Boolean Algebra, Identities, Proofs, and Minimization

Algorithms, Programming, and Data Representation · Module 92313 · Chapter 12a

From Fundamentals to AI Apocalypse · personal study guide

Contents

Start from memory 2

How to use this guide 2

How the ideas fit together 3

1.1 Foundations: one structure, several views 4

1.2 Core identities and proof tools 7

1.3 Completeness, normal forms, and minimisation 9

1.4 Karnaugh maps remove changing variables 11

1.5 Shannon views, OBDDs, and algebraic normal form 13

1.6 Verification, hazards, and wrap-up 15

Formula map 16

Derive it cold 17

Explain it 17

Mixed practice 18

Exam-style problems 19

Slips in the notes 20

Self-check 21

Answers 22

Made by Thanate Amornjiraporn (Focus)

This guide develops Boolean logic through algebraic structures, proof tools, function minimisation, and decision diagrams.

Start from memory

Answer before reading. Mark each response as sure, unsure, or a guess; uncertainty shows what to revisit.

1. Name the three operations and two constants in a Boolean algebra.
2. What is the dual of $x + 0 = x$?
3. State one absorption law.
4. What two cases does Shannon expansion split on?
5. How many literals remain in a K-map group of 8 cells for a 5-variable function?

Answers: see "Start from memory" under Answers at the end.

How to use this guide

Study one short section, close it, then reconstruct its laws or argument from memory. Mix proof, algebra, and diagram questions so you practise choosing the right tool.

1. **Day 0:** Try the cold start, then study §§1.1–1.2 and derive duality and De Morgan's laws.
2. **Day 1:** Rebuild the NAND proof and normal forms in §1.3; solve mixed practice 1–8.
3. **Day 3:** Draw K-map groupings from memory and solve practice 9–14.
4. **About day 7:** Reconstruct Shannon expansion, cofactors, and OBDD reduction; solve practice 15–26.
5. **About day 14:** Complete both exam-style problems without notes and compare them with the slips in the notes.
6. **Before the exam:** Tick only skills you can show cold; return to any section referenced by a missed answer.

When stuck: name the definition, write one transformation at a time, and check the result on a small truth table. A guess counts as uncertain until you verify it.

Made for Thanadol Anonjitaporn (Focus)

How the ideas fit together

The same Boolean functions can be viewed as algebra, sets, rings, normal forms, maps, or decision diagrams. Each view makes a different task easier.

Assumed · the axioms

- **Boolean algebra** §1.1

  join · meet · complement

  Rests on: nothing; these are the axioms. They support equivalent algebraic and lattice views.

Defined · choose a form

- **Complete connectives** §1.3

  NAND or NOR alone

  Rests on: the axioms. A complete set can express every Boolean function.

- **Karnaugh maps** §1.4

  adjacent cells → fewer literals

  Rests on: normal forms. Gray-code adjacency exposes variables that can be eliminated.

Derived · prove and split

- **Identities and proofs** §1.2

  duality · De Morgan · absorption

  Rests on: the axioms. Derived laws shorten proofs and expressions.

- **Shannon expansion** §1.5

  $f = x\, f_1 + x'\, f_0$

  Rests on: the identities. Cofactors lead to decision diagrams and dependency tests.

- **Verification and hazards** §1.6

  same function · safe transitions

  Rests on: minimised forms and cofactors. Logical equivalence and physical timing answer different questions.

Question	Useful view
Can two formulas be shown equal?	Use axioms, derived identities, or a truth table.
Can a gate set express every function?	Show it can express NOT, AND, and OR.
Can a function be represented more compactly?	Try a normal form, a K-map, or a reduced OBDD.

Made by Thanate Amornjiraporn (Focus)

1.1 Foundations: one structure, several views

A Boolean algebra packages truth operations into a system of equations. The same structure can also be understood through order, sets, or arithmetic modulo two.

> **In plain words:** a Boolean algebra is any system with two operations that behave like OR and AND, a complement that behaves like NOT, and the constants 0 and 1, all obeying the same short list of rules. Logic, sets (with $\cup$, $\cap$ and complement) and switching circuits are all examples, so a law proved once holds in each of them.

Made by Thanate Amornjiraporn (Focus)

Operations and axioms

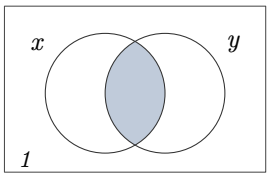
Write the structure as (B, +, ·, ′, 0, 1). Join (+) and meet (·) are binary operations; complement (′) is unary. The Huntington-style axioms used in the lecture state commutativity, associativity, distributivity, identities, and complements.

Core axioms

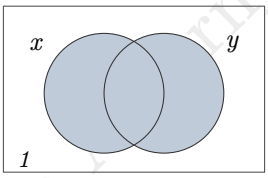
$$x + y = y + x$$
$$(x + y) + z = x + (y + z)$$
$$x \cdot (y + z) = x \cdot y + x \cdot z$$
$$x + y \cdot z = (x + y) \cdot (x + z)$$
$$x + 0 = x$$
$$x \cdot 1 = x$$
$$x + x' = 1$$
$$x \cdot x' = 0$$

Read it as: "x or y equals y or x", and so on down the list; the last two say "x or not x is 1" and "x and not x is 0".

