, CS_k\} \]

  \( k \): number of conditions in a decision table and \( CS \subseteq C \)

  The condition subject indicates a set of condition symbols of a decision table. This means, the condition subject must be a subset of the condition set. In table 1, the condition subjects are solllage, gesperrt, EL(b), b and spitz. \( CS_{ZPZ_{Dwegcond2}} = \{\text{solllage, gesperrt, EL(b), b, spitz}\} \).

- **Definition 4** *Condition*
  
  \( C_i = (CS_i, \{Y, N\}) \)
A condition is an ordered pair. Each pair indicates the condition subject and its possible values. If the possible values of a condition subject are Y and N, then the condition is called limited condition entry. In table 1, the condition solllage is specified as an ordered pair \((\text{solllage}, \{Y, N\})\).

- **Definition 5** Condition space
  \[ \text{SPACE}(C) = \{Y, N\}^k \]
  Condition Space indicates the possible combination of the condition entries in a decision table. Since all condition entries are limited condition entry, the total number of possible combinations is \(|\text{SPACE}(C)| = 2^k\) without the introduction of ‘Don’t-care’ entries. For example, the total number of condition of decision table 1 is 5 and the total number of possible combinations is then 32.

- **Definition 6** Action set
  \[ A = \{A_1, A_2, \ldots, A_a\} \]
  The action set is a set of action symbols. It contains all the possible actions that could be triggered in the domain. For example, in the railway domain, \(A = \{ZPZ_{\text{positive}}, F\text{"U"MBlik}\}\).

- **Definition 7** Action domain
  \[ \text{AD} = \{\{Y, N\}\} \]
  Similar to the condition domain, it contains the possible values of each action symbol. There are only two possible values of the action symbol that are used in DT\(_{\text{OOLH}}\). They are Y (Yes) and N (No). They indicate whether a safe route can be built during each phase of the development process under the logic of OOLH.

- **Definition 8** Action subject
  \[ \text{AS} = \{\text{AS}_k\} \]
  \[ \text{AS}_k \in A \]
  The action subject is a set of action symbols in a decision table. From this definition, each DT\(_{\text{OOLH}}\) can only have one action symbol and it has been defined in the action set. In table 1, the action subject is \(ZPZ_{\text{Dwegcond2}}\). \(\text{AS}_{ZPZ_{\text{Dwegcond2}}} = \{ZPZ_{\text{Dwegcond2}}\}\).

- **Definition 9** Action
  \[ A_i = (\text{AS}_{m_i}, \{Y, N\}) \]
  The action is a set of ordered pair of action symbols and their possible values. If the possible values of an action subject are the truth values
Y and N, then the action is called limited action entry. In table 1, the Action is specified as an ordered pair \((ZP_{D_{\text{wedge}cond2}}, \{Y, N\})\).

- **Definition 10** *Action space*

  \[
  \text{SPACE}(A) = \{Y, N\}
  \]

  The action space indicates the possible pairs of actions in the decision table. Each \(\text{DT}_{\text{OOLH}}\) can be specified by one action. As a result, the number of elements in this set is two.

- **Definition 11** *Decision table as matrix*

  \[
  \text{DT} = (dt_j) = \left( \begin{array}{c}
  d_{ij} \\
  a_{ij}
  \end{array} \right)
  \]

  \(i \in \{1, ..., k\}\) and \(j \in \{1, ..., n\}\)

  \(k:\) number of conditions in the decision table

  \(n:\) number of rules in condition tables

  \(d_{ij} \subseteq \{Y, N\} \setminus \{\emptyset\}\)

  \(a_{ij} \in \{Y, N\}\)

  \(\text{DT}_{\text{OOLH}}\) is defined as a matrix. Informally speaking, \(dt_j\) represents a rule in a decision table. Each of the rules is composed of conditions and one action. There is one action which is defined in the action subject. As mentioned above, 'Don’t-care' entries can be used to combine rules together. 'Don’t-care’ entries is represented as '-' and defined as \(d_{ij} = '-'\) which corresponds to \(d_{ij} = \{Y, N\}\).

1.2 Decision Tables of OOLH

The properties of \(\text{DT}_{\text{OOLH}}\) has been given in the last section. The three requirements of \(\text{DT}_{\text{OOLH}}\) are discussed in this section. These three requirements are consistency \(\text{DT}_{\text{consistency}}\), exclusivity \(\text{DT}_{\text{exclusivity}}\) and inclusiveness \(\text{DT}_{\text{inclusiveness}}\).

Before defining these three requirements of decision tables, a concept called rule overlapping is first introduced. Rule overlapping means that the
intersection of all condition entries of two rules is not empty. Figure 2 shows the three possible rule overlappings of decision tables. In figure 2(a), R1 and R2 are the same and the intersection of the condition entries is (Y, Y, N). The intersection of the condition entries is (Y, Y, N) in figure 2(b). In figure 2(c), the intersection is (Y, Y, -). Overlapping DT$_{\text{overlap}}$ is formally defined as follows:

**Definition 12 DT$_{\text{overlap}}$**

\[
(\exists j, k \in \{1, \ldots, n\})(\bigwedge_{i=1}^{c} (d_{ij} \cap d_{ik} \neq \emptyset))
\]

If a decision table contains overlapping rules, two aspects need to be considered and analyzed. They are consistency and exclusivity. A decision table is called inconsistent if the action entries of any overlapping rules are different. In other words, same situations can lead to different actions based on the decision table. There exists a contradiction in the decision table. In figure 3 (a), although the intersection of two rules is not empty, these rules are consistent because the actions of these rules are the same. Figure 3 (b) and (c) have inconsistencies among rules because there are overlapping rules, but their actions are different. Consistency DT$_{\text{consistency}}$ of decision tables is defined as follows:

**Definition 13 DT$_{\text{consistency}}$**

\[
(\forall j, k \in \{1, \ldots, n\})(\bigwedge_{i=1}^{c} (d_{ij} \cap d_{ik} \neq \emptyset)) \rightarrow a_{1j} = a_{1k}
\]

If a decision table contains overlapping rules and the rules are consistent, exclusivity can be considered. In other words, there is no ambiguity in choosing rules to be applied with a given condition pair. For example, in figure 4(a), (b) and (c), it is not clear which rule should be applied with a given condition tuple (Y, Y, N) because of the overlapping. Figure 4(d) shows one of the possibilities in ensuring the exclusivity of rules of the decision table in
**Figure 4:** Exclusivity of rules in decision tables

**Figure 5:** Inclusiveness of rules in decision tables

Figure 4(c). Exclusivity $DT_{exclusivity}$ is formally defined as follows:

