Digital Logic Design: a rigorous approach ©

Chapter 9: Representation of Boolean Functions by Formulas

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

April 20, 2020

Book Homepage:
http://www.eng.tau.ac.il/~guy/Even-Medina

Normal Forms of Boolean Functions

- A normal form is a restricted syntax for Boolean Formulas.
- For example, Negation Normal Form (NNF) allows negations only of variables or constants.
- We now consider two more normal forms:
  - Disjunctive Normal Form (DNF) also called Sum of Products (SoP)
  - Conjunctive Normal Form (CNF) also called Product of Sums (PoS)
- We will also consider polynomials over a finite field!

Literals

Definition (literal)

A variable or a negation of a variable is called a literal.

Example

- X
- $\textsc{not}(X)$

Product / Conjunction

Recall that:

- $\text{AND}, \cdot, \wedge$ denote the same logical connective.
- Associativity of AND function allows us to omit parenthesis.

Definition (product/conjunction)

A Boolean formula φ is a conjunction (or a product) if

$$\varphi = \ell_1 \text{ AND } \cdots \text{ AND } \ell_k,$$

for $k \geq 1$ and every ℓ_i is a literal.

Example

$$\begin{aligned} X \cdot \bar{Y} \cdot Z &= (X \text{ AND } \bar{Y} \text{ AND } Z) \\ &= (X \wedge \bar{Y} \wedge Z) \end{aligned}$$

Notation

- With each product p, we associate the set of variables that appear in p.
- The set of variables that appear in p is denoted by $vars(p)$.
- Let $vars^+(p)$ denote the set of variables that appear in p that appear without negation.
- Let $vars^-(p)$ denote the set of variables that appear in p that with negation.
- Let $literals(p)$ denote the set of literals that appear in p.
- $p = \bigwedge_{\ell \in literals(p)} \ell = (\bigwedge_{X_i \in vars^+(p)} X_i)$ AND $(\bigwedge_{X_i \in vars^-(p)} \bar{X}_i)$.

Example

Let $p = X_1 \cdot \bar{X}_2 \cdot X_3$, then $vars(p) = \{X_1, X_2, X_3\}$, $vars^+(p) = \{X_1, X_3\}$ and $vars^-(p) = \{X_2\}$, and $literals(p) = \{X_1, \bar{X}_2, X_3\}$.

Simple Products

Definition (simple product)

A product term p is simple if every variable appears at most once in p.

a simple product: $X_1 \cdot X_2 \cdot \bar{X}_3$
not simple: $X \cdot X$, $X_1 \cdot X_2 \cdot \bar{X}_1$
Recall that:

1. $X \cdot \bar{X}$ is a contradiction
2. $X \cdot X$ is logically equivalent to X
3. $\bar{X} \cdot \bar{X}$ is logically equivalent to $\bar{X}$.

Claim

Every product is a contradiction or logically equivalent to a simple product.

exercise!

Minterms

Definition (minterm)

A simple product term p is a minterm with respect to a set U of variables if $vars(p) = U$.

Example

$U = \{X, Y, Z\}$. Minterms: $X \cdot Y \cdot Z$, $\bar{X} \cdot \bar{Y} \cdot Z$.

question

How many different minterms are there with respect to U?

lemma

A minterm p attains the truth value 1 for exactly one truth assignment.

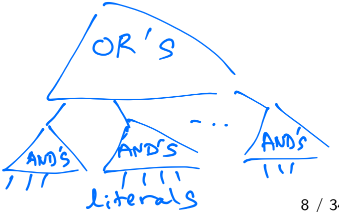
Sum-of-Products (SOP) / Disjunctive Normal Form (DNF)

Definition (SoP/DNF)

A Boolean formula φ is called a sum-of-products (SOP) (or in Disjunctive Normal Form (DNF)) if satisfies one of the following conditions:

1. $\varphi = p_1 + \cdots + p_k$, where $k \geq 2$ and each p_i is a product
2. φ is a product

(the case of a product is a degenerate case for $k = 1$ and includes the case of a single literal.)



Each of the following formulas is a sum-of-products.

