

Digital Logic Design: a rigorous approach ©

Chapter 6: Propositional Logic

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

May 13, 2020

Book Homepage:

<http://www.eng.tau.ac.il/~guy/Even-Medina>

Building Blocks of Boolean Formulas

The building blocks of a Boolean formula are constants, variables, and connectives.

- 1 A **constant** is either 0 or 1. As in the case of bits, we interpret a 1 as “true” and a 0 as a “false”. The terms constant and bit are synonyms; the term bit is used in Boolean functions and in circuits while the term constants is used in Boolean formulas.
- 2 A **variable** is an element in a set of variables. We denote the set of variables by U . The set U does not contain constants. Variables are usually denoted by upper case letters.
- 3 **Connectives** are used to build longer formulas from shorter ones. We denote the set of connectives by \mathcal{C} .

Logical Connectives

We consider unary, binary, and higher arity connectives.

- 1 There is only one **unary connective** called **negation**. Negation of a variable A is denoted by $\text{NOT}(A)$, $\neg A$, or \bar{A} .
- 2 There are several **binary connectives**, the most common are AND (denoted also by \wedge or \cdot) and OR (denoted also by \vee or $+$). A binary connective is applied to two formulas. We later show the relation between binary connectives and Boolean functions $B : \{0, 1\}^2 \rightarrow \{0, 1\}$.
- 3 A connective has **arity** j if it is applied to j formulas. The arity of negation is 1, the arity of AND is 2, etc.

Example: parse tree

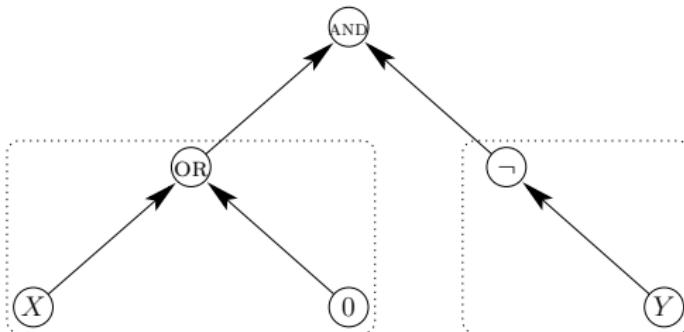


Figure: A parse tree that corresponds to the Boolean formula $((X \text{ OR } 0) \text{ AND } (\neg Y))$. The rooted trees that are hanging from the root of the parse tree (the AND connective) are bordered by dashed rectangles.

We use parse trees to define Boolean formulas.

Definition

A **parse tree** is a pair (G, π) , where $G = (V, E)$ is a rooted tree and $\pi : V \rightarrow \{0, 1\} \cup U \cup \mathcal{C}$ is a labeling function that satisfies:

- 1 A leaf is labeled by a constant or a variable. Formally, if $v \in V$ is a leaf, then $\pi(v) \in \{0, 1\} \cup U$.
- 2 An interior vertex v is labeled by a connective whose arity equals the in-degree of v . Formally, if $v \in V$ is an interior vertex, then $\pi(v) \in \mathcal{C}$ is a connective with arity $\deg_{in}(v)$.

We usually use only unary and binary connectives. Thus, unless stated otherwise, a parse tree has an in-degree of at most two.

- We use strings that contain constants, variables, connectives, and parenthesis to construct **Boolean formulas**.
- We use parse trees to define Boolean formulas.
- This definition is constructive (inorder traversal of the parse tree).

Examples of Good and Bad Formulas

- $(A \text{ AND } B)$
- $(A \text{ OR } B)$
- $A \text{ OR OR } B$ not a Boolean formula!
- $((A \text{ AND } B) \text{ OR } (A \text{ AND } C) \text{ OR } 1)$.
- If φ and ψ are Boolean formulas, then $(\varphi \text{ OR } \psi)$ is a Boolean formula.
- If φ is a Boolean formula, then $(\neg\varphi)$ is a Boolean formula.

We will stick to parse trees, and now show how they are parsed to generate valid Boolean formulas.

Algorithm 1 INORDER(G, π) - An algorithm for generating the Boolean formula corresponding to a parse tree (G, π) , where $G = (V, E)$ is a rooted tree with in-degree at most 2 and $\pi : V \rightarrow \{0, 1\} \cup U \cup \mathcal{C}$ is a labeling function.