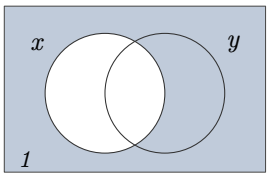
The lecture's axiom list also states the dual commutative and associative laws for meet. It prints only the first distributive law; the second is needed, because XOR and AND on {0,1} satisfy the printed list but not x+x=x.



x · y: in the set view, meet is intersection.



x + y: join is union.



x′: the complement is everything in 1 outside x.

Duality swaps + with · and 0 with 1. The axiom list is self-dual, so every equation derivable from the axioms has a derivable dual. For example, x + x = (x + x)·1 = (x + x)·(x + x′) = x + x·x′ = x uses the second distributive law, and its dual is x·x = x. A **distributive complemented bounded lattice** is the same structure in order language: join and meet have bottom 0 and top 1, and each element has a complement.

> **Worked proof · complements are unique**
> Suppose y behaves like a complement of x: x+y=1 and x·y=0. Show y=x′.
>
> 1. Use the identity and complement of x: y=y·1=y·(x+x′).
> 2. Distribute: y=(y·x)+(y·x′)=0+(y·x′)=y·x′.
> 3. Similarly x′=x′·1=x′·(x+y)=x′·y. Therefore y=y·x′=x′·y=x′. §1.1
>
> The proof relies on the stated complement conditions; it does not assume uniqueness in advance.

Sets, rings, and maps

The power set P(U), with union as +, intersection as ·, set complement as ′, empty set as 0, and U as 1, is a Boolean algebra. There is also a ring view: XOR is addition, AND is multiplication, and every element is idempotent under multiplication. Formula cards use a circled plus for XOR, because slide 13 reuses + for it.

Boolean ring translation

$$x \oplus y = (x \cdot y') + (x' \cdot y)$$
$$x \odot y = x \cdot y$$
$$x \wedge y = xy$$
$$x \vee y = x \oplus y \oplus xy$$
$$\neg x = 1 \oplus x$$
$$x^2 = x$$
$$x \oplus x = 0$$

Read it as: "x XOR y is (x and not y) or (not x and y)", true when exactly one of them is 1; and "x squared is x", because $x \cdot x = x$ for both 0 and 1.

Slide 13 reuses + for ring addition; here XOR keeps it apart from join. The power-set example uses symmetric difference for ring addition and intersection for multiplication.

A homomorphism h preserves the operations and constants. Its kernel is ker h = $h^{-1}(0)$, and its image is a subalgebra; when it is onto, the lecture's isomorphism theorem identifies the quotient by its kernel with the target algebra.

Term	Meaning
Order x≤y	x·y=x, or equally x+y=y; on {0,1}, x implies y
Lattice view	Set x∨y=x+y and x∧y=x·y; the axioms give a distributive complemented bounded lattice, and any such lattice gives back +, · and ′ satisfying the axioms
Characteristic 2	x XOR x=0, so −x=x: every element is its own negative
Finite Boolean rings	Each is isomorphic to F_2^n, computed componentwise

Trap: duality is a rule for transforming a proved equation; it does not mean complement every variable or reverse every term.

Near miss: the dual of $x + 0 = x$ is $x \cdot 1 = x$: swap + with ·, and 0 with 1. Swapping only the operator gives $x \cdot 0 = x$, which is false for $x = 1$. Duality swaps the constants as well.

Recall check · answer without looking

1. Which operations and constants appear in the Boolean algebra structure?
2. How does the ring view interpret XOR and AND?

Answers: see "1.1 Foundations: one structure, several views" under Answers at the end.

Study Guide by Thanate Amornpiyakul

1.2 Core identities and proof tools

Axioms are the starting point; identities are reusable consequences. Proofs become shorter when you name the law that justifies each step.

> **In plain words:** the identities here are the tools you reach for in every proof, so learn them by name. Two of them ask how a function depends on one input: the **difference** $\partial f/\partial x$ is 1 exactly where flipping x changes f, and the quantifiers $(\exists x)f$ and $(\forall x)f$ ask whether f is 1 for some, or for every, value of x.

Law	Identity
De Morgan	$(x+y)'=x'\cdot y'$ $(x\cdot y)'=x'+y'$
Absorption	$x+x\cdot y=x$ $x\cdot(x+y)=x$
Consensus	$x\cdot y+x'\cdot z+y\cdot z=x\cdot y+x'\cdot z$
Shannon expansion	$f=x\cdot f_{x=1}+x'\cdot f_{x=0}$

Worked proof · De Morgan's first law

Show that $x'\cdot y'$ is the complement of $x+y$.

1. Join: $(x+y)+x'y'=(x+y+x')(x+y+y')=1\cdot 1=1$, by the second distributive law.
2. Meet: $(x+y)\cdot x'y'=x\cdot x'\cdot y'+y\cdot x'\cdot y'=0+0=0$, by the first distributive law.
3. Uniqueness of complements from §1.1 gives $(x+y)'=x'y'$. The second law follows by duality. §1.2

Worked simplification · remove a redundant condition

Simplify $x+x\cdot y$ using only laws from the lecture.

1. Factor x with distributivity: $x\cdot 1+x\cdot y=x\cdot(1+y)$.
2. Since $1+y=(y'+y)+y=y'+(y+y)=y'+y=1$, using idempotence from §1.1, this becomes $x\cdot 1$.
3. Use the identity law: $x\cdot 1=x$. Therefore $x+x\cdot y=x$. §1.2

Read the result as: once x is already included, the stricter case x and y adds no new true assignments.

Made by (Focus)

Splitting a function into cases

The cofactor $f_{x=1}$ fixes x to 1; $f_{x=0}$ fixes it to 0. The Boolean difference is the XOR of the cofactors, so it is 1 exactly where changing x changes the output.

