Lecture 4

Shannon Expansion
Adequate Sets of Connectives
Unit Propagation
DPLL
Binary Decision Diagrams
Variable substitution (Replacement)

Variable replacement: given a Boolean formula $f$, a variable $v$, and an expression $e$, the notation $f[e/v]$ denotes the replacement of $v$ with $e$ in $f$.

Example: if $f = \neg x \land \neg y$ then

\[
\begin{align*}
f[F/y] &= \neg x \land \neg F = \neg x \land T = \neg x \\
f[T/x] &= \neg T \land \neg y = F \land \neg y = F \\
f[\neg r/y] &= \neg x \land \neg \neg r = \neg x \land r
\end{align*}
\]
Shannon Expansion

Given a variable $x$ in a formula $f$, $x$ can either be true or false. So, we can split the formula into two pieces—one in which $x$ is asserted to be false and we plug $F$ into $f$ for $x$, and one in which $x$ is asserted to be true and we plug $T$ into $f$ for $x$—and then take the disjunction.
Shannon Expansion

Given a variable $x$ in a formula $f$, $x$ can either be true or false. So, we can split the formula into two pieces—one in which $x$ is asserted to be false and we plug $F$ into $f$ for $x$, and one in which $x$ is asserted to be true and we plug $T$ into $f$ for $x$—and then take the disjunction.

Hence, we can write the logical equivalence:

$$f \equiv (x = F) \land f[F/x] \lor (x = T) \land f[T/x]$$
Shannon Expansion

Given a variable $x$ in a formula $f$, $x$ can either be true or false. So, we can split the formula into two pieces—one in which $x$ is asserted to be false and we plug $F$ into $f$ for $x$, and one in which $x$ is asserted to be true and we plug $T$ into $f$ for $x$—and then take the disjunction.

Hence, we can write the logical equivalence:

$$f \equiv (x = F) \land f[F/x] \lor (x = T) \land f[T/x]$$

Or, equivalently

$$f \equiv (\lnot x) \land f[F/x] \lor x \land f[T/x]$$
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$. 
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \land, \lor\}$ is adequate.
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \land, \lor\}$ is adequate.

**Idea:** induction on the number of variables in formula $f$. 
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \wedge, \vee\}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

- **Base Case:** $f$ has 0 variables
Adequate Set of Connectives

A set of connectives, $S$, is called **adequate** iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \land, \lor\}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

**Base Case:** $f$ has 0 variables

Then $f$ is equivalent to $T$ or $F$, both of which do not use connectives outside of $S$. 
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{ \neg, \land, \lor \}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

**Base Case:** $f$ has 0 variables

Then $f$ is equivalent to $T$ or $F$, both of which do not use connectives outside of $S$.

**Inductive Step:** $f$ has $n$ variables: $x_1, \ldots, x_n$
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \land, \lor\}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

**Base Case:** $f$ has 0 variables

Then $f$ is equivalent to $T$ or $F$, both of which do not use connectives outside of $S$.

**Inductive Step:** $f$ has $n$ variables: $x_1, \ldots, x_n$

Assume all formulas with less than $n$ variables can be written using only $S$ connectives.
Adequate Set of Connectives

A set of connectives, $S$, is called adequate iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{\neg, \wedge, \vee\}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

**Base Case:** $f$ has 0 variables

Then $f$ is equivalent to $T$ or $F$, both of which do not use connectives outside of $S$.

**Inductive Step:** $f$ has $n$ variables: $x_1, \ldots, x_n$

Assume all formulas with less than $n$ variables can be written using only $S$ connectives.

Shannon expand: $f \equiv \neg x \wedge f[F/x] \lor x \wedge f[T/x]$. 
Adequate Set of Connectives

A set of connectives, $S$, is called **adequate** iff every truth function can be written equivalently using connectives from $S$.

**Proposition:** the set $S = \{ \neg, \land, \lor \}$ is adequate.

**Idea:** induction on the number of variables in formula $f$

**Base Case:** $f$ has 0 variables

Then $f$ is equivalent to $T$ or $F$, both of which do not use connectives outside of $S$.

**Inductive Step:** $f$ has $n$ variables: $x_1, \ldots, x_n$

Assume all formulas with less than $n$ variables can be written using only $S$ connectives.

