Digital Logic Design: a rigorous approach ©

Chapter 6: Propositional Logic

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

April 10, 2012

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.

parse trees

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 $\pi : V \to \{0,1\} \cup U \cup \mathcal{C}$ 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.

example: parse tree

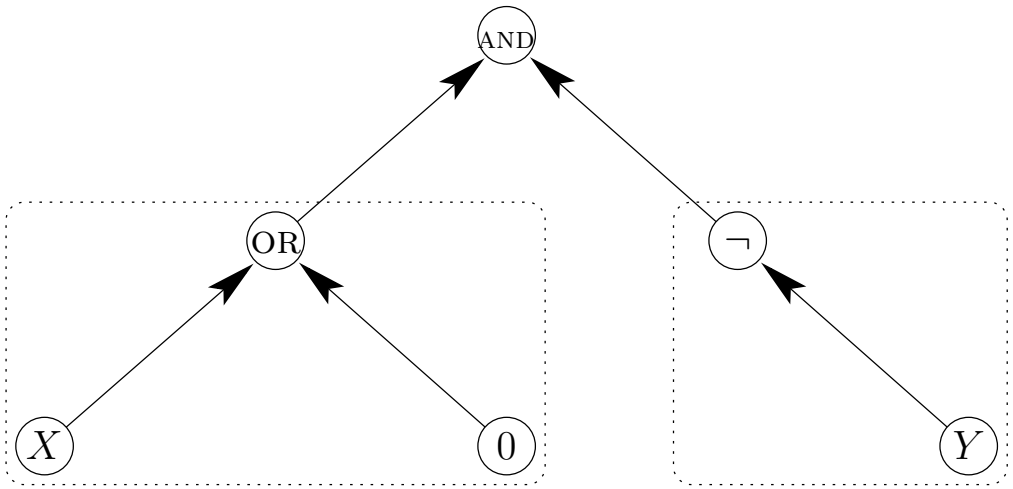

Figure: A parse tree that corresponds to the Boolean formula $((X \text{ OR } 0) \text{ AND } (\neg Y))$. The rooted trees that are hanging from the root of the parse tree (the AND connective) are bordered by dashed rectangles.

Boolean formulas

A Boolean formula is a string containing constants, variables, connectives, and parenthesis. Every parse tree defines a Boolean formula. This definition is constructive and the Boolean formula is obtained by an inorder traversal of the parse tree.
A subformula φ_i of a Boolean formula φ is the Boolean formula that is obtained by an inorder traversal of a subtree in the parse tree.

Examples of Good and Bad Formulas

- $(A \text{ AND } B)$
- $(A \text{ OR } B)$
- $A \text{ OR } \text{ OR } B)$ 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 $deg_{in}(r(G)) = 1$, 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. $\alpha \leftarrow$ INORDER(G_1, π_1).
        4. Return $(\neg\alpha)$.
    2. If $deg_{in}(r(G)) = 2$, 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. $\alpha \leftarrow$ INORDER(G_1, π_1).
        4. $\beta \leftarrow$ INORDER(G_2, π_2).
        5. Return $(\alpha\ \pi(r(G))\ \beta)$.

notation

Let $\mathcal{BF}(U, \mathcal{C})$ denote the set of Boolean formulas over the set of variables U and the set of connectives $\mathcal{C}$. To simplify notation, we abbreviate $\mathcal{BF}(U, \mathcal{C})$ by $\mathcal{BF}$ when the sets of variables and connectives are known.

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

$$
\begin{aligned}
(A + B) &= (A \vee B) = (A \text{ OR } B)\,, \\
(A \cdot B) &= (A \wedge B) = (A \text{ AND } B)\,, \\
(\neg B) &= (\text{NOT}(B)) = (\bar{B})\,, \\
(A \text{ XOR } B) &= (A \oplus B)\,, \\
((A \vee C) \wedge (\neg B)) &= ((A + C) \cdot (\bar{B}))\,.
\end{aligned}
$$

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., $a \cdot b + c \cdot d = ((a \cdot b) + (c \cdot d))$.

Boolean function of a connective

We associate a Boolean function $B_c : \{0,1\}^k \to \{0,1\}$ with each connective $c \in C$ of arity k.
notation:

- denote the function B_{AND} simply by AND, etc.
- The Boolean function associated with negation is NOT.
- The function B_X associated with a variable X is the identity function $I : \{0,1\} \to \{0,1\}$ defined by $I(b) = b$.
- The function B_σ associated with a constant $\sigma \in \{0,1\}$ is the constant function whose value is always σ.

truth assignments

Consider a Boolean formula p generated by a parse tree (G, π). We now show how to evaluate the truth value of p. First, we need to assign truth values to the variables.