1. $\varphi_1 = X \cdot Y + X \cdot Y$,
2. $\varphi_2 = (\bar{A} \text{ AND } B \text{ AND } C) \text{ OR } (A \text{ AND } \bar{B} \text{ AND } C) \text{ OR } \bar{D}$,
3. $\varphi_3 = L$.

Each of the following formulas is not a sum-of-products.

1. $(X + Y) \cdot Z$,
2. $(A \text{ OR } B) \text{ AND } (C \text{ OR } D)$.

SoP representation

Definition

For a $v \in \{0,1\}^n$, define the minterm p_v to be $p_v \triangleq (\ell_1^v \cdot \ell_2^v \cdots \ell_n^v)$, where:

$v = 011$

$$\ell_i^v \triangleq \begin{cases} X_i & \text{if } v_i = 1 \\ \bar{X}_i & \text{if } v_i = 0. \end{cases}$$

$p_v = \bar{x}_1 \cdot x_2 \cdot x_3$

Question

What is the truth assignment that satisfies p_v?

Question

Prove that the mapping $v \mapsto p_v$ is a bijection from $\{0,1\}^n$ to the set of all minterms. (over $\{x_1, \ldots, x_n\}$)

SoP representation - cont (i)

Definition (preimage)

Let $f : \{0,1\}^n \to \{0,1\}$. Let $f^{-1}(1)$ denote the set

$$f^{-1}(1) \triangleq \{v \in \{0,1\}^n \mid f(v) = 1\}.$$

Definition

The set of minterms of f is defined by

$$Min(f) \triangleq \{p_v \mid v \in f^{-1}(1)\}.$$

truth table of $f : \{0,1\}^2 \to \{0,1\}$

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

$f^{-1}(1) = \boxed{\{(0,1), (1,0)\}}$

$$\min(f) = \{ P_{01}, P_{10} \}$$

$$= \{ \bar{x}_1 \cdot x_2, x_1 \cdot \bar{x}_2 \}$$

check that: $\bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2$ expresses f

SoP Representation - cont (ii)

Theorem

Every Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ that is not a constant zero is expressed by the sum of the minterms in $Min(f)$.

Question

Let φ be the sum of the minterms in $Min(f)$ and let τ denote a truth assignment that satisfies φ (i.e., $\hat{\tau}(\varphi) = 1$). How many products in φ are satisfied by τ?

THM: $\sum_{p \in Min(f)} p$ expresses f

proof:

consider $v \in \{0,1\}^n$.

if $f(v) = 1$, then

$p_v \in Min(f)$

p_v satisfied by τ_v $(\hat{\tau}_v(p_v) = 1)$

$\Rightarrow \hat{\tau}_v\left(\sum_{p \in Min(f)} p\right) = 1$

if $f(v) = 0$, then

$\forall p \in Min(f): \hat{\tau}_v(p) = 0$

$\Rightarrow \hat{\tau}_v\left(\sum_{p \in Min(f)} p\right) = 0$ $\boxtimes$

sum-of-minterms: a "bad" example

We are interested in "short" formulas that express a given Boolean function.

- Consider the constant Boolean function $f : \{0,1\}^n \to \{0,1\}$ that is defined by $f(v) = 1$, for every v.
- The sum-of-minterms that represents f is the sum of all the possible minterms over n variables. This sum contains 2^n minterms.
- On the other hand, f can be represented by the constant 1.
- The question of finding the shortest sum-of-products that represents a given Boolean formula is discussed in more detail in our book. (unfortunately, no efficient optimization is believed to exist)

Product of Sums (PoS)/ Conjunctive Normal Form (CNF)

The second normal form we consider is called conjunctive normal form (CNF) or product of sums (POS).

Sum / Disjunction

Recall that:

- $\textsc{or}, +, \vee$ denote the same logical connective.
- Associativity of $\textsc{or}$ function allows us to omit parenthesis.

Definition

A Boolean formula s is a disjunction (or a sum) if

$$s = \ell_1 + \cdots + \ell_k,$$

for $k \geq 1$ and every ℓ_i is a literal.