Shannon expand: $f \equiv \neg x \land f[F/x] \lor x \land f[T/x]$ 

$f[F/x_1]$ and $f[T/x_1]$ both have less than $n$ variables and so can be written with just $\{ \neg, \land, \lor \}$. 
DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).
Boolean Satisfiability

DPLL uses **Unit Propagation**.

\[ \phi = \{x \lor y, \neg x \lor z, z \lor w, x, y \lor v\} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).
Boolean Satisfiability

DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \lnot x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
Boolean Satisfiability

DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
DPLL uses **Unit Propagation**.

\[ \phi = \{ \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
DPLL uses **Unit Propagation**.

\[ \phi = \{ \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
Boolean Satisfiability

DPLL uses **Unit Propagation**.

$$\phi = \{ \, z, z \lor w, x, y \lor v \, \}$$

A *unit clause* is a clause that is composed of a single literal, $u$.

1. remove every clause (other than the unit clause) containing $u$.
2. in every clause that contains $\neg u$, delete $\neg u$. 
DPLL uses **Unit Propagation**.

\[ \phi = \{ \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
3. repeat.
Boolean Satisfiability

DPLL uses **Unit Propagation**.

\[
\phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}
\]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
3. repeat.
DPLL uses **Unit Propagation**.

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, *u*.

1. remove every clause (other than the unit clause) containing *u*.
2. in every clause that contains \( \neg u \), delete \( \neg u \).
3. repeat.
Boolean Satisfiability

DPLL uses **Unit Propagation**.

\[ \phi = \{ z, x, y \lor v \} \]

A *unit clause* is a clause that is composed of a single literal, \( u \).

1. remove every clause (other than the unit clause) containing \( u \).
2. in every clause that contains \( \neg u \), delete \( \neg u \).
3. repeat.
DPLL uses **Unit Propagation**.

\[ \phi = \{ \text{z, x, y} \lor \text{v} \} \]

A *unit clause* is a clause that is composed of a single literal, *u*.

1. remove every clause (other than the unit clause) containing *u*.
2. in every clause that contains \( \neg u \), delete \( \neg u \).
3. repeat.

\[ \phi = \{ \text{z, x, y} \lor \text{v} \} \]
Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Function : DPLL(\(\phi\))
Input : CNF formula \(\phi\) over \(n\) variables
Output : true or false, the satisfiability of \(F\)
begin
    UnitPropagate(\(\phi\))
    if \(\phi\) has false clause then return false
    if all clauses of \(\phi\) satisfied then return true
    \(x \leftarrow\) SelectBranchVariable(\(\phi\))
    return DPLL(\(\phi[x \mapsto \text{true}]\)) \(\lor\) DPLL(\(\phi[x \mapsto \text{false}]\))
end
DPLL Execution Example

{z, x, y ∨ v}

UNSAT

x ↦→ F

{z, T , y ∨ v}

x ↦→ T

{F, T , y ∨ v}UNSAT

z ↦→ ...

v ↦→ F

UNSAT

{T , T , F ∨ T }

v ↦→ T

SAT

{T , T , T ∨ v}

y ↦→ T

SAT

Result: φ is satisfiable.
DPLL Execution Example

\{z, x, y \lor v\}

\ \\ \ / \\ \\
\ \ \ x \mapsto F \\
\ / \ \\
\text{UNSAT} \ \{z, F, y \lor v\}

Result: \(\phi\) is satisfiable.
DPLL Execution Example

\[ \{z, x, y \lor v\} \]

\[ \{z, F, y \lor v\} \quad \text{UNSAT} \]

\[ x \mapsto F \quad x \mapsto T \]

\[ \text{UNSAT} \quad \{z, F, y \lor v\} \quad \{z, T, y \lor v\} \]
DPLL Execution Example

\{z, x, y \lor v\} \\
\quad \downarrow \\
\quad x \mapsto F \quad x \mapsto T \\
\quad \downarrow \\
\quad UNSAT \quad \{z, F, y \lor v\} \quad \{z, T, y \lor v\} \\
\quad \downarrow \\
\quad z \mapsto F \\
\quad \downarrow \\
\quad UNSAT \quad \{F, T, y \lor v\} \\