Definition

An assignment is a function $\tau : U \to \{0, 1\}$, where U is the set of variables.

Our goal is to extend every assignment $\tau : U \to \{0, 1\}$ to a function that assigns truth values to every Boolean formula over the variables in U.

extending truth assignments to formulas

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

Definition

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

$$\hat{\tau}(p) \stackrel{\triangle}{=} \mathrm{EVAL}(G, \pi, \tau),$$

where EVAL is listed as Algorithm 2.

Algorithm 2 EVAL(G, π, τ) - 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) $\pi : V \to \{0, 1\} \cup U \cup \mathcal{C}$, and (iii) $\tau : U \to \{0, 1\}$ 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 $deg_{in}(r(G)) = 1$, 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. $\sigma \leftarrow$ EVAL(G_1, π_1, τ).
      4. Return $(\text{NOT}(\sigma))$.
   2. If $deg_{in}(r(G)) = 2$, 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. $\sigma_1 \leftarrow$ EVAL(G_1, π_1, τ).
      4. $\sigma_2 \leftarrow$ EVAL(G_2, π_2, τ).
      5. Return $(B_{\pi(r(G))}(\sigma_1, \sigma_2))$.

satisfiability and logical equivalence

Previous: fixed a set of variables U and an assignment $\tau : U \to \{0,1\}$. We then extended τ to every Boolean formula $p \in \mathcal{BF}$ over the variables U.
Now: fix a Boolean formula p over a set U of variables, and consider all possible assignments $\tau : U \to \{0,1\}$.

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

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

$$\varphi \textit{ is satisfiable} \Leftrightarrow (\neg\varphi) \textit{ is not a tautology .}$$

Proof.

The proof is as follows.

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

interpreting a Boolean formula as a function

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

Definition

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

Definition

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

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

Example

$$p = X_1 \vee X_2$$

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

a 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.

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

what about a satisfiable formula?

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.

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

□

equivalent formulas

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

$$\begin{aligned} &\Leftrightarrow \quad \forall\ \tau : \hat{\tau}(p) = \hat{\tau}(q) \\ &\Leftrightarrow \quad \forall\ v \in \{0,1\}^n : \hat{\tau}_v(p) = \hat{\tau}_v(q) \\ &\Leftrightarrow \quad \forall\ v \in \{0,1\}^n : B_p(v) = B_q(v)\,. \end{aligned}$$

Example: Composition of Boolean formulas

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

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

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

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

Proof.

The justifications of all the following lines are by the definition of evaluation:

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

and the lemma follows. □

the implication connective

The implication connective is denoted by $\rightarrow$.

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

$\rightarrow$	0	1
0	1	1
1	0	1

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

more on the implication connective

- $B_{\rightarrow}(X, Y)$ is denoted by $X \rightarrow Y$.
- 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".
- Consider the sentence "if it is raining, then there are clouds". This sentence guarantees clouds if it is raining. If it is not raining, then the sentence trivially holds (regardless of whether there are clouds or not). This explains why $X \rightarrow Y$ is always 1 if $X = 0$.

The connective NAND can be considered to an abbreviation of NOT-AND. Namely, (p NAND q) means (NOT(p AND q)).
Similarly the NOR connective is an abbreviation of NOT-OR. Namely, (p NOR q) means (NOT(p OR q)).
The Boolean functions that correspond to these functions are defined as follows.

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

truth tables

X	Y	X NAND Y
0	0	1
1	0	1
0	1	1
1	1	0

X	Y	X 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 $\leftrightarrow$.

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

The Boolean function that corresponds to equivalence is defined as follows:

$$B_{\leftrightarrow}(A, B) \stackrel{\triangle}{=} (A \rightarrow B) \text{ AND } (B \rightarrow A) .$$

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

$\leftrightarrow$	0	1
0	1	0
1	0	1

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

Claim

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

Proof.

p and q are logically equivalent

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

order matters!