- 1 Base Case: If $|V| = 1$ then return $\pi(v)$ (where $v \in V$ is the only node in V)
 - 2 Reduction Rule:
 - 1 If $\text{deg}_{in}(r(G)) = 1$, then
 - 1 Let $G_1 = (V_1, E_1)$ denote the rooted tree hanging from $r(G)$.
 - 2 Let π_1 denote the restriction of π to V_1 .
 - 3 $\alpha \leftarrow \text{INORDER}(G_1, \pi_1)$.
 - 4 Return $(\neg\alpha)$.
 - 2 If $\text{deg}_{in}(r(G)) = 2$, then
 - 1 Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ denote the rooted subtrees hanging from $r(G)$.
 - 2 Let π_i denote the restriction of π to V_i .
 - 3 $\alpha \leftarrow \text{INORDER}(G_1, \pi_1)$.
 - 4 $\beta \leftarrow \text{INORDER}(G_2, \pi_2)$.
 - 5 Return $(\alpha \ \pi(r(G)) \ \beta)$.
-

Definition

Let (G, π) denote a parse tree and let T_v denote the subtree hanging from v .

- The output φ of $\text{INORDER}(G, \pi)$ is a **Boolean formula**.
- The output of $\text{INORDER}(T_v, \pi)$ is a **subformula** of φ .

We say that Boolean formula φ is defined by the parse tree (G, π) .

- Consider all the parse trees over the set of variables U and the set of connectives \mathcal{C} .
- The set of all Boolean formulas defined by these parse trees is denoted by $\mathcal{BF}(U, \mathcal{C})$.
- To simplify notation, we abbreviate $\mathcal{BF}(U, \mathcal{C})$ by \mathcal{BF} when the sets of variables and connectives are known.

Examples

Some of the connectives have several notations. The following formulas are the same, i.e. string equality.

$$(A + B) = (A \vee B) = (A \text{ OR } B),$$

$$(A \cdot B) = (A \wedge B) = (A \text{ AND } B),$$

$$(\neg B) = (\text{NOT}(B)) = (\bar{B}),$$

$$(A \text{ XOR } B) = (A \oplus B),$$

$$((A \vee C) \wedge (\neg B)) = ((A + C) \cdot (\bar{B})).$$

We sometimes omit parentheses from formulas if their parse tree is obvious. When parenthesis are omitted, one should use precedence rules as in arithmetic, e.g., $a \cdot b + c \cdot d = ((a \cdot b) + (c \cdot d))$.

The Implication Connective

The implication connective is denoted by \rightarrow .

X	Y	$X \rightarrow Y$	\rightarrow	0	1
0	0	1	0	1	1
1	0	0	1	0	1
0	1	1	1	0	1
1	1	1			

Table: The truth table representation and the multiplication table of the implication connective.

Lemma

$A \rightarrow B$ is true iff $A \leq B$.

- The implication connective is not commutative, namely, $(0 \rightarrow 1) \neq (1 \rightarrow 0)$.
- This connective is called implication since it models the natural language templates “ Y if X ” and “if X then Y ”.
- Note that $X \rightarrow Y$ is always 1 if $X = 0$.

Connectives NAND NOR

$$\begin{aligned}\text{NAND}(A, B) &\triangleq \text{NOT}(\text{AND}(A, B)) , \\ \text{NOR}(A, B) &\triangleq \text{NOT}(\text{OR}(A, B)) .\end{aligned}$$

Truth Tables

X	Y	$X \text{ NAND } Y$
0	0	1
1	0	1
0	1	1
1	1	0

X	Y	$X \text{ NOR } Y$
0	0	1
1	0	0
0	1	0
1	1	0

NAND	0	1
0	1	1
1	1	0

NOR	0	1
0	1	0
1	0	0

The Equivalence Connective

The equivalence connective is denoted by \leftrightarrow .

$(p \leftrightarrow q)$ abbreviates $((p \rightarrow q) \text{ AND } (q \rightarrow p))$.

X	Y	$X \leftrightarrow Y$	\leftrightarrow	0	1
0	0	1	0	1	0
1	0	0	1	0	0
0	1	0	0	0	1
1	1	1	1	0	1

$$(X \leftrightarrow Y) = \begin{cases} 1 & \text{if } X = Y \\ 0 & \text{if } X \neq Y. \end{cases}$$

Order Matters!