Example

$$\begin{aligned} X + \bar{Y} + Z &= (X \ \textsc{or} \ \bar{Y} \ \textsc{or} \ Z) \\ &= (X \ \vee \ \bar{Y} \ \vee \ Z) \end{aligned}$$

Define $\mathit{vars}(s), \mathit{vars}^{+}(s), \mathit{vars}^{-}(s), \mathit{literals}(s)$ as in products.

Maxterms

Definition (simple sum)

A sum s is simple if every variable appears at most once in s.

Definition (maxterm)

A simple sum term s is a maxterm with respect to a set U of variables if $vars(s) = U$.

Question

How many maxterms are there with respect to U?

Lemma

A maxterm s is satisfied by all but one truth assignment (s attains the truth value 0 for exactly one truth assignment).

$X + \bar{Y} + Z$ $\quad \tau(X) = 0 \quad \tau(Y) = 1 \quad \tau(Z) = 0$

Product-of-Sums (PoS) / Conjunctive Normal Form (CNF)

Definition (SoP/DNF)

A Boolean formula φ is called a product-of-sums (POS) (or in Conjunctive Normal Form (CNF)) if satisfies one of the following conditions:

AND'S

1. $\varphi = s_1 \cdot \cdots \cdot s_k$, where $k \geq 2$ and each s_i is a sum
2. φ is a sum

(the case of a sum is a degenerate case for $k = 1$ and includes the case of a single literal.)

Recall that $DM(\varphi)$ is the De Morgan dual of the formula φ.

observation

1. If p is a product, then $DM(p)$ is a sum.
2. If s is a sum, then $DM(s)$ is a product.
3. If p is a minterm, then $DM(p)$ is a maxterm.
4. If s is a maxterm, then $DM(s)$ is a minterm.
5. If p is a sum-of-products, then the formula $DM(p)$ is a product-of-sums.
6. If p is a product-of-sums, then the formula $DM(p)$ is a sum-of-products.

$$x_1 \cdot x_2 \cdot x_3 \xmapsto{DM} \bar{x}_1 + \bar{x}_2 + \bar{x}_3$$

Maxterms of a Boolean Function

Definition

For a $v \in \{0,1\}^n$, define the maxterm s_v to be
$s_v \triangleq (m_1^v + \cdots + m_n^v)$, where:

$$m_i^v \triangleq \begin{cases} X_i & \text{if } v_i = 0 \\ \bar{X}_i & \text{if } v_i = 1. \end{cases}$$

Note that ℓ_i^v is logically equivalent to $\text{NOT}(m_i^v)$.

Question

Which truth assignment does not satisfy s_v?

$\hat{\tau}(m_i^v) = 0$: if $v_i = 0$: $\hat{\tau}(X_i) = 0$

if $v_i = 1$: $\hat{\tau}(\bar{X}_i) = 0$

$\hat{\tau}(X_i) = 1$

PoS representation of Boolean Functions

Definition (Maxterms of a function $f : \{0,1\}^n \to \{0,1\}$)

$$Max(f) \triangleq \{s_v \mid v \in f^{-1}(0)\} .$$

Theorem

Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ that is not a constant one is expressed by the product of the maxterms in $Max(f)$.

De Morgan Duality and CNF representation

Question

What is the relation between $Min(f)$ and $Max(\textsc{not}(f))$?

Let $U = \{X_1, \ldots, X_n\}$ and $f : \{0,1\}^n \to \{0,1\}$.

Lemma

Let p denote a minterm wrt U. Then,

$$p \in Min(f) \iff DM(p) \in Max(\textsc{not}(f))$$

Let s denote a maxterm wrt U. Then,

$$s \in Max(f) \iff DM(s) \in Min(\textsc{not}(f))$$

Theorem

$CNF(f) = DM(DNF(\textsc{not}(f)))$

Lemma: $p \in Min(f) \implies DM(p) \in Max(\bar{f})$

proof: ($\implies$) (converse exercise)

$p \in Min(f)$ means that

$\exists v \in \{0,1\}^n$: $p = p_v$ & $f(v) = 1$

but $DM(p_v) = s_v$ & $\bar{f}(v) = 0$

$\implies DM(p) \in Max(\bar{f})$

$\boxtimes$

Lemma: $s \in Max(f) \implies DM(s) \in Min(f)$

exercise!