**Definition 14** $DT_{exclusivity}$

$$\forall j, k \in \{1, \ldots, n\}((\exists i \in \{1, \ldots, c\})(d_{ij} \cap d_{ik} = \emptyset))$$

Inclusiveness of a decision table means if two rules lead to the same action and contain only one difference in the condition entry, then these two rules can be combined and this condition entry of the rule must be represented by the union of the condition domain $Y,N$, in other words, the symbol ‘-’. In figure 5, the intersection of two rules has two elements, as a result, Inclusiveness $DT_{inclusiveness}$ is defined as follows:

**Definition 15** $DT_{inclusiveness}$

$$\forall j, k \in \{1, \ldots, n\}((\exists i \in \{1, \ldots, c\})(\forall t \in \{1, \ldots, c\}-\{i\})(d_{ij} \cap d_{ik} = \emptyset) \land (d_{tj} = d_{tk}) \land a_{1j} = a_{1k} \rightarrow d_{ij} = \{Y, N\})$$

Among the idea of rule overlapping and the requirements of $DT_{OOLH}$, there is another concept of decision tables that needs to be introduced. A $DT_{OOLH}$ is said to be completed, if the possible combination of condition entries is specified in the decision table.
2 Propositional Logic and Decision Tables

The semantic meaning of a propositional formula or the corresponding boolean function can be expressed in these tables. As it has been mentioned before, one can also formulate the expected behavior of the system as a decision table for verification. These specified decision tables need to be analyzed. For example, checking the existence of contradictions. Therefore, it is necessary to search for a suitable technique to transform a propositional formula to a decision table which satisfies the requirements in section 1.2 and support the analysis of decision tables. In this section, the relationship between propositional formulas, truth tables and decision tables is discussed in section 2.1 and 2.2. In section 2.2, some techniques for transforming a propositional formula to a decision table are discussed.

2.1 Propositional formulas and Truth tables

The semantics of a propositional formula is given by its possible interpretations \[HR04\]. An interpretation is assigned based on the defined semantics of logical connectives and the truth values of each proposition of the formula. Mathematically, this assignment can be expressed as a function that maps each interpretation of the propositions to the set of truth values as follows:

\[ f : \{1, 0\}^n \rightarrow \{1, 0\} \]

\( n \): number of propositions in the formula

This function is called a boolean function\[Joh93\]. Propositions of the formula are called variables in the boolean function. Two truth values true \( T \) and false \( F \) are identical to 1 and 0, respectively. They are interchangeable in this work. Boolean functions capture the semantic meaning of propositional formulas. This means, in order to represent the semantic meaning of a propositional formula, a proper structure that can represent Boolean functions is needed. Truth tables can be used to represent boolean functions. A truth table lists all elements of a Boolean relation. It contains all possible interpretations of the corresponding propositional formula (see Table 1).

2.2 Propositional formulas and Decision tables

Another form of representing a boolean function in this work are decision tables. These decision tables must satisfy the requirements that have been discussed in section 1.2. There are different techniques to guarantee these requirements.

First, the truth table can be transformed directly into a decision table \[Mon74\]. The structure transformation is listed in table 2. Through such a transformation, decision tables represent the corresponding boolean function.

\[^1\text{In a finite set, boolean expression is isomorphic to propositional logic, informally, this means propositional algebra and boolean expression can be describe as equivalent}\]
### Propositional Logic and Decision Tables

#### Table 1: Valuations of \( \neg b \rightarrow (\text{solllage} \lor (\neg \text{gesperrt} \land \neg \text{DW}(b))) \)

<table>
<thead>
<tr>
<th></th>
<th>( b )</th>
<th>solllage</th>
<th>gesperrt</th>
<th>DW(b)</th>
<th>( \neg b \rightarrow (\text{solllage} \lor (\neg \text{gesperrt} \land \neg \text{DW}(b))) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>9</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>12</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

#### Table 2: Structure of truth and decision tables

<table>
<thead>
<tr>
<th>Truth table</th>
<th>Decision table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposition</td>
<td>condition</td>
</tr>
<tr>
<td>Proposition formula</td>
<td>action</td>
</tr>
<tr>
<td>Interpretation(proposition)</td>
<td>Limited condition entry</td>
</tr>
<tr>
<td>Interpretation(formula)</td>
<td>Limited action entry</td>
</tr>
<tr>
<td>T</td>
<td>Y</td>
</tr>
<tr>
<td>F</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 2: Structure of truth and decision tables
and have two characteristics. The first characteristic is that the condition and action entries are limited condition and action entries. It is because the semantic assignments of propositional logic are truth values, true and false, which are the same as Yes and No. Furthermore, decision tables contain all the possible combinations of condition entries and the corresponding actions without introducing 'Don’t-care' entries. In this case, DT\textsubscript{consistency} and DT\textsubscript{exclusivity} can be guaranteed based on this transformation. However, DT\textsubscript{inclusiveness} cannot be satisfied. For example, (R1,R2) of table 3 can be combined. A condition with a 'Don’t-care' entry can be viewed as redundant in applying the rule. Users do not need to take this condition into account when the rule is checked. As a result, directly transformed decision tables need to be processed further in order to fulfil the property DT\textsubscript{inclusiveness}.

One of the possible ways in processing such decision tables is comparing each pair of rules that can be consolidated. One of the drawbacks is that the order and way in consolidating the rules affects the size of the decision tables and a way in handling this problem needs to be designed. In the paper [K.75], Quine-McCluskey algorithm [Jr.59] is suggested to apply for optimizing a decision table. In other words, processing a decision table, such that DT\textsubscript{inclusiveness} can be guaranteed. As mentioned above, a decision table can be defined by a boolean function. The minimum set of minterms called prime implicants that is needed to represent the boolean function is found by Quine-McCluskey algorithm. In a decision table, this means that the minimum set of rules that can be used to represent the same set of actions is found. A Boolean function is the representation of a truth table, as a result, one can also apply this algorithm to obtain a compact decision table. However, the resulting decision table by applying this algorithm does not satisfy DT\textsubscript{exclusivity}. It is because a minterm can be used more than once to minimize another minterm. However, in decision table, if one rule is used to consolidate another rule more than once, the rules will overlap. The complexity of both methods is exponential and do not support analysis and manipulation of decision tables. They are therefore not suitable as the technique for transforming propositional formulas to decision tables. It must be addressed that finding the smallest size of decision tables, or in other words, the most optimized decision table, is an NP-hard problem [ZB99]. The focus of this work is finding out an existing and well-defined technique that considers this problem and generates an OOLH decision table automatically. This technique can also provide methods or operations in analyzing decision tables.
OBDDs (Ordered Binary Decision Diagrams) are one type of these structures [Bry86] (see section 3). An OBDD with the best variable ordering is a compact form in expressing a boolean function. OBDDs are based on the concept of binary decision trees and reduction rules are applied to the trees to obtain a compact representation of boolean functions. There are different advantages in applying this technique for transformation. First, a decision table that is generated based on the definition of OBDD fulfills DT\textit{exclusivity} and DT\textit{consistency}. For example, if a formula or a decision table is contradictory, only zero terminal will be generated in an OBDD. Secondly, with the suitable variable ordering of the boolean function, it is guaranteed that the transformed decision table fulfills the requirement DT\textit{inclusiveness}. Finally, decision tables can be analyzed by applying the defined operations of OBDDs. For example, structural analysis of decision tables. This means checking whether a decision table is complete and fulfills DT\textit{consistency}. Some research work on structural analysis has been done, however, the applied methods and techniques have not been formally defined [HZ95].