Derivation · Shannon expansion

1. Insert the complement law: $f=f\cdot 1=f\cdot(x+x')$.
2. Distribute: $f=x\cdot f+x'\cdot f$.
3. Where $x=1$ only $f_{x=1}$ matters, so $x\cdot f=x\cdot f_{x=1}$; likewise $x'\cdot f=x'\cdot f_{x=0}$. Therefore $f=x\cdot f_{x=1}+x'\cdot f_{x=0}$. §1.2

Difference and quantification

$$\partial f/\partial x = f_{x=1} \oplus f_{x=0}$$
$$\partial(f \oplus g)/\partial x = \partial f/\partial x \oplus \partial g/\partial x$$
$$\partial(fg)/\partial x = f\cdot\partial g/\partial x \oplus g\cdot\partial f/\partial x \oplus \partial f/\partial x\cdot\partial g/\partial x$$
$$(\exists x)f = f_{x=1} + f_{x=0}$$
$$(\forall x)f = f_{x=1}\cdot f_{x=0}$$
$$(\forall x)f \le f \le (\exists x)f$$

Read it as: "the difference of f in x is f with x set to 1, XOR f with x set to 0"; "there exists x: f" is the OR of the two cofactors, and "for all x: f" is their AND.

f does not depend on x exactly when its difference in x is 0. Existential quantification (smoothing) asks whether either case works; universal quantification (consensus) asks whether both cases work. Both remove dependence on x.

Worked example · difference and quantifiers

Let $f=x+y$.

1. Cofactors: $f_{x=1}=1$ and $f_{x=0}=y$.
2. Difference in x: 1 XOR $y=y'$. Changing x matters only when $y=0$.
3. Quantifiers: $(\exists x)f=1+y=1$ and $(\forall x)f=1\cdot y=y$. Check: $y\le x+y\le 1$. §1.2

Memory hook: for existential, OR the two cofactors; for universal, AND them. Check the question's quantifier before simplifying.

Near miss: for $f = x\cdot y$, $\partial f/\partial x = (1\cdot y)\oplus(0\cdot y) = y$, not 1: f depends on x only when $y = 1$. A formula can contain x and still ignore it for some inputs.

Recall check · answer without looking

1. Why can the complement uniqueness theorem prove De Morgan's first law?
2. What does a Boolean difference of 0 tell you about x?

Answers: see "1.2 Core identities and proof tools" under Answers at the end.

1.3 Completeness, normal forms, and minimisation

A representation tells you how to build a function. Completeness says whether a chosen set of connectives can represent every function; minimisation asks how small a representation can be.

> **In plain words:** a set of connectives is **complete** if every Boolean function can be written with them alone. NOT, AND and OR are complete, and so is NAND on its own, because it can rebuild all three. **Minimisation** is a separate question: of all the formulas for a function, which is the smallest?

Build every function from a small gate set

A connective set is functionally complete if it can express every Boolean function. NAND alone is complete because it can express NOT, AND, and OR. NOR alone is complete by the dual argument.

NAND construction

$$x \uparrow y = (x \cdot y)'$$
$$\neg x = x \uparrow x$$
$$x \cdot y = (x \uparrow y) \uparrow (x \uparrow y)$$
$$x + y = (x \uparrow x) \uparrow (y \uparrow y)$$

Read it as: "x NAND y is not (x and y)", and "not x is x NAND x".
Once NOT, AND, and OR are available, any formula built from those connectives can be expressed with NAND.

Post's lattice classifies families of Boolean functions closed under composition. The lecture also contrasts monotone functions, which preserve order, with affine functions, which are parity expressions over F_2; a function that is both monotone and affine is a constant or a single variable. (The lecture says only "constant", but f=x is a counterexample; see the slips.) Any XOR of two or more inputs falls from 1 to 0 when a second input turns on.

Made by Thanate

Write and minimise formulas

Every Boolean function has a disjunctive normal form (OR of minterms for true rows) and a conjunctive normal form (AND of maxterms that exclude false rows). These forms are systematic, though finding a minimum two-level form is hard (deciding this from a truth table is NP-complete, as the lecture states), so practical methods use structure and heuristics.

Worked example · XOR as DNF and CNF

Let f(A,B)=1 on rows 01 and 10, and 0 on rows 00 and 11.

1. For the true rows, the minterms are A′B and AB′, so the DNF is A′B+AB′.
2. The false rows 00 and 11 give maxterms (A+B) and (A′+B′), so the CNF is (A+B)(A′+B′).
3. Both forms describe XOR: the output is 1 exactly when the inputs differ. §1.3

Quine–McCluskey combines minterms that differ in one literal, repeats until prime implicants remain, then selects a cover. Its steps preserve coverage: every combined implicant covers only true assignments, and every listed true minterm remains covered.

Worked example · Quine–McCluskey on minterms 1, 3, 5, 7

1. Write the rows in A,B,C order: 001, 011, 101, 111.
2. Combine pairs differing in one bit: 0−1, −01, −11, 1−1.
3. Combine again: 0−1 with 1−1 gives −−1, and −01 with −11 gives the same cube.
4. The only prime implicant is −−1, so f=C. Check: C is 1 on exactly the four odd rows. §1.3

Trap: a complete gate set guarantees expressiveness, not a small or fast circuit.

Near miss: {AND, OR} without NOT looks almost complete, but it cannot produce NOT x: a formula built from AND and OR alone outputs 0 when every input is 0, whereas NOT x gives 1 there.