Result: \(\phi\) is satisfiable.
DPLL Execution Example

\{z, x, y \lor v\}
\{z, F, y \lor v\} \text{UNSAT}

x \mapsto F \quad x \mapsto T

\text{UNSAT} \quad \{z, F, y \lor v\} \quad \{z, T, y \lor v\}

\text{UNSAT} \quad \{F, T, y \lor v\} \quad \{T, T, y \lor v\}

\text{Result: } \varphi \text{ is satisfiable.
DPLL Execution Example

\{z, x, y \lor v\}

\{z, F, y \lor v\} UNSAT

\xrightarrow{\text{\textcolor{red}{F}}} \{z, T, y \lor v\}

\xrightarrow{\text{\textcolor{red}{T}}} \{F, T, y \lor v\} UNSAT

\xrightarrow{\text{\textcolor{blue}{F}}} \{T, T, y \lor v\}

\xrightarrow{\text{\textcolor{blue}{T}}} \{T, T, F \lor v\}

\text{Result: } \phi \text{ is satisfiable.}
DPLL Execution Example

\{z, x, y \lor v\}
\{z, F, y \lor v\} UNSAT
\begin{align*}
x \mapsto &~ F & x \mapsto &~ F \\
& & & & & & & & & & & & \text{UNSAT} & & \{z, F, y \lor v\} & & \{z, T, y \lor v\} \\
& & z \mapsto &~ F & z \mapsto &~ T \\
& & & & & & & & & & \text{UNSAT} & & \{F, T, y \lor v\} & & \{T, T, y \lor v\} \\
& & & & & & y \mapsto &~ F \\
& & & & & & & & & & \{T, T, F \lor v\} \\
& & & & & & & & & & & & v \mapsto &~ F \\
& & & & & & & & & & & & & & \text{UNSAT} & & \{T, T, F \lor F\} \\
\end{align*}

Result: \(\phi\) is satisfiable.
DPLL Execution Example

\[
\{z, x, y \lor v\}
\]

\[
x \mapsto F \quad x \mapsto T
\]

UNSAT \(
\{z, F, y \lor v\}
\)

UNSAT \(
\{z, T, y \lor v\}
\)

\[
z \mapsto F \quad z \mapsto T
\]

UNSAT \(
\{F, T, y \lor v\}
\)

\(
\{T, T, y \lor v\}
\)

\[
y \mapsto F
\]

\[
v \mapsto F \quad v \mapsto T
\]

UNSAT \(
\{T, T, F \lor v\}
\)

UNSAT \(
\{T, T, F \lor F\}
\)

\(
\{T, T, F \lor T\}
\)

SAT

Result: \( \phi \) is satisfiable.
DPLL Execution Example

\[
\{z, x, y \lor v\} \quad \text{UNSAT}
\]

\[
x \mapsto F \quad x \mapsto T
\]

\[
\text{UNSAT} \quad \{z, F, y \lor v\} \quad \{z, T, y \lor v\}
\]

\[
z \mapsto F \quad z \mapsto T
\]

\[
\text{UNSAT} \quad \{F, T, y \lor v\} \quad \{T, T, y \lor v\}
\]

\[
y \mapsto F \quad y \mapsto T
\]

\[
\{T, T, F \lor v\} \quad \{T, T, T \lor v\} \quad \text{SAT}
\]

\[
v \mapsto F \quad v \mapsto T
\]

\[
\text{UNSAT} \quad \{T, T, F \lor F\} \quad \{T, T, F \lor T\} \quad \text{SAT}
\]

Result: \(\phi\) is satisfiable.
DPLL Execution Example

Result: \( \phi \) is satisfiable.
Function : DPLL(\(\phi\))
Input : CNF formula \(\phi\) over \(n\) variables
Output : true or false, the satisfiability of \(F\)
begin
    UnitPropagate(\(\phi\))
    if \(\phi\) has false clause then return false
    if all clauses of \(\phi\) satisfied then return true
    \(x \leftarrow \) SelectBranchVariable(\(\phi\))
    return DPLL(\(\phi[x \rightarrow true]\)) \(\lor\) DPLL(\(\phi[x \rightarrow false]\))
end
DPLL can be converted into a procedure for \#CNF-SAT.