One of the drawbacks of this technique is finding the suitable or best variable ordering of a boolean function. This ordering affects the size of an OBDD representing a boolean function and the corresponding decision table [BW96] and [ZB99]. In figure 6(a), the ordering of the variables is \((b,\text{solllage},\text{gesperrt},\text{DW}(b))\), while in figure 6(b), it is \((\text{solllage},\text{gesperrt},\text{DW}(b),b)\). The first OBDD is smaller than the latter one. This implies, the number of rules in the decision table will be affected (see figure 7). The focus of this work is to investigate a generation of a suitable ordering for a
Figure 8: Transformation from proposition formula to decision table

Figure 8: Transformation from proposition formula to decision table

Figure 8: Transformation from proposition formula to decision table

Figure 8: Transformation from proposition formula to decision table

boolean function, such that a DT_{OOLH} can be generated, trying to minimize the size of the decision table with defined operations. Figure 8 describes the activities and the decisions that have to be made during the transformation of a propositional formula to a decision table with OBDD. This algorithm is called AOBDD_{transformation}.
3 Introduction to OBDDs

The advantages of using OBDDs as the transformation technique in this work has been briefly discussed in the above section. In this section, an introduction to OBDDs is first given. OBDDs are defined based on three different concepts. These concepts are Binary Decision Diagram (BDD), Reduced Binary Decision Diagram (RBDD) and variable orderings. The first concept is discussed in section 3.1. An introduction to RBDDs is given in 3.2. The concept of variable orderings and OBDDs is elaborated in section 3.3.

3.1 Definition of BDDs

A boolean function can be represented by a truth table. A truth table can be viewed as a decision table without the properties of DT\textit{inclusiveness}. This means the directly transformed decision table from a truth table contains redundant rules (see section 2.2). Algorithms need to be developed to reduce these redundant rules by combining rules together. Similarly, a boolean function can also be represented by a BDD (see figure 10). If this binary decision diagram is directly transformed as a decision table, then this decision table fulfills DT\textit{consistency} and DT\textit{exclusiveness}. However, it does not have the properties DT\textit{inclusiveness}. Before giving the formal definition of a BDD, some concepts and notations must be defined.

![Figure 9: A directed graph](image)

- **Definition 16** Directed graphs
  \[ G = (V, E) \]

  \( V \) is a finite set of vertices and \( E \) is a finite set of edges

  A directed graph is defined by a finite set of vertices and a finite set of edges. In figure 9, there are four vertices and four edges. It can be defined as a directed graph \( G \) as follows: \( G = (V, E) \), \( V = \{v_1, v_2, v_3, v_4\} \) and \( E = \{e_1, e_2, e_3, e_4, e_5\} \).

- **Definition 17** Initial node and terminal node of an edge \( e \)
\[ \text{initial} : E \rightarrow V \]
\[ \text{terminal} : E \rightarrow V \]

An edge can be considered as a connection between two vertices. One of the vertices is the beginning of a connection, called the initial node of an edge \( e \). The end of the connection is called the terminal node of the edge \( e \). The function initial is defined to map the edge to its initial node and terminal maps the edge to its terminal node. In figure 9, if \( \text{init}(e_1) = v_1 \) and \( \text{terminal}(e_1) = v_2 \), then \( e_1 \) is the edge that connects the vertices \( v_1 \) and \( v_2 \).

- **Definition 18** A Path in a graph is a sequence of edges \( e_1, e_2, ..., e_n \)
  \((\forall i \in \{1, ..., n - 1\})(\text{terminal}(e_i) = \text{initial}(e_{i+1}))\)
  \[ e_i \in E \]

A path is composed of at least one edge and two vertices. The edges of a path have the relationship that the terminal node of an edge is the initial node of the following edge. In figure 9, a path from the vertices \( v_2 \) to \( v_4 \) is \((e_3, e_5)\). The sequence \( e_3, e_5 \) forms a path because \( \text{initial}(e_5) = \text{terminal}(e_3) \).

- **Definition 19** A cycle in a graph is a path \( e_1, e_2, ..., e_n \)
  \[ \text{terminal}(e_n) = \text{initial}(e_1) \]

As a cycle, the terminal node of the last edge of a path is the initial node of the first edge of a path. In figure 9, the edges \( e_3 \) and \( e_4 \) form a cycle. It is because \( \text{initial}(e_4) = \text{terminal}(e_3) \). This sequence also forms a cycle because \( \text{terminal}(e_4) = \text{initial}(e_3) \). A graph without a cycle is called an acyclic graph.

- **Definition 20** \( v_0 \in V \) is an initial node of a graph
  , iff
  \[ (\forall e \in E)(\text{terminal}(e) \neq v_0) \]

An initial node \( v_0 \) of a graph is a vertex that does not have an incoming edge. In other words, the terminal node of any edge cannot be the initial node of the graph. In figure 9, \( v_1 \) is the only initial node.

- **Definition 21** \( v \in V \) is a terminal node of a graph
  , iff
  \[ (\forall e \in E)(\text{initial}(e) \neq v) \]

An initial node \( v_0 \) of a graph is a vertex that does not have an incoming edge. In other words, the terminal node of any edge cannot be the initial node of the graph. In figure 9, \( v_1 \) is the only initial node.
$V_t$ is the set of the terminal nodes of a graph. A terminal node of a graph is a vertex that does not have an outgoing edge. In other words, the initial node of any edge of the graph cannot be the terminal node. The set of terminal nodes of figure 9 is $\{v_4\}$.