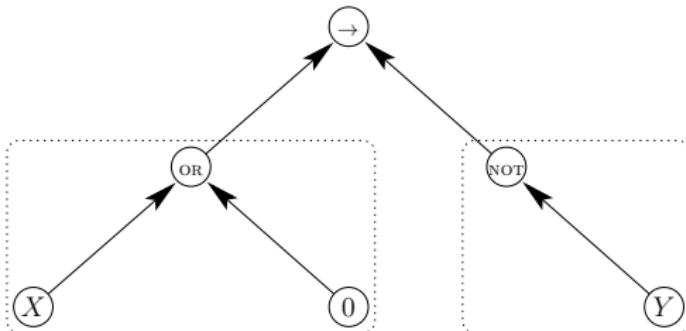


Figure: The parse tree of the Boolean formula $((X \text{ OR } 0) \rightarrow (\neg Y))$. The root is labeled by an implication connective. The rooted trees hanging from the root are encapsulated by dashed rectangles.

Recapping

- Variables: X, Y, Z, \dots
- Logical connectives:
 - unary: NOT
 - binary: AND, OR, NOR, NAND, \rightarrow , \leftrightarrow
- Parse Trees: rooted tree labeled by variables and connectives.
- Boolean Formula: defined by inorder traversal of parse tree.
- Attach Boolean operators to logical connectives.

Syntax vs. Semantics

- **Syntax** - grammatic rules that govern the construction of Boolean formulas (rules: parse trees + inorder traversal)
- **Semantics** - functional interpretation of a formula

Syntax has a purpose: to provide well defined semantics!

Syntax vs. Semantics

Logical connectives have two roles:

- **Syntax**: building block for Boolean formulas (“glue”).
- **Semantics**: define a truth value based on a Boolean function.

To emphasize the semantic role: given a k -ary connective $*$, we denote the semantics of $*$ by a Boolean function

$$B_* : \{0, 1\}^k \rightarrow \{0, 1\}$$

Example

- $B_{\text{AND}}(b_1, b_2) = b_1 \cdot b_2$.
- $B_{\text{NOT}}(b) = 1 - b$.

Semantics of Variables and Constants

- The function B_X associated with a variable X is the identity function $B_X(b) = b$.
- The function B_σ associated with a constant $\sigma \in \{0, 1\}$ is the constant function $B_\sigma(b) = \sigma$.

truth assignments

Let U denote the set of variables.

Definition

A **truth assignment** is a function $\tau : U \rightarrow \{0, 1\}$.

Our goal is to extend every assignment $\tau : U \rightarrow \{0, 1\}$ to a function

$$\hat{\tau} : \mathcal{BF}(U, \mathcal{C}) \rightarrow \{0, 1\}$$

Thus, a truth assignment to variables actually induces truth values to every Boolean formula.

The extension $\hat{\tau} : \mathcal{BF} \rightarrow \{0, 1\}$ of an assignment $\tau : U \rightarrow \{0, 1\}$ is defined as follows.

Definition

Let $p \in \mathcal{BF}$ be a Boolean formula generated by a parse tree (G, π) . Then,

$$\hat{\tau}(p) \triangleq \text{EVAL}(G, \pi, \tau),$$

where EVAL is listed in the next slide.

EVAL is also an algorithm that also employs inorder traversal over the parse tree!

Algorithm 2 $\text{EVAL}(G, \pi, \tau)$ - evaluate the truth value of the Boolean formula generated by the parse tree (G, π) , where (i) $G = (V, E)$ is a rooted tree with in-degree at most 2, (ii) $\pi : V \rightarrow \{0, 1\} \cup U \cup \mathcal{C}$, and (iii) $\tau : U \rightarrow \{0, 1\}$ is an assignment.

① Base Case: If $|V| = 1$ then

- ① Let $v \in V$ be the only node in V .
- ② $\pi(v)$ is a constant: If $\pi(v) \in \{0, 1\}$ then return $(\pi(v))$.
- ③ $\pi(v)$ is a variable: return $(\tau(\pi(v)))$.

② Reduction Rule:

- ① If $\text{deg}_{in}(r(G)) = 1$, then (*in this case* $\pi(r(G)) = \text{NOT}$)
 - ① Let $G_1 = (V_1, E_1)$ denote the rooted tree hanging from $r(G)$.
 - ② Let π_1 denote the restriction of π to V_1 .
 - ③ $\sigma \leftarrow \text{EVAL}(G_1, \pi_1, \tau)$.
 - ④ Return $(\text{NOT}(\sigma))$.
- ② If $\text{deg}_{in}(r(G)) = 2$, then
 - ① Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ denote the rooted subtrees hanging from $r(G)$.
 - ② Let π_i denote the restriction of π to V_i .
 - ③ $\sigma_1 \leftarrow \text{EVAL}(G_1, \pi_1, \tau)$.
 - ④ $\sigma_2 \leftarrow \text{EVAL}(G_2, \pi_2, \tau)$.
 - ⑤ Return $(B_{\pi(r(G))}(\sigma_1, \sigma_2))$.

