A propositional formula is determined up to logical equivalence by its truth table. If the formula has $n$ variables then its truth table requires space $\Omega(2^n)$ to represent. In this lecture we introduce a data structure called a binary decision diagram which gives a representation that is potentially much more compact. We furthermore show how binary decision diagrams can be used to decide satisfiability, validity, and logical equivalence.

While binary decision diagrams have been used successfully in practice, they don’t allow us to circumvent the worst-case difficulty of the various computational problems associated with propositional logic. Indeed, since there are $2^{2^n}$ different formulas on $n$ variables up to logical equivalence, we need space $\Omega(2^n)$ in the worst case to represent formulas up to logical equivalence.

1 Optimizing truth tables

Suppose we want to decide whether two propositional formulas $F$ and $G$ are logically equivalent. To do so, we can construct their truth tables, listing the involved propositional variables systematically in the same order, and compare if these are equal. For example, the truth table for $P \lor (Q \land R)$, where we list the variables in the order $P, Q, R$, is:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$R$</th>
<th>$P \lor (Q \land R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Of course, this is horribly inefficient: only 3 variables already require $2^3 = 8$ rows. We can try to do better. Looking at the last two rows of the table, where $P$ and $Q$ are assigned 1, we see that the formula evaluates to 1 regardless of the value assigned to $R$. Similarly for rows 5 and 6. One optimization of the table thus looks like:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$R$</th>
<th>$P \lor (Q \land R)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>1</td>
</tr>
</tbody>
</table>

We see that in fact, the last two rows don’t depend on $Q$ either. Also, we can collapse the first two rows, giving the following optimised truth table:
2 Binary decision diagrams

We formally represent the “optimised truth tables” above as *binary decision diagrams*. Like a truth table, a binary decision diagram represents the truth value of a formula under all possible assignments.

**Definition 1.** A *binary decision diagram* for a propositional formula $F$ is a directed acyclic graph such that:

- there is a distinguished *root* node, from which all other nodes are reachable;
- leaves are labelled 0 or 1;
- interior vertices are labelled with a propositional variable $P_i$ used in $F$;
- no propositional variable $P_i$ appears more than once in a branch from the root to a leaf;
- each interior vertex has two outgoing edges: the *false edge*, denoted by a dotted line, and the *true edge*, denoted by a solid line.

For example, here is a binary decision diagram for the formula $P \lor (Q \land R)$:

![Diagram](attachment:binary_decision_diagram.png)

Given a binary decision diagram, a *partial assignment* $A_b$ is associated with each branch $b$ from the root to a leaf. We define $A_b(P) = 1$ if the true edge was taken at vertex $P$ along branch $b$, and $A_b(P) = 0$ if the false edge was taken. The leaf of the branch $b$ is labelled with $A_b(F)$. If the assignment is partial, it must assign to enough propositional variables so that $A_b(F)$ is defined.

In the example above, the assignment $A_{010}$ that is associated with the branch that goes left, right, left, is given by $A_{010}(P) = 0$, $A_{010}(Q) = 1$, $A_{010}(R) = 0$.

**Definition 2.** Recall that two directed graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijection $f : V \to V'$ between the respective set of vertices such that $(u, v) \in E$ iff $(f(u), f(v)) \in E'$ for all $u, v \in V$, that is, $f$ preserves the edge relation. We say that two binary decision diagrams are isomorphic if there is an isomorphism of the underlying directed graphs that respects the labelling of the vertices and the edges (in particular, $(u, v)$ is a *false edge* if and only if $(f(u), f(v))$ is a *false edge*).
Input: Binary decision diagram
if there are more than two leaves then perform reduction 1
while possible do
begin
  if applicable then perform reduction 2
  if applicable then perform reduction 3
end
return

Figure 1: Binary decision diagram reduction algorithm

3 Reduction

The example binary decision diagram above is a special case, in which the directed acyclic graph is a tree, and a full assignment is associated with each branch. It is clear that these correspond precisely to a truth table, and hence are no more efficient. The key feature of binary decision diagrams is that we can compress them without losing the ability to evaluate the formula under all assignments. There are three such optimizations we can perform:

1. **Remove duplicate leaves**: if the decision diagram has more than one leaf with the same label, replace them by a single one, and redirect all edges which pointed to any of the old leaves to the single new one.