- **Definition 22** $v \in V$ is a non-terminal node of a graph, iff

  $$(\exists e \in E)(\text{initial}(e) = v)$$

A non-terminal node of a graph is a vertex that must have an outgoing edge. In other words, if a vertex $v$ is a non-terminal node, there must exist an edge whose initial node is $v$. The set of non-terminal nodes of figure 9 is $\{v_1, v_2, v_3\}$.

**Definition 23**

A BDD is formally defined as a directed acyclic graph with:

- a unique initial node $v_{\text{initial}}$
- non-terminal nodes are labeled with a function $\text{variable} : (V - V_t) \rightarrow L$ where L is a set of variables of a boolean function. Each non-terminal node has exactly two children that are assigned by two functions $\text{low} : (V - V_t) \rightarrow V$ and $\text{high} : (V - V_t) \rightarrow V$ and the properties:

  $$((\forall v \in (V - V_t))(\exists e_1, e_2 \in E)((\text{initial}(e_1) = \text{initial}(e_2) = v) \land (\text{terminal}(e_1) = \text{low}(v)) \land (\text{terminal}(e_2) = \text{high}(v)) \land (\text{edge}(e_1) = 0) \land (\text{edge}(e_2) = 1))$$

  $$((\forall v \in (V - V_t))(|\{e \in E|\text{initial}(e) = v\}| = 2)$$

  where $\text{edge} : E \rightarrow \{0, 1\}$
- terminal nodes are labeled with a function $\text{value} : V_t \rightarrow \{0, 1\}$

The usage of graphical symbols of BDD is based on the paper [Bry86]. Figure 10 shows a BDD representation of the formula $a \land b$. Terminal nodes and non-terminal nodes are represented by squares and circles, respectively. The initial node of the diagram is always drawn at the top. Edges labeled with 0 are represented by dashed lines, while edges labeled with 1 are drawn as solid lines. Furthermore, the direction of the edges will not be indicated in the graph. Normally, the orientation of an edge is defined from top to bottom. The BDD in figure 10 is formally defined as follows:

$\begin{align*}
V &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\} \\
E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\
L &= \{a, b\} \\
V_t &= \{v_4, v_5, v_6, v_7\}
\end{align*}$
Introduction to OBDDs

The vertices are defined without labeling in a directed graph. In BDDs, the non-terminal and terminal nodes are labeled with the variables of the boolean function and values 0 or 1, respectively. Terminal nodes are the function values of a boolean function. Each of the non-terminal nodes has exactly two children. They are the low child and the high child. They can be reached by traversing via the dashed line and solid line, respectively. A BDD can be used to represent a boolean function. In figure 10, the path from $v_1$ to $v_4$ represents $(1,1,1)$ or the interpretation $(T,T,T)$ in the truth table of this formula. One can read this path as if the variable $a$ is assigned by the truth value $T$ (1) and the variable is assigned by the truth value $T$ (1), then the truth value of this formula is $T$ (1).

3.2 Definition of RBDDs

In figure 10, the BDD has a redundancy. It can be also viewed as a redundancy in a decision process. For example, when the value zero is assigned to the variable $a$, the function value is 0. The function value is independent of the assignment of the variable $b$. The size of this BDD can be reduced if this redundancy is removed. A BDD is a RBDD if the redundances in the BDD are eliminated based on three requirements of a RBDD [Bry86], [HR04] and [And98]. These three requirements remove the unnecessary decision points in a BDD and reduce the size of a BDD. It is the same as consolidating rules in a decision table. These three requirements are defined as follows:

Definition 24
1. There exists only two terminal nodes labeled with 0 and 1. 
   \(|V_t| = 2\) and \(value\) is a bijective function.

2. No non-terminal node has the same children
   \((\forall v \in (V - V_t))(\neg(low(v) = high(v)))\)

3. No non-terminal nodes with the same variable labeling have the same children
   \((\forall v_1, v_2 \in (V - V_t))(variable(v_1) = variable(v_2) \land \neg(v_1 = v_2) \rightarrow \\
   \neg(low(v_1) = low(v_2) \land high(v_1) = high(v_2)))\)

In a BDD, the non-terminal nodes can only be labelled 0 or 1, the size of a BDD can be reduced by sharing the non-terminal nodes. This idea leads to the first requirement of a RBDD. The second requirement does not allow the two outgoing edges of a node \(v_i\) point to the same node \(v_j\). This means, the evaluation of the boolean function does not depend on the assignment of \(v_i\). In this case, the vertex \(v_i\) will be removed and the incoming edges of \(v_i\) will redirected to \(v_j\). In figure 11(a), one of the vertices with a label b points to the same terminal node 0. This vertex is then removed and its incoming edge is redirected to the terminal node 0 in figure 11(b). The third requirement states that if there exist two nodes \(v_i\) and \(v_j\) with the same variable labeling sharing the same children or subgraph, then one of the nodes, say \(v_j\) and its outgoing edges must be removed. The incoming edges of \(v_j\) are then redirected to \(v_i\). In figure 12(a), the vertices \(v_2\) and \(v_3\) do not fulfil the last requirement of RBDD because \(variable(v_2) = variable(v_3)\), \(low(v_2) = low(v_3)\) and \(high(v_2) = high(v_3)\). As a result, one of the nodes must be removed. In 12(b), \(v_3\) is removed. However, this graph is not an RBDD because \(low(v_1) = high(v_1)\). Based on the second requirement, \(v_1\) is redundant and therefore it must be eliminated. Figure 12(c) is the valid RBDD.

![Figure 11: Reductions on BDD of \(a \land b\)](image-url)
3.3 Definition of OBDDs

Figure 13 shows a valid RBDD, it satisfies the three requirements of RBDD. However, the path \(v_1, v_2, v_3\) and \(v_4\) is redundant because the sub-path from \(v_3\) to \(v_4\) will never be taken because \(\text{variable}(v_1) = \text{variable}(v_3)\) and \(\text{edge}(e_1) = 1\). This means the variable \(a\) has been assigned with the value 1 and cannot be assigned with a new value. The latter assignment will not be considered.

Based on this concept, variable ordering is introduced to RBDD. A pair of RBDD and a variable ordering is called Ordered Reduced Binary Decision Diagram (OBDD) if:

**Definition 25**

\[
(\forall e \in E)(\text{terminal}(e) \notin V_t \rightarrow (\text{order}(\text{variable}(\text{initial}(e)))) < \text{order}(\text{variable}(\text{terminal}(e))))
\]

\(\text{order} : L \rightarrow \{0, 1, \ldots n\}\) and \(\text{order}\) is a bijective function.