Evaluation:

- Fix a truth assignment $\tau : U \rightarrow \{0, 1\}$.
- Extend τ to every Boolean formula $p \in \mathcal{BF}$.

Formula as a function:

- Fix a Boolean formula p .
- Consider all possible truth assignments $\tau : U \rightarrow \{0, 1\}$.

Definition

Let p denote a Boolean formula.

- ① p is **satisfiable** if there exists an assignment τ such that $\hat{\tau}(p) = 1$.
- ② p is a **tautology** if $\hat{\tau}(p) = 1$ for every assignment τ .

Definition

Two formulas p and q are **logically equivalent** if $\hat{\tau}(p) = \hat{\tau}(q)$ for every assignment τ .

Examples

- 1 Show that $\varphi \stackrel{\triangle}{=} (X \oplus Y)$ is satisfiable.
- 2 Let $\varphi \stackrel{\triangle}{=} (X \vee \neg X)$. Show that φ is a tautology.

$\tau(X)$	$\text{NOT}(\tau(X))$	$\hat{\tau}(X \vee \neg X)$
0	1	1
1	0	1

more examples

Let $\varphi \triangleq (X \oplus Y)$, and let $\psi \triangleq (\bar{X} \cdot Y + X \cdot \bar{Y})$. Show that φ and ψ are logically equivalent.

We show that $\hat{\tau}(\varphi) = \hat{\tau}(\psi)$ for every assignment τ . We do that by enumerating all the $2^{|U|}$ assignments.

$\tau(X)$	$\tau(Y)$	$\text{AND}(\text{NOT}(\tau(X)), \tau(Y))$	$\text{AND}(\tau(X), \text{NOT}(\tau(Y)))$	$\hat{\tau}(\varphi)$	$\hat{\tau}(\psi)$
0	0	0	0	0	0
1	0	0	1	1	1
0	1	1	0	1	1
1	1	0	0	0	0

Table: There are two variables, hence the enumeration consists of $2^2 = 4$ assignments. The columns that correspond to $\hat{\tau}(\varphi)$ and $\hat{\tau}(\psi)$ are identical, hence φ and ψ are equivalent.

Satisfiability and Tautologies

Lemma

Let $\varphi \in \mathcal{BF}$, then

φ is satisfiable $\Leftrightarrow (\neg\varphi)$ is not a tautology .

Proof.

$$\begin{aligned}\varphi \text{ is satisfiable} &\Leftrightarrow \exists \tau : \hat{\tau}(\varphi) = 1 \\ &\Leftrightarrow \exists \tau : \text{NOT}(\hat{\tau}(\varphi)) = 0 \\ &\Leftrightarrow \exists \tau : \hat{\tau}(\neg(\varphi)) = 0 \\ &\Leftrightarrow (\neg\varphi) \text{ is not a tautology} .\end{aligned}$$

□

Every Boolean String Represents an Assignment

Definition

Given a binary vector $v = (v_1, \dots, v_n) \in \{0, 1\}^n$, the assignment $\tau_v : \{X_1, \dots, X_n\} \rightarrow \{0, 1\}$ is defined by $\tau_v(X_i) \triangleq v_i$.

Example

Let $n = 3$.

$$v[1 : 3] = 011$$

$$\tau_v(X_1) = v[1] = 0$$

$$\tau_v(X_2) = v[2] = 1$$

$$\tau_v(X_3) = v[3] = 1$$

Question

Prove that $v \mapsto \tau_v$ is a bijection from $\{0, 1\}^n$ to truth assignments

$$\{\tau \mid \tau : \{X_1, \dots, X_n\} \rightarrow \{0, 1\}\}.$$

Every Boolean Formula Represents a Function

Assume that $U = \{X_1, \dots, X_n\}$.

Definition

A Boolean formula p over the variables $U = \{X_1, \dots, X_n\}$ defines the Boolean function $B_p : \{0, 1\}^n \rightarrow \{0, 1\}$ by

$$B_p(v_1, \dots, v_n) \triangleq \hat{\tau}_v(p).$$

Example

$$p = X_1 \vee X_2$$

$$B_p(0, 0) = 0, \quad B_p(0, 1) = 1, \dots$$

Every Boolean Formula Represents a Function (cont)

Assume that $U = \{X_1, \dots, X_n\}$.