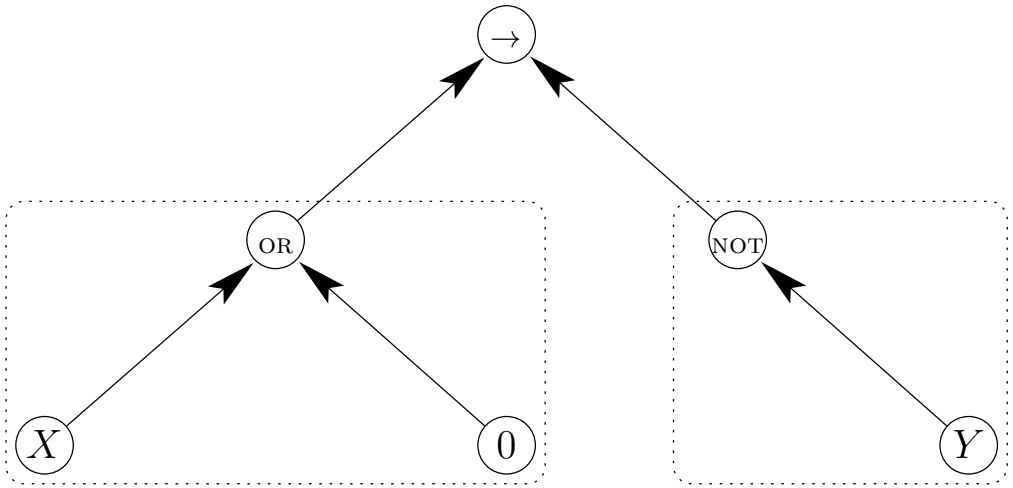

Figure: A parse tree that corresponds to the Boolean formula $((X \text{ OR } 0) \rightarrow (\neg Y))$. The root is labeled by an implication connective. The rooted trees hanging from the root are encapsulated by dashed rectangles.

literals

Definition

A literal is a variable or its negation.

For example, in the Boolean formula $(X \cdot (Y + \bar{X}))$ there are three literals: $X, \bar{X}$, and Y.

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 each 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