In this work, if \(\text{order}(\text{variable}(v_i)) < \text{order}(\text{variable}(v_j))\), then \(\text{variable}(v_i)\) is said to be ordered higher than \(\text{variable}(v_j)\). A path from the initial node to the last non-terminal node of an OBDD represents a subset of the domain of the boolean function. The terminal node of this path represents the corresponding projection in the co-domain. An OBDD \(\text{obdd}\) is used to evaluate
the corresponding boolean function \( f \) and transform : \( L \rightarrow \{1, 0\} \) in the following way:

**Algorithm 1** \( AOBDD_{\text{evaluation}} \)

1. Start at \( v_{\text{initial}} \) of \( \text{obdd} \)
2. while (current vertex \( v \notin V_t \))
   \{ if \( \text{transform} (\text{variable}(v)) = 0 \) \( v = \text{low}(v) \) else \( v = \text{high}(v) \) \}
3. if \( \text{value}(v) = 0 \) \( f(x_1, x_2, ..., x_n) = 0 \) else \( f(x_1, x_2, ..., x_n) = 1 \)

OBDD is the canonical representation of a boolean function [Bry86]. This property brings advantages in manipulating OBDDs and analyzing boolean functions. First, the redundant variables of a boolean function can be found by establishing the corresponding OBDD. Checking the equivalence of two boolean functions can be achieved by comparing the structure of the corresponding OBDDs. These OBDDs must have a compatible variable ordering. Furthermore, the validity of a propositional formula can be tested by establishing the corresponding OBDD. If this OBDD has only the node \( v \), \( v \in V_t \) and \( \text{value}(v) = 0 \), then the propositional formula is not valid. Finally, there is a set of defined boolean operations that can be applied to manipulate OBDDs. These boolean operations include \( \overline{f} \), \( f + g \) and \( f \cdot g \).

The size of an OBDD in representing a boolean function depends on the variable ordering. In other words, OBDDs that are built based on the same boolean function with different variable orderings have different sizes. When an OBDD is used to represent a decision table, the number of rules is also affected by the size of the OBDD (see figure 6 and 7). As a result, finding the variable orderings for the boolean function is an important issue in using OBDDs. Static and dynamic approaches have been developed in finding the optimal ordering [BM02]. Strategies are designed to generate the ordering of an OBDD based on information of the specific application area in the static approach [FFN88]. Instead of using specific information, the variable ordering of a built OBDD is changed progressively in dynamic approaches. One of these strategies is the sifting algorithm. The main concept of this approach is swapping adjacent variables locally to search for the best ordering. A variable is chosen to exchange the position with its adjacent variable upwardly and downwardly w.r.t the OBDD, the minimum size of the OBDD and the corresponding position is then recorded. If the size of the current order is bigger than the expected, the process of swapping terminates. The algorithm continues to swap the next variable until all the variables have been used for estimation [Rud93] and [The97].

It is an NP-complete problem to find the best ordering to obtain the most compact OBDD [BW96]. Furthermore, finding the smallest size of decision tables is an NP-hard problem [ZB99]. As mentioned above, the focus of this work is to try minimizing the number of rules of a decision table. This decision table must satisfy the requirements of \( DT_{\text{OOLH}} \), \( DT_{\text{consistency}} \) and \( DT_{\text{exclusiveness}} \) and \( DT_{\text{inclusiveness}} \). As a result, another focus is put on
obtaining an initial variable ordering for a given formula and evaluating an OBDD w.r.t the requirements of $DT_{OOLH}$. These focuses are discussed in the next section.
4 Transformation of OBDDs and Decision tables

In figure 8, the process of transforming a propositional formula to a decision table is generally described. In this section, the idea is further elaborated. The initialization of the variable ordering and the evaluation of an OBDD w.r.t the requirements of DT TOOL are discussed in section 4.1. The idea of mapping from an OBDD to a decision table is considered in section 4.2.

4.1 Propositional formulas and OBDD

Algorithm 2 AOBDD transformation

1. ordering = initial(formula)
2. do{
   obdd = build(obdd, ordering)
   ordering = checkInclusiveness(obdd)
} until(ordering $\neq \emptyset$)
3. DT = transformToDT(obdd)

The first step of \textit{AOBDD transformation} is to initialize the variable ordering of the OBDD. In order to obtain the initial variable ordering to build the OBDD, the propositional formula is first transformed into Disjunctive Normal Form (DNF) for the static analysis. As mentioned in section ??, a DNF is a propositional formula and it is a disjunction of conjunctive clauses. A DNF of a propositional formula is equivalent to the sum of product of the corresponding equivalent boolean function (see table 5). The variable ordering of a boolean function $f(x_1, x_2, ..., x_n)$ w.r.t the DNF $c_1 \lor c_2 \ldots \lor c_m$ is generated with heuristic rules as follows:

1. a variable in a clause $c_n$ with less variables will have a higher position in the ordering.
2. the variables in the same clause $c_n$ should be as close to each other in the variable ordering as possible. This means that the variables in the same clause form a sequence in the ordering.
3. a variable with a high occurrence in all clauses will have a higher position in the ordering.

If a clause of a formula in DNF is evaluated to be true, then the evaluation of this formula is true. As a result, if a clause depends on a single proposition, then the formula can be evaluated immediately. In an OBDD, one of the edges of the vertex labeling with this variable leads immediately to one of the terminal nodes. The size of the OBDD can be reduced. This idea is elaborated in the first rule. The second rule states that closely related variables should be ordered as close as possible. If the truth value assignments of
variables in the same clause are known, the formula can be evaluated immediately. Consider for example, \((x_1 \land x_2) \lor (x_3 \land x_4)\). Instead of ordering the variables as \((x_1, x_3, x_3, x_4)\), one of the proper orderings is \((x_1, x_2, x_3, x_4)\) because the pair \(x_1, x_2\) is closer together in the formula. The same principle applies to the pair \(x_3, x_4\). The transformation of the propositional formula to a DNF and the static analysis are implemented in the method `initial(formula)`.