2. **Remove redundant tests**: if both outgoing edges of a vertex $P$ point to the same vertex $Q$, eliminate vertex $P$, and redirect all its incoming edges to $Q$.

3. **Remove duplicate interior vertices**: if two distinct vertices are the roots of isomorphic sub-binary decision diagrams, delete one of them, and redirect all its incoming edges to the other one.

Note that the first optimization is a special case of the third. We call these optimizations *reductions*, and we say a binary decision diagram is *reduced* when no reductions can be applied. Reduction generally turns a tree into a directed acyclic graph.

**Theorem 3.** Every propositional formula $F$ has a reduced binary decision diagram.

*Proof.* First, there always exists the full binary decision diagram for $F$. Now consider applying the algorithm in Figure 1. This algorithm surely terminates since each step reduces the number of nodes in the diagram. Observe moreover that the three reductions preserve the property of being a directed acyclic graph. Therefore the algorithm results in a well-defined binary decision diagram which, by the termination condition of the loop, is necessarily reduced. \qed
Let’s compute the reduced binary decision diagram of the full one for the formula $P \lor (Q \land R)$.

First, we merge all leaves into just two:

Next we apply reduction 2 repeatedly: once on the left-hand side of the diagram, and twice on the right-hand side, vertex $R$ has both outgoing edges leading to the same vertex.

We can now apply reduction 2 again to delete the right-hand vertex $Q$:

Now none of reduction 2 or 3 is applicable, so we have a reduced binary decision diagram. It has four branches, with the following associated assignments:

$$
\begin{align*}
&A_{0,0}(P) = 0 & A_{0,0}(Q) = 0 \\
&A_{0,1,0}(P) = 0 & A_{0,1,0}(Q) = 1 & A_{0,1,0}(R) = 0 \\
&A_{0,1,1}(P) = 0 & A_{0,1,1}(Q) = 1 & A_{0,1,1}(R) = 1 \\
&A_{1}(P) = 1
\end{align*}
$$

4 Ordered decision diagrams

Reduced binary decision diagrams are a potentially space-efficient representation of (the semantics of) a propositional formula. However, they lack the structure to permit efficient operations on
truth functions, such as equivalence testing and application of Boolean operations. For example, the following two binary decision diagrams are both reduced and represent the same truth table, but they are structurally unalike.

\[ \begin{array}{ccc}
0 & 1 & 1 \\
R & P & Q \\
Q & \end{array} \] 

\[ \begin{array}{ccc}
0 & 1 & 1 \\
R & P & Q \\
Q & \end{array} \] (2)

The problem is that the propositional variables are ordered differently along different branches. For example, the branch 00 is labelled QR in the left diagram, but PQ in the right one.

This motivates the following definition.

**Definition 4.** Let \((P_1, \ldots, P_n)\) be an ordered list of propositional variables without duplicates, and consider a binary decision diagram all of whose variables occur in the list. We say the list is an ordering for the diagram if \(i < j\) whenever \(P_i\) occurs before \(P_j\) in a branch from the root to a leaf. An ordered binary decision diagram is a binary decision diagram which has an ordering for some list of variables.

For example, the right diagram in (2), and the full tree of (1) both have ordering \((P, Q, R)\).

**Theorem 5.** Every propositional formula \(F\), equipped with an ordering of its propositional variables, has a unique reduced ordered binary decision diagram up to isomorphism.

**Proof.** Given a formula \(F\), the full tree gives an ordered binary decision diagram that represents \(F\). Since the algorithm in Figure 1 respects the variable ordering, we can use it to obtain a reduced ordered binary decision diagram that represents \(F\). This establishes existence and it remains to prove uniqueness.

Let \(G\) and \(G'\) be two reduced diagrams that represent \(F\). We show that \(G\) and \(G'\) are isomorphic. The proof is by induction on the number of variables in \(F\).

The key to the induction step is to show that the respective roots of \(G\) and \(G'\) have the same label. For this the following notion will be useful. Let us say that a variable \(P\) is dependent if there exists an assignment \(A\) such that \(A[P \rightarrow 1][F] \neq A[P \rightarrow 0][F]\). Otherwise we say that \(P\) is independent.