Recall check · answer without looking

1. How does NAND express NOT, AND, and OR?
2. What does DNF list, and why is finding a minimum exact form difficult?

Answers: see "1.3 Completeness, normal forms, and minimisation" under Answers at the end.

1.4 Karnaugh maps remove changing variables

A K-map lays truth-table rows out so neighbouring cells differ in one variable. Grouping adjacent 1s makes that changing variable disappear from the corresponding implicant.

> **In plain words:** a **Karnaugh map** is a truth table folded into a grid in which each cell differs from its neighbours in exactly one variable. Grouping a block of neighbouring 1s makes the variables that change inside the block drop out, which leaves a shorter product term.

For n variables, a map has 2^n cells in Gray-code order. A legal group is a rectangle of 2^k cells (possibly wrapping), so that it contains every combination of k changing variables. Each variable that changes across the group is eliminated; the implicant keeps n−k literals. Use don't-care rows as either 0 or 1 when that makes a useful group.

K-map group rule

$$\text{Map size}: 2^n \text{ cells}$$
$$\text{Group size}: 2^k \text{ cells}$$
$$\text{Remaining literals}: n-k$$

Read it as: "a map for n variables has 2 to the n cells; a group of 2 to the k cells leaves $n-k$ literals".
Cover every required 1 at least once. Do not include a required 0. A don't-care can be included or ignored.

A / BC	00	01	11	10
0	0	0	1	1
1	0	0	1	1

A group of four 1s in the columns BC = 11 and 10: A and C change inside it and drop out, leaving the single literal B.

A / BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1

Wrap-around: the columns 10 and 00 are neighbours (only B differs), so these four 1s form one group, the literal C′.

> **Worked example · simplify using an explicit bit order**
> Use variable order A,B,C with A as the most significant bit. Suppose f=1 for minterms 2, 3, 6, and 7.
>
> 1. These rows are 010, 011, 110, and 111, so B=1 in every true row.
> 2. On the map, they form a group of four. A and C vary; B stays fixed.
> 3. The group leaves one literal: f=B. Verify that B=0 rows map to 0 and B=1 rows map to 1. §1.4

Trap: write the variable bit order beside minterm indices. A minterm number has no meaning without a variable order.

Near miss: three adjacent 1s look like a natural block, but a group must have 1, 2, 4, 8 … cells; with three cells no variable drops out. Cover them with two overlapping pairs instead.

Recall check · answer without looking

1. How many literals remain after grouping 8 cells on a 5-variable map?
2. What is the safe way to use a don't-care cell?

Answers: see "1.4 Karnaugh maps remove changing variables" under Answers at the end.

Made by Thanate Amornjiraporn (Focus)

1.5 Shannon views, OBDDs, and algebraic normal form

Shannon expansion recursively splits a function into simpler cases. Reusing the same split structure leads to decision diagrams; a different basis expresses the function as parity over F_2.

> **In plain words: Shannon expansion** splits a function on one input: what is f when $x = 1$, and what is it when $x = 0$? Repeating the split gives a decision tree, and merging identical branches gives a compact decision diagram, an OBDD. Algebraic normal form writes the same function as an XOR of AND terms instead.

Shannon expansion

$$f = x \cdot f_{x=1} + x' \cdot f_{x=0}$$

Read it as: "f equals x and (f with x = 1), or not x and (f with x = 0)": x selects which cofactor passes through. The selected input x chooses between the two cofactors, like a 2-to-1 multiplexer.

Worked example · expand on A

For f(A,B)=A·B+A′, compute the two cofactors and reconstruct f.

1. Set A=1: $f_{A=1}$=B+0=B. Set A=0: $f_{A=0}$=0+1=1.
2. Substitute: f=A·B+A′·1.
3. The expansion matches the starting expression and separates the A=1 and A=0 cases. §1.5

An ordered binary decision diagram (OBDD) applies Shannon expansion in a fixed variable order. Reduction merges isomorphic subgraphs and removes tests whose two branches are identical. For one fixed order, the reduced OBDD is unique; a different variable order can produce a different diagram size.

The algebraic normal form (ANF) writes f as XORs of monomials over F_2. Every Boolean function has a unique ANF; its coefficients can be computed by a Möbius transform. Uniqueness follows because the monomials form a basis for all truth functions on n variables. Unlike DNF, ANF uses XOR as its addition operation.

Worked example · OBDD for A?B:C with order A,B,C

1. Split on A: the 1-branch is B and the 0-branch is C.
2. Split the B branch: B=1 gives terminal 1, B=0 gives terminal 0. The C branch is the same shape.
3. Each test has different branches, and the B and C nodes test different variables, so nothing reduces further: 3 nonterminal nodes.
4. Check: with order B,C,A the same function needs 5 nodes, so order matters. §1.5

Worked example · Möbius transform of NAND

Outputs for rows 00, 01, 10, 11 are 1,1,1,0, with A as the first bit.

1. B pass: XOR each B=1 entry with its B=0 partner, giving 1,0,1,1.
2. A pass: XOR each A=1 entry with its A=0 partner, giving 1,0,0,1.
3. The entries are the coefficients of 1, B, A, AB, so f=1 XOR AB.
4. Check: AB=1 only on row 11, where f=0. §1.5

ANF shape

$$f \text{ is the XOR of } a_S \cdot \prod_{i \in S} x_i \text{ over } S \subseteq [n]$$

Each coefficient a_S is 0 or 1. The empty product is 1; the XOR collects the selected monomials.