Definition

A Boolean formula p over the variables $U = \{X_1, \dots, X_n\}$ defines the Boolean function $B_p : \{0, 1\}^n \rightarrow \{0, 1\}$ by

$$B_p(v_1, \dots, v_n) \stackrel{\triangle}{=} \hat{\tau}_v(p).$$

The mapping $p \mapsto B_p$ is a function from $\mathcal{BF}(U, \mathcal{C})$ to set of Boolean functions $\{0, 1\}^{\{\{0, 1\}^n\}}$. Is this mapping one-to-one? is it onto?

Every Tautology Induces a Constant Function

Claim

A Boolean formula p is a tautology if and only if the Boolean function B_p is identically one, i.e., $B_p(v) = 1$, for every $v \in \{0, 1\}^n$.

Proof.

$$\begin{aligned} p \text{ is a tautology} &\Leftrightarrow \forall \tau : \hat{\tau}(p) = 1 \\ &\Leftrightarrow \forall v \in \{0, 1\}^n : \hat{\tau}_v(p) = 1 \\ &\Leftrightarrow \forall v \in \{0, 1\}^n : B_p(v) = 1. \end{aligned}$$



Claim

A Boolean formula p is a satisfiable if and only if the Boolean function B_p is not identically zero, i.e., there exists a vector $v \in \{0, 1\}^n$ such that $B_p(v) = 1$.

Proof.

$$\begin{aligned} p \text{ is a satisfiable} &\Leftrightarrow \exists \tau : \hat{\tau}(p) = 1 \\ &\Leftrightarrow \exists v \in \{0, 1\}^n : \hat{\tau}_v(p) = 1 \\ &\Leftrightarrow \exists v \in \{0, 1\}^n : B_p(v) = 1 . \end{aligned}$$



Claim

Two Boolean formulas p and q are logically equivalent if and only if the Boolean functions B_p and B_q are identical, i.e., $B_p(v) = B_q(v)$, for every $v \in \{0, 1\}^n$.

Proof.

p and q are logically equivalent

$$\Leftrightarrow \forall \tau : \hat{\tau}(p) = \hat{\tau}(q)$$

$$\Leftrightarrow \forall v \in \{0, 1\}^n : \hat{\tau}_v(p) = \hat{\tau}_v(q)$$

$$\Leftrightarrow \forall v \in \{0, 1\}^n : B_p(v) = B_q(v) .$$



Example: Composition of Boolean formulas

If $\varphi = (\alpha_1 \text{ AND } \alpha_2)$, then

$$\begin{aligned} B_\varphi(v) &= \hat{\tau}_v(\varphi) \\ &= \hat{\tau}_v(\alpha_1 \text{ AND } \alpha_2) \\ &= B_{\text{AND}}(\hat{\tau}_v(\alpha_1), \hat{\tau}_v(\alpha_2)) \\ &= B_{\text{AND}}(B_{\alpha_1}(v), B_{\alpha_2}(v)). \end{aligned}$$

Thus, we can express complicated Boolean functions by composing long Boolean formulas.

Composition of Boolean formulas

Lemma

If $\varphi = \alpha_1 \circ \alpha_2$ for a binary connective \circ , then

$$\forall v \in \{0, 1\}^n : B_\varphi(v) = B_\circ(B_{\alpha_1}(v), B_{\alpha_2}(v)).$$

Claim

Two Boolean formulas p and q are logically equivalent if and only if the formula $(p \leftrightarrow q)$ is a tautology.

substitution

Substitution is used to compose large formulas from smaller ones. For simplicity, we deal with substitution in formulas over two variables; the generalization to formulas over any number of variables is straightforward.

- ① $\varphi \in \mathcal{BF}(\{X_1, X_2\}, \mathcal{C})$,
- ② $\alpha_1, \alpha_2 \in \mathcal{BF}(U, \mathcal{C})$.
- ③ (G_φ, π_φ) denotes the parse tree of φ .

Definition

Substitution of α_i in φ yields the Boolean formula $\varphi(\alpha_1, \alpha_2) \in \mathcal{BF}(U, \mathcal{C})$ that is generated by the parse tree (G, π) defined as follows.