Function : $\text{DPLL}(\phi, t)$
Input : CNF formula $\phi$ over $n$ variables; $t \in \mathbb{Z}$
Output : $\#\phi$, the model count of $\phi$
begin
  UnitPropagate($\phi$)
  if $\phi$ has false clause then return 0
  if all clauses of $\phi$ satisfied then return $2^t$
  $x \leftarrow \text{SelectBranchVariable}(\phi)$
  return $\text{DPLL}(\phi[x \mapsto \text{true}], t - 1) + \text{DPLL}(\phi[x \mapsto \text{true}], t - 1)$
end
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, \ n = 5 \]

\[ \{z, x, y \lor v\} t = 5 \]
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, \ n = 5 \]

\[ \{ z, x, y \lor v \} t = 5 \]

\[ x \rightarrow F \]

\[ 0 \ \{ z, F, y \lor v \} t = 4 \]
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, \ n = 5 \]

\[ \{z, x, y \lor v\} t = 5 \]

\[ x \mapsto F \quad x \mapsto T \]

\[ 0 \ \{z, F, y \lor v\} t = 4 \quad \{z, T, y \lor v\} t = 4 \]
\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, \ n = 5 \]
Counting with DPLL

\[ \phi = \{x \lor y, \neg x \lor z, z \lor w, x, y \lor v\}, \ n = 5 \]

\{z, x, y \lor v\} \ t = 5

\begin{aligned}
&x \mapsto \ F & x \mapsto \ T \\
&\{z, F, y \lor v\} \ t = 4 & \{z, T, y \lor v\} \ t = 4 \\
&z \mapsto \ F & z \mapsto \ T \\
&\{F, T, y \lor v\} \ t = 3 & \{T, T, y \lor v\} \ t = 3
\end{aligned}

Result: 0 + 0 + 0 + 2 + 4 = 6 models
Counting with DPLL

\[ \phi = \{x \lor y, \neg x \lor z, z \lor w, x, y \lor v\}, \ n = 5 \]

\[
\begin{align*}
\{z, x, y \lor v\} & \quad t = 5 \\
\{z, F, y \lor v\} & \quad t = 4 \\
\{z, T, y \lor v\} & \quad t = 4 \\
\{F, T, y \lor v\} & \quad t = 3 \\
\{T, T, y \lor v\} & \quad t = 3 \\
\{F, T, F \lor v\} & \quad t = 3 \\
\{T, T, F \lor v\} & \quad t = 2
\end{align*}
\]

Result: 0 + 0 + 0 + 2 + 4 = 6 models
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, n = 5 \]

\[
\begin{align*}
\{z, x, y \lor v\} & \quad t = 5 \\
0 \quad \{z, F, y \lor v\} & \quad t = 4 \\
\{z, T, y \lor v\} & \quad t = 4 \\
0 \quad \{F, T, y \lor v\} & \quad t = 3 \\
\{T, T, y \lor v\} & \quad t = 3 \\
0 \quad \{F, T, y \lor v\} & \quad t = 3 \\
\{T, T, F \lor v\} & \quad t = 2 \\
0 \quad \{T, T, F \lor v\} & \quad t = 1
\end{align*}
\]
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, n = 5 \]

\[ \{z, x, y \lor v\} t = 5 \]

\[ x \mapsto F \]

\[ x \mapsto T \]

\[ 0 \{z, F, y \lor v\} t = 4 \]

\[ 0 \{z, T, y \lor v\} t = 4 \]

\[ z \mapsto F \]

\[ z \mapsto T \]

\[ 0 \{F, T, y \lor v\} t = 3 \]

\[ 0 \{F, T, y \lor v\} t = 3 \]

\[ y \mapsto F \]

\[ \{T, T, F \lor v\} t = 2 \]

\[ v \mapsto F \]

\[ v \mapsto T \]

\[ 0 \{T, T, F \lor F\} t = 1 \]

\[ 2^1 = 2 \{T, T, F \lor T\} t = 1 \]

Result: \( 0 + 0 + 0 + 2 + 4 = 6 \) models
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, n = 5 \]