In step 2 of \(AOBDD\) transformation, the initial variable ordering is used to build the OBDD via the BDD package in the method `build(obdd, ordering)` [Dre02]. In order to ensure that the resulting OBDD satisfies the three requirements of \(DT_{OOLH}\), this result is then checked in the method `checkInclusiveness(obdd)` based on the following requirement:

1. No non-terminal nodes with the same variable labeling have the same low child or high child

\[
(\forall v_1, v_2, v_3 \in (V - V_t))((\text{variable}(v_1) = \text{variable}(v_2)) \land (\text{low}(v_1) = \text{low}(v_2) \lor \text{high}(v_1) = \text{high}(v_2))) \rightarrow \neg((\text{low}(v_3) = v_1 \land \text{high}(v_3) = v_2) \lor (\text{low}(v_3) = v_2 \land \text{high}(v_3) = v_1))
\]

Based on this requirement, if a pair of non-terminal nodes with the same labeling points to the same node by edges with the same labeling, then they are not allowed to be the children of any non-terminal node at the same time. If there exits such non-terminal node (e.g \(v_3\) in figure 14(a) and (b)), then it will lead to the same decision result via this pair of non-terminal nodes (e.g \(v_1\) and \(v_2\)) independent of the boolean value assignment of the parent. In an OBDD, redundancy can only be removed if the parent node points to the pair of non-terminal nodes has the same structure \((\text{low}(v_1) = \text{low}(v_2) \land \text{high}(v_1) = \text{high}(v_2))\). Here, this pair of non-terminal has either the same low child or high child. In order to remove this redundancy, the variable of the parent node \((\text{variable}(v_3))\) and the variable of the pair of non-terminal nodes \((\text{variable}(v_1) \text{ or variable}(v_2))\) are exchanged in the variable ordering of the OBDD. As a result, the parent node will be removed based on the definition of OBDD. For example, in figure 14(a), the vertices \(v_2\) and \(v_3\) have the same variable labeling and low child. The outgoing edges of both vertices with a label 0 point to the same terminal node and they are the children of \(v_1\). As a result, this OBDD does not satisfy the requirement. In figure 14(b), the OBDD does not fulfill the requirement because \(\text{variable}(v_2) = \text{variable}(v_3) \land \text{low}(v_2) = \text{low}(v_3)\), \(\text{low}(v_1) = v_3\) and \(\text{high}(v_1) = v_2\). As a result, the variable \(b\) and variable \(b\) are swapped in the ordering. The result of the OBDD with the changed variable ordering is shown in figure 14(c).

### 4.2 OBDD and Decision Tables

The algorithm for evaluating a boolean function with the corresponding OBDD has been discussed in section 3.3 (see Algorithm 1). A path of an OBDD represents the relation between the assignment of the boolean variables and the function value. In other words, each path of an OBDD is a representation of a rule of the decision table (see table 5). Given a path
Transformation of OBDDs and Decision Tables

Table 4: Equivalent structural relationships of propositional formulas and boolean functions

<table>
<thead>
<tr>
<th>Propositional formula</th>
<th>Boolean function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposition (φ, ψ)</td>
<td>boolean variable (a, b)</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>¬φ</td>
<td>(\overline{a})</td>
</tr>
<tr>
<td>φ ∧ ψ</td>
<td>a · b</td>
</tr>
<tr>
<td>φ ∨ ψ</td>
<td>a + b</td>
</tr>
<tr>
<td>φ → ψ</td>
<td>(\overline{a} + b)</td>
</tr>
</tbody>
</table>

Table 4: Equivalent structural relationships of propositional formulas and boolean functions

![Figure 14: OBDD of \((b \land a) \lor (b \land c)\)](image)

As it has been mentioned above, the decision table this is obtained from an OBDD satisfies the requirements of DToolH. If the transformed decision table does not contain any overlapping rules, then DT\text{consistency} and
Table 5: Transformation of OBDDs and decision tables

<table>
<thead>
<tr>
<th>OBDD Path</th>
<th>Decision table rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L )</td>
<td>( CS )</td>
</tr>
<tr>
<td>( \text{edge} (e) = 0 \land \text{initial} (e) = CS_i )</td>
<td>( d_{ij} = N )</td>
</tr>
<tr>
<td>( \text{edge} (e) = 1 \land \text{initial} (e) = CS_i )</td>
<td>( d_{ij} = Y )</td>
</tr>
<tr>
<td>( \text{value} (v) = 0 \land v \in (V - V_t) )</td>
<td>( a_{ij} = N )</td>
</tr>
<tr>
<td>( \text{value} (v) = 1 \land v \in (V - V_t) )</td>
<td>( a_{ij} = Y )</td>
</tr>
<tr>
<td>( \phi \rightarrow \psi )</td>
<td>( \pi + b )</td>
</tr>
</tbody>
</table>

Table 6: An decision table

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DWEL (b)</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>solllage</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>bHSS</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>z</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>überlappen</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

DT\textsubscript{exclusiveness} are satisfied. As it has been defined in section 1, in a decision table, if an intersection of a condition entry of any two rules is not empty, then this decision table does not contain any overlapping rules. Based on the mapping of OBDDs and \( DT_{OOLH} \), no overlapping rules will be produced because of the property of OBDDs. First, duplicated variables along any path of OBDDs are not allowed. A path of an OBDD is mapped to a rule of the decision table, the last node of the path is a non-terminal vertex. A non-terminal vertex of an OBDD (corresponding to a condition) must have two out-going edges \( e_1 \) and \( e_2 \) (corresponding to the condition entries) and \( \text{edge} (e_1) \neq \text{edge} (e_2) \). In other words, there exists at least one condition entry of any two rules has no intersection.
A propositional formula is used to describe a true sentence or situation. The situation is expressed by atomic formulas and the connectives in propositional logic. Similarly, in a \( DT_{OOLH} \), each rule is used to describe a situation with the condition entries and the corresponding action entry. To transform a decision table to a propositional formula, rules with action entry \( Y \) are the connective \( \lor \). The condition entries of each rule are connected by the connective \( \land \) which means 'and'. This propositional formula describes the logic of the decision table. As a result, based on the definition of decision tables in section 1.1, the logic that is represented by a completed decision table \( DT_{OOLH} \) can be expressed by a propositional formula in DNF. Given a decision table \( DT \), a DNF \( f \) is constructed as follows:

**Algorithm 4 \( AOBDD_{DT_{DNF}} \)**

1. find the rules with \( a_{1j} = Y \).
2. \( c_m = C_i \),
   while \( i \neq k \) \{ if \( d_{ij} = N \) \( c_m = c_m \land \neg C_i \) else \( c_m = c_m \land C_i \) \}
3. \( f = c_1 \lor c_2 \ldots \lor c_n \), where \( n \) : number of rules with \( a_{1j} = Y \)

For example, the propositional formula that expresses the logic of the decision table in figure 1 is \((\neg\text{solllage}) \lor (\neg\text{solllage} \land \neg Usp \land \neg EL(b)) \lor (\neg\text{solllage} \land \neg Usp \land EL(b) \land \neg bHSS \land \neg \text{spitz})\). DNF can be used to build the corresponding OBDD.