For every leaf of $v \in G_\varphi$ that is labeled by a variable X_i , replace the leaf v by a new copy of $(G_{\alpha_i}, \pi_{\alpha_i})$.

example: substitution

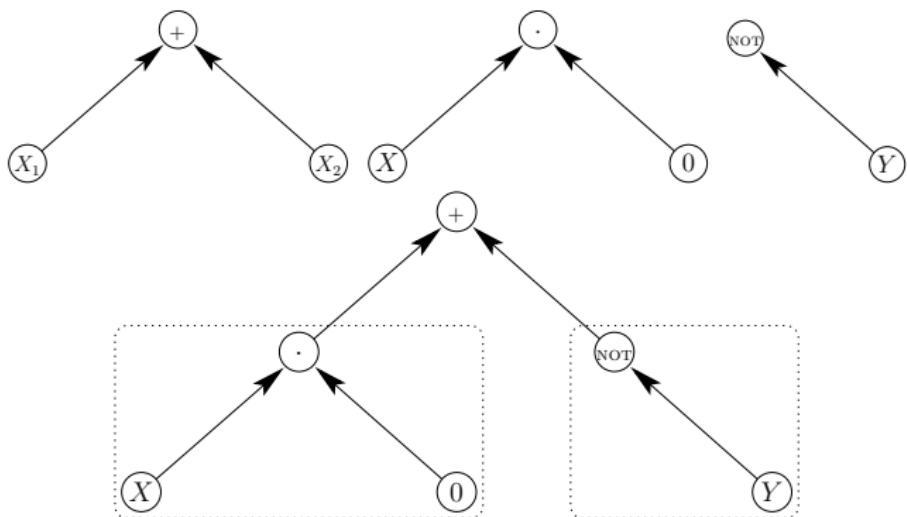


Figure: $\varphi, \alpha_1, \alpha_2, \varphi(\alpha_1, \alpha_2)$

Substitution can be obtained by applying a simple “find-and-replace”, where each instance of variable X_i is replaced by a copy of the formula α_i , for $i \in \{1, 2\}$.

One can easily generalize substitution to formulas

$\varphi \in \mathcal{BF}(\{X_1, \dots, X_k\}, \mathcal{C})$ for any $k > 2$. In this case, $\varphi(\alpha_1, \dots, \alpha_k)$ is obtained by replacing every instance of X_i by α_i .

Lemma

For every assignment $\tau : U \rightarrow \{0, 1\}$,

$$\hat{\tau}(\varphi(\alpha_1, \alpha_2)) = B_\varphi(\hat{\tau}(\alpha_1), \hat{\tau}(\alpha_2)). \quad (1)$$

substitution preserves logical equivalence

Let

- $\varphi \in \mathcal{BF}(\{X_1, X_2\}, \mathcal{C})$,
- $\alpha_1, \alpha_2 \in \mathcal{BF}(U, \mathcal{C})$,
- $\tilde{\varphi} \in \mathcal{BF}(\{X_1, X_2\}, \tilde{\mathcal{C}})$,
- $\tilde{\alpha}_1, \tilde{\alpha}_2 \in \mathcal{BF}(U, \tilde{\mathcal{C}})$.

Corollary

If α_i and $\tilde{\alpha}_i$ are logically equivalent, and φ and $\tilde{\varphi}$ are logically equivalent, then $\varphi(\alpha_1, \alpha_2)$ and $\tilde{\varphi}(\tilde{\alpha}_1, \tilde{\alpha}_2)$ are logically equivalent.

Example

$$\varphi = \neg(X_1 \cdot X_2)$$

$$\alpha_1 = A \rightarrow B$$

$$\alpha_2 = C \leftrightarrow D$$

$$\tilde{\varphi} = \bar{X}_1 + \bar{X}_2$$

$$\tilde{\alpha}_1 = \bar{A} + B$$

$$\tilde{\alpha}_2 = \neg(C \oplus D)$$

example: changing connectives

Let $\mathcal{C} = \{\text{AND}, \text{XOR}\}$. We wish to find a formula

$\tilde{\beta} \in \mathcal{BF}(\{X, Y, Z\}, \mathcal{C})$ that is logically equivalent to the formula

$$\beta \triangleq (X \cdot Y) + Z.$$

Parse β : $\varphi(\alpha_1, \alpha_2)$ with $\alpha_1 = (X \cdot Y)$ and $\alpha_2 = Z$.