THM: $\mathrm{CNF}(f) = \mathrm{DM}(\mathrm{DNF}(\bar{f}))$

Proof: (sketch)

$$\begin{array}{ccccc} f & \xrightarrow{\text{not}} & \bar{f} & & \\ & & \cong & & \\ \sum_{v \in \bar{f}^{-1}(1)} p_v & \rightsquigarrow & \mathrm{DNF}(\bar{f}) & \xrightarrow[\mathrm{DM}]{\text{not}} & \mathrm{DM}(\mathrm{DNF}(\bar{f})) \end{array}$$

2 negations $\Longrightarrow$

$$f \cong \underbrace{\mathrm{DM}(\mathrm{DNF}(\bar{f}))}_{\text{a CNF formula.}}$$

Representation by polynomials

Definition

The Galois Field $GF(2)$ is defined as follows.

1. Elements: the elements of $GF(2)$ are $\{0, 1\}$. The zero is called the additive unity and one is called the multiplicative unity.
2. Operations:
    1. addition which is simply the XOR function, and
    2. multiplication which is simply the AND function.

In the context of $GF(2)$ we denote multiplication by $\cdot$ and addition by $\oplus$.

$GF(2)$ properties

We are used to infinite fields like the rationals (or reals) with regular addition and multiplication. In these fields, $1 + 1 \neq 0$. However, in $GF(2)$, $1 \oplus 1 = 0$.

Observation

$X \oplus X = 0$, *for every* $X \in \{0, 1\}$.

A minus sign in a field means the additive inverse.

Definition

The element $-X$ stands for the element Y such that $X \oplus Y = 0$.

$GF(2)$ properties - more

Observation

In $GF(2)$, the additive inverse of X is X itself, namely $-X = X$, for every $X \in \{0, 1\}$.

Thus, we need not write minus signs, and adding an X is equivalent to subtracting an X.

The distributive law holds in $GF(2)$, namely:

Observation

$(X \oplus Y) \cdot Z = (X \cdot Z) \oplus (Y \cdot Z)$, *for every* $X, Y, Z \in \{0, 1\}$.

Let X^k denote the product (AND of literals)

$$X^k \triangleq \overbrace{X \cdot \dots \cdot X}^{k \text{ times}}.$$

We define $X^0 = 1$, for every $X \in \{0,1\}$. The following observation proves that multiplication is idempotent.

Observation

$X^k = X$, for every $k \in \mathbb{N}^+$ and $X \in \{0,1\}$.

$GF(\cancel{w}2)$ is a field like the reals

The structure of a field allows us to solve systems of equations. In fact, Gauss elimination works over any field. The definition of a vector space over $GF(2)$ is just like the definition of vector spaces over the reals. Definitions such as linear dependence, dimension of vector spaces, and even determinants apply also to vector spaces over $GF(2)$.

$$x_1 \quad \overbrace{x_1 \oplus \overbrace{x_2 \oplus \overbrace{(-x_2)}^{=x_2}}^{0}} = -x_2 = x_2$$

- $$X_1 \oplus X_2 = 0 \quad \Leftrightarrow \quad X_1 = X_2 .$$

- We show how to solve a simple systems of equalities over $GF(2)$ using Gauss elimination. Consider the following system of equations

$$\begin{array}{lccccccc} & \overset{1}{\overset{\shortparallel}{X_1}} & \oplus & \overset{0}{X_2} & \oplus & \overset{1}{X_3} & = 0\,, \\ 1) & X_1 & \oplus & X_2 & \oplus & X_3 & = 0\,, \\ 2) & X_1 & & & \oplus & X_3 & = 0\,, \\ 3) & & & X_2 & \oplus & X_3 & = 1\,. \end{array}$$

$(1+2)$ $\boxed{x_2 = 0}$

$3) \Rightarrow$ $0 \oplus x_3 = 1 \Rightarrow \boxed{x_3 = 1}$

Polynomials over $GF(2)$

Definition

A monomial in $GF(2)$ over the variables in the set U is a finite product of the elements in U or a constant in $\{0, 1\}$.

Observation

Every monomial p in $GF(2)$ over the variables in U equals a constant or a simple product of variables in p.