Trap: an OBDD is only canonical after you fix the variable order.

Near miss: two OBDDs for the same function can look completely different if they test the variables in different orders; the diagram is unique only once the order is fixed. Comparing diagrams built with different orders proves nothing about equivalence.

Recall check · answer without looking

1. How does a Shannon expansion combine its two cofactors?
2. What conditions make an OBDD reduced and canonical?

Answers: see "1.5 Shannon views, OBDDs, and algebraic normal form" under Answers at the end.

Made by Thanate Amornjirapo

1.6 Verification, hazards, and wrap-up

Equivalent formulas can behave differently while a real circuit switches. Verification must check both the Boolean function and, where needed, the physical transitions.

> **In plain words:** two formulas can be logically equal and still behave differently in hardware. While an input changes, signals travel along paths of different length, and the output can blip to a wrong value for an instant: a **hazard**. Verification checks the function and, for circuits, the transitions too.

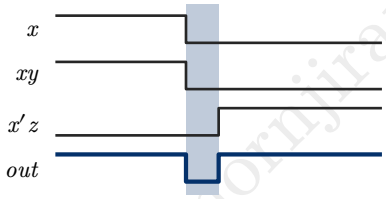
Static-1 hazards

Consensus relation

$$xy + x'z + yz = xy + x'z$$

Read it as: "$xy + x'z + yz$ equals $xy + x'z$": the yz term never changes the value, but it covers the moment when x switches.

The term yz is logically redundant but can preserve coverage during a circuit transition.



the output dips while x falls

A static-1 hazard: when x falls with y = z = 1, the term x′z switches on a moment after xy switches off, so the output dips to 0. The consensus term yz would hold it at 1.

A static-1 hazard occurs when the output should stay 1 for a small input change but a particular two-level sum-of-products implementation briefly drops to 0 because its paths have different delays. If two product terms hand off across one changing literal, their consensus term supplies overlapping coverage, for one input changing at a time. To form it, take terms xA and x′B: the consensus is AB.

Derivation · the consensus theorem

1. Expand: yz=yz(x+x′)=xyz+x′yz.
2. Absorb: xy+xyz=xy and x′z+x′yz=x′z.
3. Therefore xy+x′z+yz=xy+x′z. §1.6

Worked example · add the consensus term

For f=xy+x′z, the consensus term is yz.

1. The terms xy and x′z differ on x. When y=z=1, exactly one is 1 at each stable input, so the output should stay 1; during the change a delay can leave both at 0.
2. Add yz. The consensus theorem gives xy+x′z+yz=xy+x′z, so the Boolean function is unchanged.
3. In a two-level sum-of-products implementation, yz keeps the output covered during the handoff and removes the static-1 hazard described in the lecture. §1.6

Soundness and completeness

Equational reasoning is sound when every derived equation is valid in every Boolean algebra model. It is complete for tautological equivalence when every such equivalence can be derived using the calculus; that needs the full axiom list of §1.1, including the second distributive law. The lecture's wrap-up ties this proof view to the alternate ring, set, map, and decision-diagram representations. Its further-reading slide points to Stone on representation, Post on completeness, and Bryant on decision diagrams.

Sign check: Boolean equivalence says outputs agree for stable inputs; it does not guarantee zero propagation delay in hardware.

Near miss: for $f = x{\cdot}y + x'{\cdot}z$ with $y = z = 1$, the output should stay 1 while x switches, yet for an instant both $x{\cdot}y$ and $x'{\cdot}z$ can be 0. Adding the redundant term $y{\cdot}z$ keeps one term at 1 throughout.

Recall check · answer without looking

1. Why can a redundant consensus term help a physical circuit?
2. What is the difference between soundness and completeness?

Answers: see "1.6 Verification, hazards, and wrap-up" under Answers at the end.

Formula map

Idea	Relation	Use
Duality	Swap + with · and 0 with 1	Derive the matching law.
De Morgan	$(x+y)'=x'\cdot y'(x\cdot y)'=x'+y'$	Move a complement across an operation.
Absorption	$x+x\cdot y=x$	Remove a term already covered by x.
Consensus	$xy+x'z+yz=xy+x'z$	Remove a logically redundant term; add it back to cover a hazard.
Shannon	$f=xf_1+x'f_0$	Split on a variable.
Boolean difference	Difference in x = f_1 XOR f_0	Test whether x can change the result.
K-map group	2^k cells leave n−k literals	Eliminate variables that change.
NAND	$x\uparrow y=(xy)'$	Build NOT, AND, and OR from one gate type.

Made by Thanate Angkornjiraphanit

Derive it cold

On blank paper, rebuild each chain using only earlier definitions and laws.

1. **Duality:** each axiom's dual is an axiom, so swapping symbols in a proof of x+x=x proves x·x=x. §1.1
2. **De Morgan:** show $x'y'$ complements x+y, then use uniqueness. §1.2
3. **Absorption:** x+xy=x(1+y)=x. §1.2
4. **Shannon:** f=f(x+x′)=xf+x′f, then read each term as a cofactor. §1.2
5. **NAND completeness:** derive NOT, AND, and OR, then cite functional completeness of those connectives. §1.3
6. **K-map grouping:** pair adjacent cells and show the changing variable disappears. §1.4
7. **OBDD uniqueness:** explain why the fixed order plus reduction rules determine the same graph. §1.5
8. **Consensus and hazard cover:** expand yz with x+x′ and show it is absorbed by xy+x′z. §1.6

Explain it

Answer each "why" or "when" in your own words before opening the model response.