5 Analysis of Decision Tables and OBDDs

The OBDD technique provides defined operations for manipulating OBDDs. This is one of the reasons for applying this technique to represent a boolean function or the corresponding decision table. As mentioned above, if the expected behavior or the design of an RIS has been specified in a decision table, checking the correctness of OOLH or the conformity of this design against OOLH can also be accomplished in OOLH Tool. This decision table provided by users must satisfy the requirements of \( DT_{OOLH} \). The conformity of a decision table against these requirements and its completeness can be checked by building the corresponding OBDDs and manipulate them with the defined operations. The verification of decision tables can also be achieved by applying this method. In this section, the concepts of checking the conformity of a decision table against the requirements of \( DT_{OOLH} \) is explained and checking the completeness of a decision table is described in section 5.1, 5.2 and 5.3. The idea of applying the defined operations of OBDDs to verify decision tables is given in section 5.4.

5.1 Consistency of Decision tables

Given a decision table \( DT \), the consistency of rules in \( DT \) is checked based on the following steps:
Algorithm 5 $AOBDD_{\text{consistency}}$

1. Build $obdd_Y$ based on the rules with $a_{1j} = Y$
2. Build $obdd_N$ based on the rules with $a_{1j} = N$
3. Apply $obdd_Y \cdot obdd_N$ and the result is $obdd_{\text{consistency}}$.

A path with 1 as the terminal node in $obdd_{\text{consistency}}$ indicates the inconsistency of the decision table.

In section 1.1, the definition of consistency $DT_{\text{consistency}}$ has been given. A decision table contains inconsistency, if there exists a pair of overlapping rules and their action entries are different. Each path of an OBDD represents a rule. $obdd_Y$ is built based on the rules of the decision table. The action entries of these rules are $Y$. As a result, the paths of $obdd_Y$ with 1 as the terminal node represent the rules with the action entries $Y$ in the decision table. This set of paths must have 0 as the terminal node in $obdd_N$. If one of these paths has 1 as the terminal node in $obdd_N$, then the decision table contains a contradiction. It is because those paths of $obdd_N$ with 1 as the terminal node represent rules of the decision table with the action entry $N$. This concept is expressed by the propositional formula $obdd_Y \land obdd_N$ and the corresponding equivalent boolean function is $obdd_Y \cdot obdd_N$.

The decision table in figure 3(c) contains an inconsistency. This inconsistency is found by applying $AOBDD_{\text{consistency}}$ as illustrated in figure 16. The OBDD in figure 16(a) is built based on R1 because its action entry is $Y$. The results of step 2 are shown in figure 16(b). This figure is built based on R2. Figure 16(c) shows the result of the algorithm. There is a single path with 1 as the terminal node and the corresponding transformed decision rule is $(Y,Y,-)$. This is the inconsistent rule of the decision table as shown in figure 3.

![Figure 16: Inconsistency and OBDDs](image)

5.2 Completeness of Decision tables

Given a consistent decision table $DT$, the completeness of $DT$ is checked based on the following steps:

Algorithm 6 $AOBDD_{\text{completeness}}$
1. Build $obdd_Y$ based on the rules with $a_{1j} = Y$
2. Build $obdd_N$ based on the rules with $a_{1j} = N$
3. Apply $obdd_Y + obdd_N$ and the result is $obdd_{completeness}$.

This algorithm is called $AOBDD_{completeness}$. A path with 0 as the terminal node in $obdd_{completeness}$ indicates a missing rule of the decision table. If $obdd_{completeness}$ has only 1 as the terminal node, the decision table is complete.

There are two possible representations of the paths in $obdd_Y$ that lead to the terminal node 0. The first possible representation are rules with action entries $N$. The second representation are missing rules. The rules in the first case are supposed to be found in $obdd_N$. If these rules cannot be found in $obdd_N$ as a path with 1 as the terminal node, then they are the missing rules and the decision table is incomplete. This concept is expressed by the propositional formula $obdd_Y \lor obdd_N$ and the corresponding equivalent boolean function is $obdd_Y + obdd_N$.

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DWEL(b)</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>C2</td>
<td>soillage</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>C3</td>
<td>bHSS</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C4</td>
<td>z</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A1</td>
<td>überlappen</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 7: An incomplete decision table

For example, there are missing rules in Table 7. These rules can be found by applying $AOBDD_{completeness}$. Figure 17 illustrates the steps of $AOBDD_{completeness}$. The OBDD in figure 17(a) is $obdd_Y$ and $obdd_N$ is shown in figure 17(b). The result of the algorithm $obdd_{completeness}$ is illustrated in 17(c). The paths with 0 as the terminal node represent the missing rules. These paths can be transformed into decision rules based on the algorithm described in section 4.2. The missing rules are (Y,N,N,N,-) and (Y,N,Y,-,-).

![Figure 17: Completeness and OBDDs](image-url)
5.3 Exclusivity and Inclusiveness

Given a decision table $DT$, the exclusivity and inclusiveness of rules in $DT$ is checked based on the following steps:

Algorithm 7 $AOBDD_{exandin}$

1. Build $obdd_Y$ based on the rules with $a_{ij} = Y$ and apply $AOBDD_{transformation}$

2. Build $obdd_N$ based on the rules with $a_{ij} = N$ and apply $AOBDD_{transformation}$

This algorithm is called $AOBDD_{exandin}$. The paths of $obdd_Y$ with 1 as the terminal node represent the rules of $DT$ with action entry $Y$, while the paths of $obdd_N$ with 1 as the terminal node illustrate the rules with action entry $N$.

Based on the concepts and the definition of OBDDs, $obdd_Y$ and $obdd_N$ satisfies $DT_{exclusivity}$. $AOBDD_{transformation}$ is applied to ensure these two OBDDs satisfy $DT_{inclusiveness}$.

If all possible rules of $DT$ have been specified and it is consistent, then all paths of $obdd_Y$ represents the rules of $DT$. The following example illustrates the steps of applying $AOBDD_{exandin}$ to ensure the exclusivity and inclusiveness in a decision table (see table 8). In this decision table, the pairs of rules R2,R4 and R5,R6 do not fulfill $DT_{exclusivity}$. Furthermore, R8 and R9 can be combined because they lead to the same action and contain only one difference in the condition. Figure 18 (a) and (b) are $obdd_Y$ and $obdd_N$ respectively. The decision table in table 9 is deduced from these two OBDDs. R1, R2 and R3 are deduced from the $obdd_Y$, while R4, R5 and R6 are deduced from $obdd_N$. This decision table fulfills $DT_{exclusivity}$ and $DT_{inclusiveness}$ as it is shown.