The induction step has two cases. The first case is that both \(G\) and \(G'\) consist of a single node. In either case \(G\) and \(G'\) are isomorphic.

The second case that one of \(G\) and \(G'\) does not consist of a single node (without loss of generality let it be \(G\)). Then \(G\) has root labelled by a propositional formula \(P\). Since \(G\) has no redundant test, the root must have two distinct children. Denote by \(G_0\) and \(G_1\) the respective diagrams rooted at the false child and true child of the root of \(G\). Then \(G_0\) represents \(F[\text{false}/P]\) and \(G_1\) represents \(F[\text{true}/P]\). Since \(G\) is reduced, \(G_0\) and \(G_1\) are non-isomorphic and so, by induction, it follows that \(F[\text{false}/P] \neq F[\text{true}/P]\). Thus \(P\) is a dependent variable and, being the root of \(G\), is the first dependent variable in the given variable ordering.
Since \( F \) has a dependent variable, \( G' \) cannot consist of a single node either. Repeating the same reasoning as for \( G \), it follows that the root vertex of \( G' \) is likewise labelled by the first dependent variable of \( F \), i.e., the roots of \( G \) and \( G' \) have the same label \( P \). Now write \( G'_0 \) and \( G'_1 \) for the two diagrams rooted at the \textit{false} child and \textit{true} child of the root of \( G' \). By the induction hypothesis we have that \( G_0 \) is isomorphic to \( G'_0 \) (since they are both reduced diagrams representing \( F[\text{false}/P] \)) and \( G_1 \) is isomorphic to \( G'_1 \) (since they are both reduced diagrams representing \( F[\text{true}/P] \)).

Now we can combine the isomorphism \( f_0 \) from \( G_0 \) to \( G'_0 \) and the isomorphism \( f_1 \) from \( G_1 \) to \( G'_1 \) to get an isomorphism \( f \) from \( G \) to \( G' \) (which maps the root of \( G \) to the root of \( G' \)). Note that we can indeed union \( f_0 \) and \( f_1 \) since \( f_0(u) = f_1(u) \) for any node \( u \) common to \( G_0 \) and \( G_1 \). Indeed, if \( H_0 \) and \( H_1 \) are the subgraphs of \( G' \) rooted at \( f_0(u) \) and \( f_1(u) \) respectively, then \( H_0 \) is isomorphic to \( H_1 \). But this forces \( f_0(u) = f_1(u) \) since \( G' \) is reduced.

From Theorem 3 we can consider a reduced ordered binary decision diagram as a \textit{canonical} representation of a formula. Such a representation suggests the following algorithms for deciding properties of formulas \( F, G \) with a chosen ordering of variables:

- \( F \) is \textit{satisfiable} if and only if 1 appears in its reduced ordered binary decision diagram;
- \( F \) is \textit{valid} if and only if its reduced ordered binary decision diagram is the single vertex 1;
- \( F \) and \( G \) are \textit{logically equivalent} if and only if they have equal reduced ordered binary decision diagrams.

For these algorithms we can choose any ordering of variables, but which one we choose influences the size of the resulting decision diagram. For example, the reduced ordered binary decision diagram for the formula \( (P_1 \land P_2) \lor \cdots \lor (P_{2n-1} \land P_{2n}) \) has only \( 2n + 2 \) vertices under the ordering \( (P_1, \ldots, P_{2n}) \), but as many as \( 2^n + 1 \) vertices under the ordering \( (P_1, P_3, \ldots, P_{2n-1}, P_2, P_4, \ldots, P_{2n}) \). With heuristics we can generally choose an efficient ordering, but in the worst case there are exponentially many vertices: there is a constant \( c > 0 \) such that for each \( n \) there is a formula whose reduced ordered binary decision diagram for any ordering has at least \( 2^{cn} \) vertices.

5 Operations on decision diagrams

Given a propositional formula \( F \), one way to obtain a reduced binary decision diagram is to construct the full tree and then apply the reduction algorithm. However this would defeat the purpose of having such a space-efficient representation. Fortunately we can perform Boolean operations directly on reduced binary decision diagrams, so we can construct a binary decision diagram by induction on the structure of \( F \).