1. Why does duality save proof work?
2. Why is uniqueness of complements useful for proving De Morgan's law?
3. When is an apparently shorter Boolean expression unsafe to use as a circuit simplification?
4. Why can NAND alone express every Boolean function?
5. When should a don't-care be included in a K-map group?
6. Why does an OBDD's variable order matter even when its reduced form is canonical?
7. Why does a K-map group eliminate the variables that change?

Answers: see "Explain it" under Answers at the end.

Made by Thanun Anornjiraporn (Focus)

Mixed practice

Try all 26 questions in order without notes. The topics are interleaved; write your confidence beside each answer before checking the hidden key.

1. State the dual of x+0=x.
2. Simplify a+a·b. Name the law.
3. Using NAND ↑, write NOT x.
4. How many cells are on a 5-variable K-map?
5. For f(p,q,r)=pq+p′r′, find $f_{p=1}$ and $f_{p=0}$.
6. What consensus term can be added to pq′+p′r to cover a static-1 handoff?
7. In a Boolean ring, calculate 1 XOR 1 and 1·1.
8. Rewrite (a+b)′ using De Morgan's law.
9. For f(A,B)=1 on rows 00 and 11 only, give one DNF.
10. A group of 4 cells is used on a 4-variable K-map. How many literals remain?
11. For f=xy+x′z, find (∃x)f and (∀x)f.
12. Distinguish soundness from completeness in equational reasoning.
13. Give the dual of x+x′=1.
14. What is the Boolean difference of f=x with respect to x?
15. The lecture says a monotone affine function must be constant. Give a counterexample, and state the correct result.
16. With WXYZ as the bit order, use the slide 36 listed rows: 1s at 1,3,7,11,15 and don't-cares at 5,9,13. What is the simplest function?
17. Write A OR B in algebraic normal form over F_2.
18. Does removing bd change the Boolean function ab+a′d+bd?
19. Why can exact minimum two-level DNF search require a heuristic in practice?
20. What does a Boolean difference of 0 with respect to x imply?
21. How many nonterminal nodes does the reduced OBDD of A?B:C have with order B,C,A?
22. With A,B,C as the bit order, f=1 on minterms 0, 2, 4, 6. Simplify f on a K-map.
23. Find the ANF of the outputs 1,0,0,1 for rows 00, 01, 10, 11.
24. For f(A,B)=0 on rows 01 and 10 only, give the CNF.
25. Read + as XOR and · as AND on {0,1}. Which familiar law fails at x=1, and which missing axiom rules this reading out?
26. Use Quine–McCluskey on minterms 4, 5, 6, 7 in A,B,C order. What is the prime implicant?

Answers: see "Mixed practice" under Answers at the end.

Made by Thanate Angornjirapern (Focus)

Exam-style problems

Allow about 20 minutes per problem. Show each identity used and check your final expression against the truth conditions.

1 · Prove, represent, and implement

For f(A,B,C)=A′B+AB′C, use Shannon expansion, NAND construction and normal forms from §§1.2, 1.3 and 1.5.

1. Use Shannon expansion on A to write the two cofactors and reconstruct f.
2. Write a NAND-only implementation for each product term and the outer OR, using NAND as the only gate.
3. State whether the given expression is a DNF and explain how you would verify equivalence to your NAND version.

Answers: see "Exam-style problems" under Answers at the end.

2 · Minimise, then check a circuit transition

Let f(A,B,C)=1 for minterms 2,3,6,7 and 0 elsewhere. Use A,B,C as the bit order with A most significant.

1. Group the true cells on a 3-variable K-map and derive the minimal expression.
2. Expand your result on A using Shannon's theorem.
3. Explain what a static-1 hazard is and when the consensus theorem helps with it.

Answers: see "Exam-style problems" under Answers at the end.

Made by Thanate Angkrijaporn (Focus)

Slips in the notes

Each item was checked against the rendered slides and is listed with its slide number. The lecture's wording is given too, so you can recognise it. Ask the lecturer which version an exam would mark.