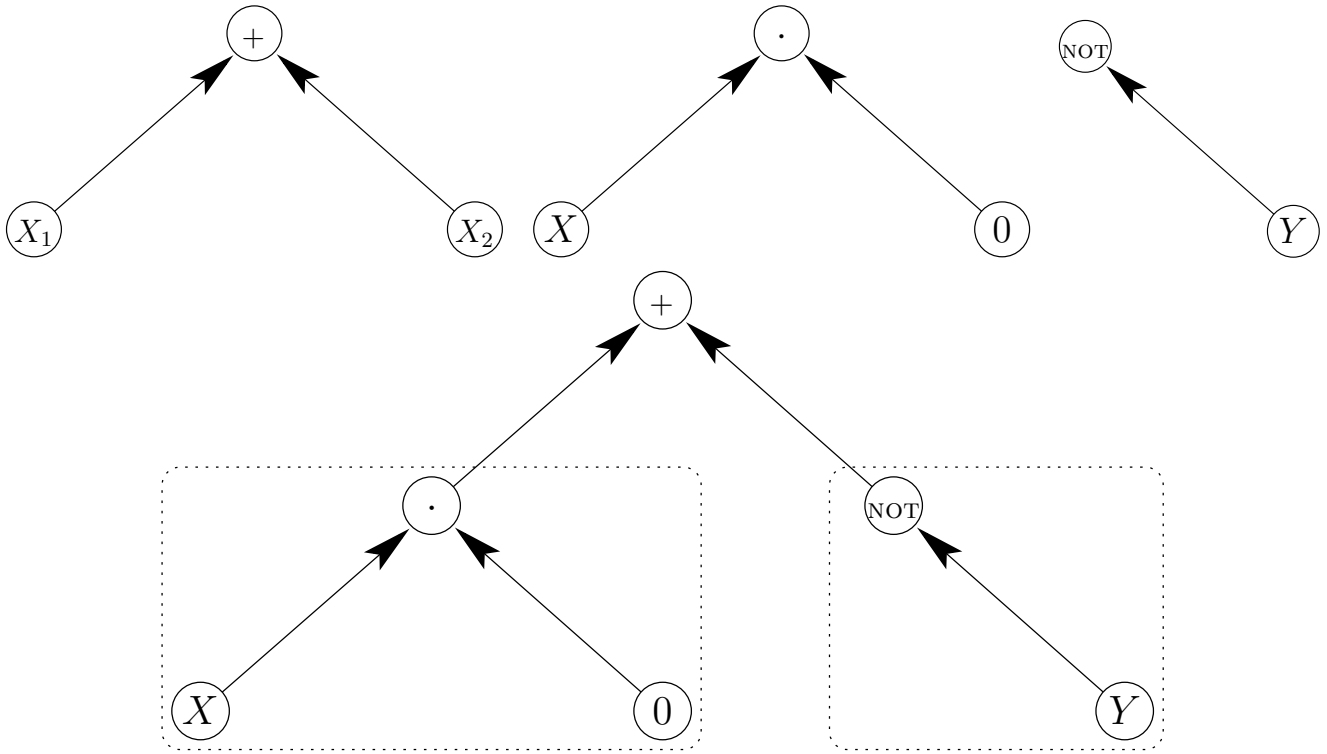

Figure: φ, α_1, α_2, $\varphi(\alpha_1, \alpha_2)$

more on substitution

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

truth values and substitution

The following lemma allows us to treat a formula φ as a Boolean function B_φ. This enables us to evaluate the truth value after substitution (i.e., $\hat{\tau}(\varphi(\alpha_1, \alpha_2))$) using B_φ and the truth values $\hat{\tau}(\alpha_i)$.

Lemma

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

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

Proof hint: induction on the size of the parse tree (G_φ, π_φ).

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

$$\varphi = \neg(X_1 \cdot X_2) \qquad \tilde{\varphi} = \bar{X}_1 + \bar{X}_2$$
$$\alpha_1 = A \to B \qquad \tilde{\alpha}_1 = \bar{A} + B$$
$$\alpha_2 = C \leftrightarrow D \qquad \tilde{\alpha}_2 = \neg(C \oplus D)$$

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

Definition

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

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.

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

$$g(v_1, \ldots, v_{n-1}) \stackrel{\triangle}{=} B(v_1, \ldots, v_{n-1}, 0),$$
$$h(v_1, \ldots, v_{n-1}) \stackrel{\triangle}{=} B(v_1, \ldots, v_{n-1}, 1).$$

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

$$p \stackrel{\triangle}{=} (q \cdot \bar{X}_n) + (r \cdot X_n)$$

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

$$\beta \stackrel{\triangle}{=} (X \cdot Y) + Z.$$

Parse β: $\varphi(\alpha_1, \alpha_2)$ with $\alpha_1 = (X \cdot Y)$ and $\alpha_2 = Z$.
Verify that $\tilde{\varphi} \in \mathcal{BF}(\{X_1, X_2\}, \mathcal{C})$ defined as follows is logically equivalent to $\varphi \stackrel{\triangle}{=} (X_1 + X_2)$.

$$\tilde{\varphi} \stackrel{\triangle}{=} X_1 \oplus X_2 \oplus (X_1 \cdot X_2).$$

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

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

You can check that $\tilde{\varphi}(\alpha_1, \alpha_2)$ is indeed logically equivalent to β.

Theorem: changing connectives

Theorem

If the Boolean functions in $\{\text{NOT}, \text{AND}, \text{OR}\}$ *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 $\beta \in \mathcal{BF}(\{X_1, \ldots, X_n\}, \mathcal{C})$ by a logically equivalent formula $\tilde{\beta} \in \mathcal{BF}(\{X_1, \ldots, X_n\}, \tilde{\mathcal{C}})$.
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 $\tau : U \to \{0,1\}$,

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

question

Let α_1 and α_2 be any Boolean formulas.

1. Consider the Boolean formula $\varphi \stackrel{\triangle}{=} \alpha_1 + \text{NOT}(\alpha_1)$. Prove or refute that φ is a tautology.
2. Consider the Boolean formula $\varphi \stackrel{\triangle}{=} (\alpha_1 \to \alpha_2) \leftrightarrow (\text{NOT}(\alpha_1) + \alpha_2)$. 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 $\text{dual}(\varphi)$?

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

1. Base Cases:
   1. If $\varphi = 0$, then return 1.
   2. If $\varphi = 1$, then return 0.
   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

DM$(X \cdot (\neg Y))$.

De Morgan Dual

Exercise

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

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.

Definition

A Boolean formula $\varphi \in \mathcal{BF}(\{X_1, \ldots, X_n\}, \{\neg, \text{OR}, \text{AND}\})$ is in negation normal form if the parse tree (G, π) of φ satisfies the following condition. If a vertex 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 $(\neg X_1 + X_2)$.

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

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

1. Base Cases: If $\varphi \in \{0, 1, X_i, (\neg X_i)\}$, then return φ.
2. Reduction Rules:
   1. If $\varphi = (\neg\varphi_1)$, then return DM(NNF($\varphi_1$)).
   2. If $\varphi = (\varphi_1 \cdot \varphi_2)$, then return (NNF($\varphi_1$) $\cdot$ NNF(φ_2)).
   3. If $\varphi = (\varphi_1 + \varphi_2)$, then return (NNF($\varphi_1$) $+$ NNF(φ_2)).

Lemma

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

Theorem

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