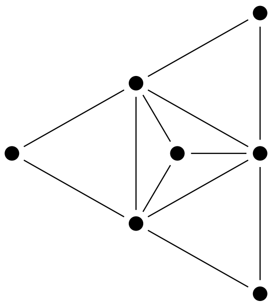
COMP9020 Lecture 5
Session 2, 2017
Logic

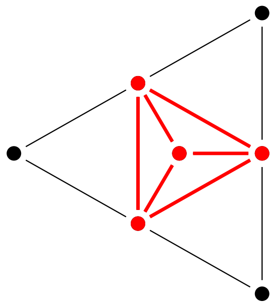
Lecture 4 recap: Graphs

- Basic definitions: Degree, (Simple, closed) paths, cycles
- Trees, Complete (k-partite) graphs: K_n, $K_{m,n}$, $K_{m,n,p}$, ...
- Eulerian paths and circuits
- Hamiltonian paths and circuits
- Graph colouring, chromatic number $\chi(G)$
- Cliques, clique number $\kappa(G)$
- Planarity

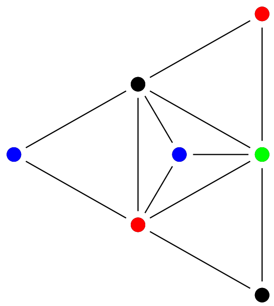
Example: Planar



Example: Clique number $\kappa(G) = 4$



Example: Chromatic number $\chi(G) = 4$



COMP9020 Lecture 5
Session 2, 2017
Logic

- Textbook (R & W) – Ch. 2, Sec. 2.1-2.5;
  Ch. 10, Sec. 10.1-10.3
- Problem sets 5 and 6
- Supplementary Exercises Ch. 2 and 10 (R & W)
- *Guidelines for good mathematical writing*

Overview

- what's a proof?
- from English to propositional logic
- truth tables, validity, satisfiability and entailment
- *applications:* program logic, constraint satisfaction problems, reasoning about specifications, digital circuits
- proof methods
- generalisation: Boolean algebras

Proofs

A **mathematical proof** of a proposition p is a chain of logical deductions leading to p from a base set of axioms.

Example

Proposition: Every group of 6 people includes a group of 3 who each have met each other or a group of 3 who have not met a single other person in that group.
Proof: by case analysis.

But what are propositions, logical deductions, and axioms? And what is a sound case analysis?

The Real World vs Symbols

$$\begin{tikzcd}
\text{symbols} \arrow[r, "\text{symbolic manipulation}"] \arrow[d, "\text{relationship}"] & \text{symbols} \arrow[d, "\text{relationship}"] \\
\text{world} \arrow[r, "\text{physical operation}"] & \text{world}
\end{tikzcd}$$

NB

"Essentially, all models are wrong. But some are useful." (G. Box)

The main relationship between symbols and the world of concern in logic is that of a *sentence of a language* being *true* in the world. A sentence of a natural language (like English, Cantonese, Warlpiri) is *declarative*, or a *proposition*, if it can be meaningfully be said to be either true or false.

Examples

- Richard Nixon was president of Ecuador.
- A square root of 16 is 4.
- Euclid's program gets stuck in an infinite loop if you input 0.
- Whatever list of numbers you give as input to this program, it outputs the same list but in increasing order.
- $x^n + y^n = z^n$ has no nontrivial integer solutions for $n > 2$.
- 3 divides 24.
- K_5 is planar.

The following are *not* declarative sentences:

- Gubble gimble goo
- For Pete's sake, take out the garbage!
- Did you watch MediaWatch last week?
- Please waive the prerequisites for this subject for me.
- x divides y. — $R(x, y)$
- $x = 3$ and x divides 24. — $P(x)$

The following are *not* declarative sentences:

- Gubble gimble goo
- For Pete's sake, take out the garbage!
- Did you watch MediaWatch last week?
- Please waive the prerequisites for this subject for me.
- x divides y. — $R(x, y)$
- $x = 3$ and x divides 24. — $P(x)$

Declarative sentences in natural languages can be *compound* sentences, built out of other sentences.
Propositional Logic is a formal representation of some constructions for which the truth value of the compound sentence can be determined from the truth value of its components.

- Chef is a bit of a Romeo *and* Kenny is always getting killed.
- Either Bill is a liar *or* Hillary is innocent of Whitewater.
- *It is not the case that* this program always halts.

Not all constructions of natural language are truth-functional:

- *Obama believes that* Iran is developing nukes.
- *Chef said* they killed Kenny.
- This program always halts *because* it contains no loops.
- The disk crashed *after* I saved my file.