- **Slide 6 · error, second distributive law missing:** the axiom list prints only x·(y+z)=x·y+x·z, although its intuition box says distribution works both ways. XOR and AND on {0,1} satisfy every printed line, yet 1 XOR 1=0, so idempotence (slide 7), absorption's step 1+y=1, duality and completeness (slide 45) do not follow from the list as printed. §1.1 adds x+y·z=(x+y)·(x+z). The word "minimal" is probably also too strong, since associativity is usually derivable from Huntington's set.
- **Slides 11 and 13 · notation clash:** slide 11 and most of the lecture use + for join, while slide 13 reuses + for ring addition, giving x+x=0 beside x+x=x. This guide writes ring addition as a circled plus in formula cards and as XOR in running text.
- **Slide 27 · error, Post's conditions incomplete:** the slide says a nonmonotone and a nonaffine operation imply completeness. The function x·y′ is both, yet it preserves 0, so it cannot make NOT or 1. Post's criterion needs five: for each of the classes 0-preserving, 1-preserving, self-dual, monotone and affine, the set must contain a function outside it. §1.3 uses only the NAND argument.
- **Slide 28 · error, monotone and affine:** the slide says such a function is constant. But f(x)=x is both. The correct result is a constant or a single input variable; enumerating all 256 functions of three variables gives exactly 0, 1, x_1, x_2 and x_3. §1.3 and practice 15 use the corrected form.
- **Slide 30 · simplification:** "NP-complete" is printed without saying what the input is. This guide states the truth-table form; the problem is harder when the input is a formula rather than a table. That last remark is a standard result, not from the notes.
- **Slide 34 · wording:** changing variables are said to be "eliminated by consensus". The step that removes a variable is xA+x′A=A, which is distribution and complement, not the consensus theorem. §1.4 says "eliminated" and §1.3 shows the combining step.
- **Slide 35 · minterm order not stated:** the slide gives Σm(1,3,5,7) and concludes f=B, without labelling the bit order. Under A,B,C with A most significant, that set describes C; the slide's conclusion needs a different order. Its intuition box says A and C vary, which fits f=B, not the listed set in that order. The drawn map has no axis labels, and its two groups are separated by one column; with Gray columns, the B=1 columns would be next to each other. This guide states the convention in its worked example.
- **Slide 36 · expression conflicts with listed rows:** with W,X,Y,Z in that bit order, the listed 1s {1,3,7,11,15} and don't-cares {5,9,13} are all odd indices, and every even index is 0, so the list simplifies to f=Z when all three don't-cares are used as 1. The printed expression YZ+X′Z+W′Z′ also covers the required zeros 0, 2, 4 and 6, while YZ+X′Z alone already covers every 1. W′Z′ is probably a typo for W′Z. The grid has no axis labels, and its coloured groups do not obviously match the printed terms; that reading is uncertain. This guide follows the listed rows, "1s at 1,3,7,11,15".
- **Slide 43 · simplification:** "adding consensus terms eliminates static hazards" holds when one input changes at a time and all the needed consensus terms are added. §1.6 says so.

Work by Thanh-Tu Antonijuan Nguyen (Focus)

Self-check

Use this list in a later session, at least a day after learning the material. Tick a skill only when you can demonstrate it without notes.

- ☐ State the Boolean operations, constants, and core axioms. (1, 7, 25)
- ☐ Apply duality to a proved equation. (1, 13)
- ☐ Prove complement uniqueness and use it for De Morgan. (Derive it cold)
- ☐ Simplify with absorption and consensus. (2, 18)
- ☐ Compute cofactors, Shannon expansion, and Boolean difference. (5, 14, 20)
- ☐ Distinguish existential and universal quantification. (11)
- ☐ Translate XOR and AND in the Boolean ring view. (7, 17, 25)
- ☐ Build NOT, AND, and OR from NAND. (3)
- ☐ Create DNF and CNF from truth conditions. (9, 24)
- ☐ Explain why minimum exact forms can be computationally hard. (19, 26)
- ☐ Group K-map cells and count the remaining literals. (4, 10, 16, 22)
- ☐ State the variable order used by minterm indices. (16)
- ☐ Explain OBDD reduction and fixed-order canonicity. (21)
- ☐ Write a simple function in algebraic normal form. (17, 23)
- ☐ Distinguish Boolean equivalence from hazard-free switching. (6, 18)
- ☐ Rebuild the derivations from their definitions.
- ☐ Complete both multi-part problems and justify each step.

Made by Thanate Angkoonsawatsakul (Focus)

Answers

Start from memory

1. Join (+), meet (·), complement (′), and constants 0 and 1. §1.1
2. $x \cdot 1 = x$, swapping + with · and 0 with 1. §1.1
3. $x + x{\cdot}y = x$, or its dual. §1.2
4. It fixes the selected variable first to 1 and then to 0, combining the two cofactors. §1.5
5. Two literals remain: a group of 2^3 eliminates 3 of 5 variables. §1.4

1.1 Foundations: one structure, several views

1. Binary join +, binary meet ·, unary complement ′, and constants 0 and 1. §1.1
2. XOR is ring addition and AND is ring multiplication; each element satisfies x^2=x. §1.1

1.2 Core identities and proof tools

1. Show $x' \cdot y'$ has join 1 and meet 0 with x+y; it is therefore the unique complement of x+y. §1.2
2. The two cofactors agree everywhere, so flipping x never changes f; f does not depend on x. §1.2

1.3 Completeness, normal forms, and minimisation

1. NOT x=x↑x; AND x·y=(x↑y)↑(x↑y); OR x+y=(x↑x)↑(y↑y). §1.3
2. DNF ORs minterms for true rows. Deciding exact two-level minimality from a truth table is NP-complete, as the lecture states, so structured methods and heuristics matter. §1.3

1.4 Karnaugh maps remove changing variables

1. Five minus three changing variables leaves two literals. §1.4
2. Choose 0 or 1 if useful, but never use a don't-care to cover a required zero. §1.4

1.5 Shannon views, OBDDs, and algebraic normal form

1. Multiply the x=1 cofactor by x, the x=0 cofactor by x′, then OR the two cases. §1.5
2. Fix a variable order, merge isomorphic subgraphs, and remove redundant tests; uniqueness is for that fixed order. §1.5

Made by Thanate Amornjirapong (Focus)

1.6 Verification, hazards, and wrap-up

1. It preserves overlapping coverage while unequal path delays could otherwise create a brief 1-to-0 glitch. §1.6
2. Soundness means every proved equation is valid. Completeness means every tautological equivalence is derivable. §1.6