Find $\tilde{\varphi} \in \mathcal{BF}(\{X_1, X_2\}, \mathcal{C})$ that is logically equivalent to

$$\varphi \triangleq (X_1 + X_2).$$

$$\tilde{\varphi} \triangleq X_1 \oplus X_2 \oplus (X_1 \cdot X_2).$$

Apply substitution to define $\tilde{\beta} \triangleq \tilde{\varphi}(\alpha_1, \alpha_2)$, thus

$$\begin{aligned}\tilde{\beta} &\triangleq \tilde{\varphi}(\alpha_1, \alpha_2) \\ &= \alpha_1 \oplus \alpha_2 \oplus (\alpha_1 \cdot \alpha_2) \\ &= (X \cdot Y) \oplus Z \oplus ((X \cdot Y) \cdot Z)\end{aligned}$$

Indeed $\tilde{\beta}$ is logically equivalent to β .

Complete Sets of Connectives

Every Boolean formula can be interpreted as Boolean function. In this section we deal with the following question: Which sets of connectives enable us to express every Boolean function?

Definition

A Boolean function $B : \{0, 1\}^n \rightarrow \{0, 1\}$ is **expressible** by $\mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$ if there exists a formula $p \in \mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$ such that $B = B_p$.

Definition

A set \mathcal{C} of connectives is **complete** if every Boolean function $B : \{0, 1\}^n \rightarrow \{0, 1\}$ is expressible by $\mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$.

Completeness of $\{\neg, \text{AND}, \text{OR}\}$

Theorem

The set $\mathcal{C} = \{\neg, \text{AND}, \text{OR}\}$ is a complete set of connectives.

Proof Outline: Induction on n (the arity of Boolean function).

- ① Induction basis for $n = 1$.
- ② Induction step for $B : \{0, 1\}^n \rightarrow \{0, 1\}$ define:

$$g(v_1, \dots, v_{n-1}) \triangleq B(v_1, \dots, v_{n-1}, 0),$$

$$h(v_1, \dots, v_{n-1}) \triangleq B(v_1, \dots, v_{n-1}, 1).$$

- ③ By induction hyp. $\exists r, q \in \mathcal{BF}(\{X_1, \dots, X_{n-1}\}, \mathcal{C})$:
 $B_r = h$ and $B_q = g$
- ④ Prove that $B_p = B$ for the formula p defined by

$$p \triangleq (q \cdot \bar{X}_n) + (r \cdot X_n)$$

Theorem: changing connectives

Theorem

If the Boolean functions in $\{\text{NOT, AND, OR}\}$ are expressible by formulas in $\mathcal{BF}(\{X_1, X_2\}, \tilde{\mathcal{C}})$, then $\tilde{\mathcal{C}}$ is a complete set of connectives.

Proof Outline:

- ① Express $\beta \in \mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$ by a logically equivalent formula $\tilde{\beta} \in \mathcal{BF}(\{X_1, \dots, X_n\}, \tilde{\mathcal{C}})$.
- ② How? induction on the parse tree that generates β .

Important Tautologies

Theorem

The following Boolean formulas are tautologies.

- 1 *law of excluded middle:* $X + \bar{X}$
- 2 *double negation:* $X \leftrightarrow (\neg\neg X)$
- 3 *modus ponens:* $((X \rightarrow Y) \cdot X) \rightarrow Y$
- 4 *contrapositive:* $(X \rightarrow Y) \leftrightarrow (\bar{Y} \rightarrow \bar{X})$
- 5 *material implication:* $(X \rightarrow Y) \leftrightarrow (\bar{X} + Y)$.
- 6 *distribution:* $X \cdot (Y + Z) \leftrightarrow (X \cdot Y + X \cdot Z)$.

Substitution in Tautologies

Recall the lemma:

Lemma

For every assignment $\tau : U \rightarrow \{0, 1\}$,

$$\hat{\tau}(\varphi(\alpha_1, \alpha_2)) = B_\varphi(\hat{\tau}(\alpha_1), \hat{\tau}(\alpha_2)). \quad (2)$$

question

Let α_1 and α_2 be any Boolean formulas.

- 1 Consider the Boolean formula $\varphi \triangleq \alpha_1 + \text{NOT}(\alpha_1)$. Prove or refute that φ is a tautology.
- 2 Consider the Boolean formula $\varphi \triangleq (\alpha_1 \rightarrow \alpha_2) \leftrightarrow (\text{NOT}(\alpha_1) + \alpha_2)$. Prove or refute that φ is a tautology.

De Morgan's Laws

Theorem (De Morgan's Laws)

The following two Boolean formulas are tautologies:

- ① $(\neg(X + Y)) \leftrightarrow (\bar{X} \cdot \bar{Y})$.
- ② $(\neg(X \cdot Y)) \leftrightarrow (\bar{X} + \bar{Y})$.