NB

Various **modal logics** *extend classical propositional logic to represent, and reason about, these and other constructions.*

The Three Basic Connectives of Propositional Logic

symbol	text
$\wedge$	"and", "but", ";", ":"
$\vee$	"or", "either ... or ..."
$\neg$	"not", "it is not the case that"

Truth tables:

A	B	A $\wedge$ B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A $\vee$ B
F	F	F
F	T	T
T	F	T
T	T	T

A	$\neg$ A
F	T
T	F

Applications I: Program Logic

Example

```
if x > 0 or (x <= 0 and y > 100):
```

Let $p \stackrel{\text{def}}{=} (\texttt{x > 0})$ and $q \stackrel{\text{def}}{=} (\texttt{y > 100})$

$p \;\vee\; (\neg p \wedge q)$

p	q	$\neg p$	$\neg p \wedge q$	$p \vee (\neg p \wedge q)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

This is equivalent to $p \vee q$. Hence the code can be simplified to

```
if x > 0 or y > 100:
```

Somewhat more controversially, consider the following constructions:

- if A then B
- A only if B
- B if A
- A implies B
- it follows from A that B
- whenever A, B
- A is a sufficient condition for B
- B is a necessary condition for A

Each has the property that if the whole statement is true, and A is true, then B is true.

We can *approximate* the English meaning of these by "not (A and not B)", written $A \rightarrow B$, which has the following truth table:

A	B	A → B
F	F	
F	T	
T	F	F
T	T	T

How to interpret $A \rightarrow B$ when A is false?

E.g. "If I am the president of Australia, then I have blue eyes"

"All presidents of Australia have blue eyes" vs.
"All presidents of Australia do not have blue eyes"

"If `false` then `true`" and "If `false` then `false`" are *vacuously* true

How to interpret $A \rightarrow B$ when A is false?

E.g. "If I am the president of Australia, then I have blue eyes"

"All presidents of Australia have blue eyes" vs.
"All presidents of Australia do not have blue eyes"

"If `false` then `true`" and "If `false` then `false`" are *vacuously* true

How to interpret $A \rightarrow B$ when A is false?

E.g. "If I am the president of Australia, then I have blue eyes"

"All presidents of Australia have blue eyes" vs.
"All presidents of Australia do not have blue eyes"

"If `false` then `true`" and "If `false` then `false`" are *vacuously* true

We can *approximate* the English meaning of these by "not (A and not B)", written $A \rightarrow B$, which has the following truth table:

A	B	A $\rightarrow$ B
F	F	T
F	T	T
T	F	F
T	T	T

While only an approximation to the English, 100+ years of experience have shown this to be adequate for capturing *mathematical reasoning*.
(Moral: mathematical reasoning does not need all the features of English.)

Examples

LLM: Problem 3.2

p = "you get an HD on your final exam"
q = "you do every exercise in the book"
r = "you get an HD in the course"

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book.

(c) To get an HD in the course, you must get an HD on the exam.

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course.

Examples

LLM: Problem 3.2

p = "you get an HD on your final exam"
q = "you do every exercise in the book"
r = "you get an HD in the course"

Translate into logical notation:

(a) You get an HD in the course although you do not do every exercise in the book. $r \wedge \neg q$

(c) To get an HD in the course, you must get an HD on the exam.
$r \rightarrow p$

(d) You get an HD on your exam, but you don't do every exercise in this book; nevertheless, you get an HD in this course.
$p \wedge \neg q \wedge r$

Unless

A unless B can be approximated as $\neg B \to A$

E.g.
I go swimming unless it rains = If it is not raining I go swimming.
Correctness of the translation is perhaps easier to see in:
I don't go swimming unless the sun shines = If the sun does not shine then I don't go swimming.

Note that "I go swimming unless it rains, but sometimes I swim even though it is raining" makes sense, so the translation of "A unless B" should not imply $B \to \neg A$.

Just in case

A just in case B usually means *A if, and only if, B*; written $A \leftrightarrow B$

The program terminates just in case the input is a positive number.
= The program terminates if, and only if, the input is positive.

I will have an entree just in case I won't have desert.
= If I have desert I will not have an entree and vice versa.

It has the following truth table:

A	B	A $\leftrightarrow$ B
F	F	T
F	T	F
T	F	F
T	T	T

Same as $(A \to B) \wedge (B \to A)$

Propositional Logic as a Formal Language

Let $Prop = \{p, q, r, \ldots\}$ be a set of basic propositional letters. Consider the *alphabet*

$$\Sigma = Prop \cup \{\top, \bot, \neg, \wedge, \vee, \to, \leftrightarrow, (,)\}$$

The set of **formulae of propositional logic** is the smallest set of words over Σ such that

- $\top$, $\bot$ and all elements of *Prop* are formulae
- If ϕ is a formula, then so is $\neg\phi$
- If ϕ and ψ are formulae, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \to \psi)$, and $(\phi \leftrightarrow \psi)$.

Convention: we often drop parentheses when there is no ambiguity. $\neg$ binds more tightly than $\wedge$ and $\vee$, which in turn bind more tightly than $\to$ and $\leftrightarrow$.

Logical Equivalence

Two formulas ϕ, ψ are **logically equivalent**, denoted $\phi \equiv \psi$ if they have the same truth value for all values of their basic propositions.