Explain it

1. The duality principle preserves derivability when + and ·, and 0 and 1, are exchanged. Once one equation is proved, its dual follows from the same axiom system without repeating the argument. §1.1
2. It is enough to verify that the proposed expression has join 1 and meet 0 with the original expression. Those two properties make it a complement, and uniqueness identifies it with the negation. §1.2
3. Logical equivalence alone does not account for unequal propagation delays. A two-level sum-of-products implementation may glitch during a transition, so a consensus term can be needed even when it is algebraically redundant. §1.6
4. NAND can produce NOT by tying its inputs together. It can then produce AND and OR, and those connectives can express every Boolean function. §1.3
5. Include it when it helps make a larger legal group and removes literals. Ignore it when it does not help; it is not a required 1 and must never force a group through a required 0. §1.4
6. Canonicity holds for a fixed order. Changing the order changes the sequence of Shannon splits and can therefore change the reduced diagram's shape and size. §1.5
7. The group contains every combination of the changing variables while the remaining variables stay fixed. Combining those cases with Boolean identities removes the changing literals, leaving only the fixed ones. §1.4

Made by Thanate Angkrigaporn (Focus)

Mixed practice

1. $x \cdot 1 = x$, swapping join with meet and 0 with 1. §1.1
2. $a + a \cdot b = a$ by absorption. §1.2
3. $x' = x \uparrow x$ because NAND of x with itself negates x. §1.3
4. $2^5 = 32$ cells, one per input assignment. §1.4
5. $f_{p=1} = q + 0 = q$; $f_{p=0} = 0 + r' = r'$. §1.5
6. From pA and $p'B$ with $A = q'$ and $B = r$, the consensus is $q'r$, giving $pq' + p'r + q'r$. The added term preserves the function and overlaps the handoff. §1.6
7. 1 XOR 1 = 0 by characteristic 2; $1 \cdot 1 = 1$ by the multiplicative identity. §1.1
8. $(a + b)' = a'b'$. §1.2
9. $A'B' + AB$, the two minterms for rows 00 and 11. §1.3
10. Two literals: 4 cells means $k = 2$, so $n - k = 4 - 2 = 2$. §1.4
11. The cofactors are y and z, so $(\exists x)f = y + z$ and $(\forall x)f = yz$. §1.2
12. Soundness means every derivable equation is valid; completeness means every tautological equivalence is derivable. §1.6
13. $x \cdot x' = 0$, the dual of $x + x' = 1$. §1.1
14. 1: the cofactors of x with respect to itself are 1 and 0, and 1 XOR 0 = 1. §1.2
15. $f(x) = x$ is monotone and affine. Correct: a constant or a single input variable, because any XOR of two or more inputs falls from 1 to 0 when a second input turns on. §1.3
16. Every odd index is either 1 or a don't-care and every even index is 0, so f = Z under the stated order. §1.4
17. A OR B = A XOR B XOR AB. Check rows 00, 01, 10, and 11 to verify the outputs 0,1,1,1. §1.5
18. No. bd is the consensus of ab and $a'd$, so it is redundant in the stable function, though it may matter for hazards. §1.6
19. Deciding minimality from a truth table is NP-complete, as the lecture states, so the search can grow too costly; structured methods and heuristics help. §1.3
20. The cofactors agree everywhere, so changing x never changes the function. §1.2
21. 5: B on top; each B branch needs its own C test, because they lead to different subfunctions; and those C tests lead to two different A tests, one for A and one for A'. §1.5
22. $f = C'$. The even rows fill columns 00 and 10, a group that wraps; A and B vary while C = 0. §1.4
23. B pass gives 1,1,0,1; A pass gives 1,1,1,0. So the coefficients of 1,B,A,AB give 1 XOR A XOR B. Check: row 11 gives 1 XOR 1 XOR 1 = 1. §1.5
24. $(A + B')(A' + B)$, one maxterm for each false row 01 and 10. §1.3
25. Idempotence: $1 + 1 = 0$, not 1. The second distributive law fails too (x = y = 1, z = 0 gives 1 against 0), so that axiom rules this reading out. §1.1
26. 100, 101, 110, 111 combine to 10− and 11−, then to 1−−, which is A. §1.3

Made by Thanatee Anornjiraporn (Focus)

Exam-style problems

1. For A=1, f=B′C; for A=0, f=B. Therefore f=A·(B′C)+A′·B, the original function after reordering its terms. §1.2 §1.5
2. Make complements with x′=x↑x. For p=A′B, compute u=A′↑B=(A′B)′, then p=u↑u. For q=AB′C, compute v=A↑B′=(AB′)′, r=v↑v=AB′, s=r↑C=(AB′C)′, then q=s↑s. Finally f=(p↑p)↑(q↑q), which is p OR q by De Morgan. §1.3
3. Yes: it is an OR of product terms. A truth table or algebraic rewriting can show that the gate expression produces the same value on every assignment. §1.3

Exam-style problems

1. The rows 010, 011, 110, and 111 all have B=1; they form a group of four. A and C vary, so f=B. §1.4
2. B is independent of A, so both A cofactors equal B. Shannon gives f=AB+A′B=B(A+A′)=B. §1.2 §1.5
3. A static-1 hazard is a brief drop to 0 when the intended output stays 1 during a small input change. In a two-level sum-of-products implementation, an added consensus term can maintain overlapping coverage without changing the Boolean function. §1.6

Made by Thanate Amornjiraporn (Focus)

Personal study guide based on Bernhard Kainz, “Boolean Algebra, Identities, Proofs, and Minimization: An advanced mathematical perspective for CS undergraduates,” supplied as APDR lecture01 - extended.pdf (48 slides, dated 5 October 2025). Written in my own words; no lecture pages or figures are reproduced. Source inconsistencies are listed under Slips in the notes. Not an official FAU publication; check course details against the current materials. Made by **Thanate Amornjiraporn (Focus)**.