\textbf{Definition 6.} Let \( F_1 \) and \( F_2 \) be propositional variables with ordered binary decision diagrams \( G_1 \) and \( G_2 \) defined with respect to a compatible ordering of the union of the propositional variables occurring in \( F_1 \) and \( F_2 \). For each operation \( \bullet \in \{ \lor, \land, \oplus \} \), we define an ordered binary decision for \( F_1 \bullet F_2 \) as follows:

The vertices of \( G_1 \bullet G_2 \) are pairs \((u, v)\), with \( u \) a vertex of \( G_1 \) and \( v \) a vertex of \( G_2 \). The root of \( G_1 \bullet G_2 \) is the pair \((u, v)\) such that \( u \) is the root of \( G_1 \) and \( v \) is the root of \( G_2 \). The remaining vertices of \( G_1 \bullet G_2 \) are constructed as follows.

Let \((u, v)\) be an vertex of \( G_1 \bullet G_2 \). Then
1. If $u$ and $v$ both have label $P$, respective false children $u_0, v_0$, and respective true children $u_1, v_1$, then $(u, v)$ has label $P$, false child $(u_0, v_0)$ and true child $(u_1, v_1)$.

2. If $u$ has label $P$, $v$ is either a leaf or has label $Q > P$, and $u$ has false child $u_0$ and true child $u_1$, then $(u, v)$ has label $P$, false child $(u_0, v)$, and true child $(u_1, v)$.

3. If $v$ has label $P$, $u$ is either a leaf or has label $Q > P$, and $v$ has false child $v_0$ and true child $v_1$, then $(u, v)$ has label $P$, false child $(u, v_0)$, and true child $(u, v_1)$.

4. If $u$ and $v$ are both leaves with respective labels $b_1, b_2 \in \{0, 1\}$, then $(u, v)$ is a leaf with label $b_1 \cdot b_2$.

The key property of $G_1 \cdot G_2$ is that each branch from a root to a leaf corresponds to a unique pair of root-to-leaf branches in $G_1$ and $G_2$.

Notice that the operation of negation on ordered binary decision diagrams can be implemented by simply interchanging the labels 0 and 1 on the leaves.

**Example 7.** Let’s apply the above construction with $F_1 = P$ and $F_2 = Q \land R$, with $G_1$ and $G_2$ as below, the ordering $(P, Q, R)$, and the operation $\lor$.

![Diagram](image)

Then we obtain

![Diagram](image)

which is indeed an ordered binary decision diagram for $P \lor (Q \land R)$.

Given reduced ordered binary decision diagrams $G_1$ and $G_2$, the diagram $G_1 \cdot G_2$ need not be reduced. However we can apply the algorithm in Figure 1 to obtain an equivalent reduced diagram.

6 **Comparison**

To summarise, let’s compare the different representations of propositional formulas that we’ve seen: propositional formulas themselves, formulas in DNF, formulas in CNF, truth tables, and binary decision diagrams. A representation can be *compact* or not, in the sense that it takes little memory to store the representation. It can be easy or hard to decide *satisfiability* and *validity*
in that representation. Finally, it can be easy or hard to perform operations on formulas in that representation, such as building the representation of a *conjunction*, *disjunction*, or *negation*, of formulas.

<table>
<thead>
<tr>
<th>Representation</th>
<th>compact</th>
<th>satisfiability</th>
<th>validity</th>
<th>(\wedge)</th>
<th>(\vee)</th>
<th>(\neg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulas</td>
<td>often</td>
<td>hard</td>
<td>hard</td>
<td>easy</td>
<td>easy</td>
<td>easy</td>
</tr>
<tr>
<td>DNF</td>
<td>sometimes</td>
<td>easy</td>
<td>hard</td>
<td>hard</td>
<td>easy</td>
<td>hard</td>
</tr>
<tr>
<td>CNF</td>
<td>sometimes</td>
<td>hard</td>
<td>easy</td>
<td>easy</td>
<td>hard</td>
<td>hard</td>
</tr>
<tr>
<td>Truth tables</td>
<td>never</td>
<td>hard</td>
<td>hard</td>
<td>hard</td>
<td>hard</td>
<td>hard</td>
</tr>
<tr>
<td>Decision diagrams</td>
<td>often</td>
<td>easy</td>
<td>easy</td>
<td>medium</td>
<td>medium</td>
<td>easy</td>
</tr>
</tbody>
</table>