- By commutativity: $X_1 \cdot X_2 \cdot X_3 \cdot X_1 = X_1^2 \cdot X_2 \cdot X_3$.
- Positive exponents can be reduced to one. For example, $X_1^2 \cdot X_2 \cdot X_3$ equals $X_1 \cdot X_2 \cdot X_3$.

Polynomials

Definition

A polynomial in $GF(2)$ over the variables in the set U is a finite sum of monomials.

Example: $X_1 \cdot X_2 \oplus X_1 \cdot X_3 \oplus X_2 \cdot X_3 \oplus 1$.

We denote the set of all polynomials in $GF(2)$ over the variables in U by $GF(2)[U]$. Just as multivariate polynomials over the reals can be added and multiplied, so can polynomials in $GF(2)[U]$.

representation by polynomials in $GF(2)[U]$

Every polynomial $p \in GF(2)[U]$ is a Boolean function $f_p : \{0,1\}^{|U|} \rightarrow \{0,1\}$. The converse is also true.

$p(x) \neq \sin x$
$p(x) \neq e^x$
....

Theorem

Every Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be represented by a polynomial in $GF(2)[U]$, where $U = \{X_1, \ldots, X_n\}$.

proof outline

- easy: f is constant.
- $f^{-1}(1) \triangleq \{v \in \{0,1\}^n \mid f(v) = 1\}$.
- For each $v \in f^{-1}(1)$, we define the product p_v. The polynomial $p \in GF(2)[U]$ is defined as follows.

$x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot x_4 = \underline{x_1 \cdot (1 \oplus x_2) \cdot (1 \oplus x_3) \cdot x_4}$

not a monomial!

$$p \triangleq \overset{\text{XOR}}{\bigoplus_{v \in f^{-1}(1)}} p_v .$$

$OR(1,1) = 1$
$XOR(1,1) = 0$
$OR(0,\ldots,0,1) = 1$
$XOR(0,0,\ldots,0,1) = 1$

$DNF(f) = \overset{\text{OR}}{\sum_{v \in f^{-1}(1)}} P_v$

why can we replace OR by XOR?

Side effect

$\forall f \quad \exists$ poly $p \in GF(2)[u]: \quad p = f$

Corollary

The set of connectives $\{\text{XOR}, \text{AND}\}$ *is complete.*

Satisfiability

The problem of satisfiability of Boolean formulas is defined as follows.

$(x \cdot y \vee z) \cdot w \vee (\bar{x})$

(string / parse tree)

Input: A Boolean formula φ.

Output: The output should equal "yes" if φ is satisfiable. If φ is not satisfiable, then the output should equal "no".

Note that the problem of satisfiability is quite different if the input is a truth table of a Boolean function. In this case, we simply need to check if there is an entry in which the function attains the value 1.

$\varphi \rightarrow \boxed{\text{SAT}} \rightarrow$ yes/no

yes $\Leftrightarrow$ $\exists$ truth assign τ

$\hat{\tau}(\varphi) = 1$

Relation to P vs. NP

The main open problem in Computer Science since 1971 is whether $P = NP$. We will not define the classes P and NP, but we will phrase an equivalent question in this section.

Consider a Boolean formula φ. Given a truth assignment τ, it is easy to check if $\hat{\tau}(\varphi) = 1$. We showed how this can be done in Algorithm EVAL. In fact, the running time of the EVAL algorithm is linear in the length of φ.

On the other hand, can we find a satisfying truth assignment by ourselves (rather than check if τ is a satisfying assignment)? Clearly, we could try all possible truth assignments. However, if n variables appear in φ, then the number of truth assignments is 2^n.

Satisfiability and P vs. NP

We are ready to formulate a question that is equivalent to the question $P = NP$.

Satisfiability in polynomial time

Does there exist a constant $c > 0$ and an algorithm Alg such that:

1. Given a Boolean formula φ, algorithm Alg decides correctly whether φ is satisfiable.
2. The running time of Alg is $O(|\varphi|^c)$, where $|\varphi|$ denotes the length of φ.

This seemingly simple question turns out to be a very deep problem about what can be easily computed versus what can be easily proved. It is related to the question whether there is a real gap between checking that a proof is correct and finding a proof.