In table 8, the pair of rules R3,R4 does not satisfy $DT_{inclusiveness}$. However, these two rules have not been combined based on the algorithm $AOBDD_{exandin}$. If R3 and R4 are combined together, the resulting rule tuple is $(Y,Y,-,Y,Y)$. This rule overlaps with R2 and the decision table does not fulfill $DT_{exclusivity}$. This shows the advantage of applying the defined operations of OBDDs to ensure the exclusivity and inclusiveness in decision tables. If these two properties are checked by comparing each pair of rules, the rules must be compared twice in principle in order to ensure both properties. Furthermore, if $DT_{inclusiveness}$ is first to be considered and R3 and R4 are consolidated, then the resulting decision table does not satisfy $DT_{exclusivity}$. The computational steps of checking $DT_{inclusiveness}$ are wasted.

There are two aspects which need to be addressed. The first aspect is the initial variable ordering of $obdd_Y$ and $obdd_N$ is first fixed based on the condition subject ordering found in the decision table. The first step of $AOBDD_{transformation}$ is not executed in $AOBDD_{exandin}$. The second aspect is the consistency of the decision table. When a decision table fulfills $DT_{exclusivity}$ and $DT_{inclusiveness}$, the decision table is not guaranteed to fulfill $DT_{consistency}$. As a result, $DT_{consistency}$ must be checked by applying $AOBDD_{consistency}$ before the application of $AOBDD_{exandin}$. 
Table 8: Exclusivity and Inclusiveness of rules in a decision table

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DWEL(b)</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C2</td>
<td>sollage</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>C3</td>
<td>bHSS</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>C4</td>
<td>z</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>A1</td>
<td>überlappen</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 9: Resulting decision table

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>DWEL(b)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>C2</td>
<td>sollage</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>C3</td>
<td>bHSS</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>C4</td>
<td>z</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A1</td>
<td>überlappen</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 18: $AOBDD_{exandin}$ and OBDDs

5.4 Verification

As it has mentioned before, one of the focus of these work is to check whether the system requirements specification conforms the expected behavior of a RIS. If the expected behavior of the system is described in a decision table and it satisfies the requirement $DT_{consistency}$ of $DT_{OOLH}$ and is complete, then the conformity can be checked by applying the concept of logical consequence $DT_{OOLH} \models DT_{expectedbehavior}$. This means, a situation leads to a further establishment or the issue of a safe route in $DT_{OOLH}$ must also be allowed in $DT_{expectedbehavior}$. If this sequent is valid, the situations that lead to a further establishment or the issue of a safe route in OOLH are expected. This idea is expressed by the propositional formula $DT_{OOLH} \rightarrow DT_{expectedbehavior}$. In the OBDD technique, the corresponding equivalent boolean function is expressed as $obdd_{OOLH} + obdd_{expectedbehavior}$. Those paths with 0 as the terminal node in the resultant OBDD illustrate rules that do not conform the expected
behavior of a RIS.

If $DT_{\text{expected behavior}}$ is incomplete, there are two possible interpretations of the unspecified rules. The first interpretation is these combinations of conditions lead to the rejection of developing safe routes. The second interpretation is these combinations of conditions will never happen in reality. In this work, the unspecified rules are automatically considered as rules with $a_{ij} = N$. The main reason is as follows. In RIS, if the situation of the railway system is not certain, neither safe routes should be issued nor the establishment of safe routes should be continued.

The concept in checking conformity of OOLH to the expected behavior can also be applied to verify the design of a RIS. This can be achieved by checking the logical consequence between the design and OOLH $DT_{\text{design}} \midDT_{\text{OOLH}}$. In this case, $DT_{\text{OOLH}}$ indicates those situations that are allowed to either continue the establishment or an issue of the safe route. If a situation is considered to be positive in the design, it needs to be considered as positive in the system requirements specification. By applying the same operation $\text{obdd}_{\text{design}} + \text{obdd}_{\text{OOLH}}$, the rules that do not conform the specification can be deduced from paths with 0 as the terminal node in the result OBDD.

In the railway domain, railway engineers need to check whether a combination of conditions of infrastructure elements might lead to an development of a safe route. The conventional way to handle this problem is to read the system requirements in LH-ESTW-R and draw the conclusion based on these requirements. As it has mentioned before, LH-ESTW-R is written in a natural language. Reading and understanding these safety critical requirements costs time. In OOLH, attributes of objects that need to be considered during the development of safe routes have been modelled. As a result, the efficiency of the checking process can be increased by specifying the situation based on these attributes in a decision table, $DT_{\text{situation}}$. The conformity of the situation to LH-ESTW-R can be checked by applying the concept of logical consequence $DT_{\text{situation}} \midDT_{\text{OOLH}}$. If the situation leads to an development of a safe route, the sequent is valid. This idea is expressed by the propositional formula $DT_{\text{situation}} \rightarrow DT_{\text{OOLH}}$ and the corresponding equivalent boolean function $\text{obdd}_{\text{situation}} + \text{obdd}_{\text{OOLH}}$. If the situation is valid, then the resultant OBDD has only 1 terminal node.

For example, one needs to decide whether the situation in figure 19(a) and (b) is an overlapping overlap. The concept of overlapping overlaps and these situations are defined as a decision table in table 6 and 20, respectively. The results of applying the operation $\text{obdd}_{\text{situation}} + \text{obdd}_{\text{OOLH}}$ is shown in figure 21. In figure 21(a), there is only one terminal node 1. This indicates the situation in figure 19(a) is a valid overlapping overlap. While, the OBDD in figure 21(b) has a path that leads to the 0 terminal node. This path represents the rule of the decision table in the right-hand side of Table 20. This means, this situation is not a valid overlapping overlap.
Figure 19: Two situations of a point

Figure 20: Decision table of situation (a) and (b)

Figure 21: Resultant OBDDs of situation (a) and (b)
6 Conclusion

The main focus of this note is to illustrate the concepts of transforming a propositional formula to a decision table. The decision tables that are used to express the semantics of propositional logic are first defined. They have three properties: consistency, exclusiveness and inclusiveness. OBDDs are used as the technique to support this transformation. The definition of OBDDs is also given formally in this note. Furthermore, checking whether a decision table satisfies the mentioned properties can be achieved by the defined operations in OBDDs.
Conclusion

References