De Morgan Dual

Given a Boolean Formula $\varphi \in \mathcal{BF}(U, \{\vee, \wedge, \neg\})$, apply the following “replacements”:

- $X_i \mapsto \neg X_i$
- $\neg X_i \mapsto X_i$
- $\vee \mapsto \wedge$
- $\wedge \mapsto \vee$

What do you get?

Example

$$\varphi = (X_1 + \neg X_2) \cdot (\neg X_2 + X_3)$$

is replaced by

$$\text{dual}(\varphi) = (\neg X_1 \cdot X_2) + (X_2 \cdot \neg X_3).$$

What is the relation between φ and $\text{dual}(\varphi)$?

De Morgan Dual

We define the De Morgan Dual using a recursive algorithm.

Algorithm 3 $\text{DM}(\varphi)$ - An algorithm for computing the De Morgan dual of a Boolean formula $\varphi \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$.

① Base Cases:

- ① If $\varphi = 0$, then return 1. If $\varphi = 1$, then return 0.
- ② If $\varphi = (\neg 0)$, then return 0. If $\varphi = (\neg 1)$, then return 1.
- ③ If $\varphi = X_i$, then return $(\neg X_i)$.
- ④ If $\varphi = (\neg X_i)$, then return X_i .

② Reduction Rules:

- ① If $\varphi = (\neg \varphi_1)$, then return $(\neg \text{DM}(\varphi_1))$.
 - ② If $\varphi = (\varphi_1 \cdot \varphi_2)$, then return $(\text{DM}(\varphi_1) + \text{DM}(\varphi_2))$.
 - ③ If $\varphi = (\varphi_1 + \varphi_2)$, then return $(\text{DM}(\varphi_1) \cdot \text{DM}(\varphi_2))$.
-

Example

$\text{DM}(X \cdot (\neg Y))$.

De Morgan Dual

Exercise

Prove that $DM(\varphi) \in \mathcal{BF}$.

The dual can be obtained by applying replacements to the labels in the parse tree of φ or directly to the “characters” of the string φ .

Theorem

For every Boolean formula φ , $DM(\varphi)$ is logically equivalent to $(\neg\varphi)$.

Corollary

For every Boolean formula φ , $DM(DM(\varphi))$ is logically equivalent to φ .

Nice trick, but is it of any use?!

Negation Normal Form

A formula is in negation normal form if negation is applied only directly to variables or constants. ($\neg 0 = 1$, $\neg 1 = 0$, so we can easily eliminate negations of constants)

Definition

A Boolean formula $\varphi \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$ is in **negation normal form** if the parse tree (G, π) of φ satisfies the following condition. If a vertex v in G is labeled by negation (i.e., $\pi(v) = \neg$), then v is a parent of a leaf.

Example

- $\neg(X_1 + X_2)$ and $(\neg X_1 \cdot \neg X_2)$.
- $\neg(X_1 \cdot \neg X_2)$ and $(\neg X_1 + X_2)$.

Negation Normal Form

Definition

A Boolean formula $\varphi \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR, AND}\})$ is in **negation normal form** if the parse tree (G, π) of φ satisfies the following condition. If a vertex v in G is labeled by negation (i.e., $\pi(v) = \neg$), then v is a parent of a leaf.

Lemma

If φ is in negation normal form, then so is $DM(\varphi)$.

We present an algorithm $NNF(\varphi)$ that transforms a Boolean formula φ into a logically equivalent formula in negation normal form.

Algorithm 4 $\text{NNF}(\varphi)$ - An algorithm for computing the negation normal form of a Boolean formula $\varphi \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$.

- ① Base Cases: If $\varphi \in \{0, 1, X_i, (\neg X_i), \neg 0, \neg 1\}$, then return φ .
 - ② Reduction Rules:
 - ① If $\varphi = (\neg \varphi_1)$, then return $\text{DM}(\text{NNF}(\varphi_1))$.
 - ② If $\varphi = (\varphi_1 \cdot \varphi_2)$, then return $(\text{NNF}(\varphi_1) \cdot \text{NNF}(\varphi_2))$.
 - ③ If $\varphi = (\varphi_1 + \varphi_2)$, then return $(\text{NNF}(\varphi_1) + \text{NNF}(\varphi_2))$.
-

Theorem

Let $\varphi \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$. Then:

(i) $\text{NNF}(\varphi) \in \mathcal{BF}(\{X_1, \dots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$, (ii) $\text{NNF}(\varphi)$ is logically equivalent to φ and, (iii) $\text{NNF}(\varphi)$ is in negation normal form.