\{z, x, y \lor v\} t = 5

\begin{array}{c}
\xrightarrow{x \rightarrow F} \\
\xrightarrow{x \rightarrow T} \\
\{z, F, y \lor v\} t = 4 & \{z, T, y \lor v\} t = 4 \\
\xrightarrow{z \rightarrow F} & \xrightarrow{z \rightarrow T} \\
\{F, T, y \lor v\} t = 3 & \{T, T, y \lor v\} t = 3 \\
\xrightarrow{y \rightarrow F} & \xrightarrow{y \rightarrow T} \\
\{T, T, F \lor v\} t = 2 & 2^2 = 4 \{T, T, T \lor v\} t = 2 \\
\xrightarrow{v \rightarrow F} & \xrightarrow{v \rightarrow T} \\
\{T, T, F \lor F\} t = 1 & 2^1 = 2 \{T, T, F \lor T\} t = 1
\end{array}

Result: 0 + 0 + 0 + 4 + 4 = 8 models

\[ 5 / 4 \]
Counting with DPLL

\[ \phi = \{ x \lor y, \neg x \lor z, z \lor w, x, y \lor v \}, n = 5 \]

\[ \{z, x, y \lor v\} t = 5 \]

\[ \begin{align*}
    &x \mapsto F \\
    &x \mapsto T \\
    &0 \{z, F, y \lor v\} t = 4 \\
    &\{z, T, y \lor v\} t = 4 \\
    &z \mapsto F \\
    &z \mapsto T \\
    &0 \{F, T, y \lor v\} t = 3 \\
    &\{T, T, y \lor v\} t = 3 \\
    &y \mapsto F \\
    &y \mapsto T \\
    &\{T, T, F \lor v\} t = 2 \\
    &2^2 = 4 \{T, T, T \lor v\} t = 2 \\
    &v \mapsto F \\
    &v \mapsto T \\
    &0 \{T, T, F \lor F\} t = 1 \\
    &2^1 = 2 \{T, T, F \lor T\} t = 1 \\
\end{align*} \]

Result: \[ 0 + 0 + 0 + 2 + 4 = 6 \text{ models} \]
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
A Binary Decision Diagram (BDD) is a data structure for representing the truth values of formulas in propositional logic.

Example: consider the formula $\neg x \land \neg y$ and the truth table.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\neg x \land \neg y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

The same information can be encoded in a decision tree.
We can make the decision diagram more compact.
We can make the decision diagram more compact.

Redundant terminal nodes
Binary Decision Diagrams

We can make the decision diagram more compact.

Redundant terminal nodes

Merge redundant nodes
We can make the decision diagram more compact.

Redundant terminal nodes

Merge redundant nodes

Redundant branches
We can make the decision diagram more compact.

We can remove redundant branches and merge redundant nodes to make the decision diagram more compact. A more compact representation of $\neg x \land \neg y$ is achieved by removing redundant nodes and branches.
Reduction Rule 1: Merge duplicated terminal nodes.
Reduction Rule 2: Remove redundant tests.
Reduction Rule 3: Remove duplicate sub-BDDs.

$BDD_1 \equiv BDD_2$
Reduction Rule 3: Remove duplicate sub-BDDs.

\[ BDD_1 \equiv BDD_2 \]

NOTE: They can be structurally identical, even if they overlap.
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Delete redundant tests
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Delete redundant tests
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
**Example:** Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Merge more duplicate terminals
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Redraw to “untangle”
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Redraw to “untangle”
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Redraw to “untangle”
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Remove duplicated sub-BDD
Example: Reduce the given BDD for the formula \((x \land y) \lor (\neg y \land z)\) to an ROBDD.

Remove duplicated sub-BDD
Example: Reduce the given BDD for the formula $(x \land y) \lor (\neg y \land z)$ to an ROBDD.

No more reduction possible
Important properties of BDDs

Ordered BDD (OBDD): variables are checked in a given order. E.g. $x > y > z$.

Reduced OBDD (ROBDD): Cannot be reduced any further.

Theorem: ROBDDs are unique for a given ordering.
Important properties of BDDs

**ROBDD size** is sensitive to variable ordering!

Two different ROBDDs for the formula

\[ x' \Leftrightarrow x \land y' \Leftrightarrow y \]

Variable order: \( x, x', y, y' \)  
Variable order: \( x, y, x', y' \)