Application: If ϕ and ψ are two formulae such that $\phi \equiv \psi$, then the digital circuits corresponding to ϕ and ψ compute the same function. Thus, proving equivalence of formulas can be used to *optimise* circuits.

Some Well-Known Equivalences

Excluded Middle	$p \vee \neg p \equiv \top$
Contradiction	$p \wedge \neg p \equiv \bot$
Identity	$p \vee \bot \equiv p$
	$p \wedge \top \equiv p$
	$p \vee \top \equiv \top$
	$p \wedge \bot \equiv \bot$
Idempotence	$p \vee p \equiv p$
	$p \wedge p \equiv p$
Double Negation	$\neg\neg p \equiv p$
Commutativity	$p \vee q \equiv q \vee p$
	$p \wedge q \equiv q \wedge p$

Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distribution	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
De Morgan's laws	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$
Implication	$p \to q \equiv \neg p \vee q$
	$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

Example

$((r \wedge \neg p) \vee (r \wedge q)) \vee ((\neg r \wedge \neg p) \vee (\neg r \wedge q))$

$\equiv$	$(r \wedge (\neg p \vee q)) \vee (\neg r \wedge (\neg p \vee q))$	Distrib.
$\equiv$	$(r \vee \neg r) \wedge (\neg p \vee q)$	Distrib.
$\equiv$	$\top \wedge (\neg p \vee q)$	Excl. Mid.
$\equiv$	$\neg p \vee q$	Ident.

Examples

$\boxed{2.2.18}$ Prove or disprove:

(a) $p \to (q \to r) \ \equiv \ (p \to q) \to (p \to r)$

(c) $(p \to q) \to r \ \equiv \ p \to (q \to r)$

Examples

$\boxed{2.2.18}$ Prove or disprove:

(a) $(p \to q) \to (p \to r)$

$\equiv \neg(p \to q) \vee (\neg p \vee r)$

$\equiv (p \wedge \neg q) \vee \neg p \vee r$

$\equiv (p \vee \neg p \vee r) \wedge (\neg q \vee \neg p \vee r)$

$\equiv \top \wedge (\neg p \vee \neg q \vee r)$

$\equiv p \to (\neg q \vee r)$

$\equiv p \to (q \to r)$

(c) $(p \to q) \to r \;\equiv\; p \to (q \to r)$

Counterexample:

p	q	r	$(p \to q) \to r$	$p \to (q \to r)$
F	T	F	F	T

Satisfiability of Formulas

A formula is **satisfiable**, if it evaluates to T for *some* assignment of truth values to its basic propositions.

Example

A	B	$\neg(A \to B)$
F	F	F
F	T	F
T	F	T
T	T	F

Applications II: Constraint Satisfaction Problems

These are problems such as timetabling, activity planning, etc. Many can be understood as showing that a formula is satisfiable.

Example

You are planning a party, but your friends are a bit touchy about who will be there.

1. If John comes, he will get very hostile if Sarah is there.
2. Sarah will only come if Kim will be there also.
3. Kim says she will not come unless John does.

Who can you invite without making someone unhappy?

Translation to logic: let J, S, K represent "John (Sarah, Kim) comes to the party". Then the constraints are:

1. $J \to \neg S$
2. $S \to K$
3. $K \to J$

Thus, for a successful party to be possible, we want the formula $\phi = (J \to \neg S) \wedge (S \to K) \wedge (K \to J)$ to be satisfiable.
Truth values for J, S, K making this true are called *satisfying assignments*, or *models*.

We figure out where the conjuncts are false, below. (so blank = T)

J	K	S	$J \to \neg S$	$S \to K$	$K \to J$	ϕ
F	F	F				
F	F	T		F		F
F	T	F			F	F
F	T	T			F	F
T	F	F				
T	F	T	F	F		F
T	T	F				
T	T	T	F			F

Conclusion: a party satisfying the constraints can be held. Invite nobody, or invite John only, or invite Kim and John.

Exercise

2.7.14 (supp)

Which of the following formulae are *always* true?

(a) $(p \wedge (p \to q)) \to q$ — always true

(b) $((p \vee q) \wedge \neg p) \to \neg q$ — not always true

(e) $((p \to q) \vee (q \to r)) \to (p \to r)$ — not always true

(f) $(p \wedge q) \to q$ — always true

Exercise

2.7.14 (supp)

Which of the following formulae are *always* true?

(a) $(p \wedge (p \to q)) \to q$ — always true

(b) $((p \vee q) \wedge \neg p) \to \neg q$ — not always true

(e) $((p \to q) \vee (q \to r)) \to (p \to r)$ — not always true

(f) $(p \wedge q) \to q$ — always true

Validity, Entailment, Arguments

An *argument* consists of a set of declarative sentences called *premises* and a declarative sentence called the *conclusion*.

Example

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is not late.
Conclusion:	Frank took the Toyota

An argument is *valid* if the conclusions are true *whenever* all the premises are true. Thus: if we believe the premises, we should also believe the conclusion.
(Note: we don't care what happens when one of the premises is false.)
Other ways of saying the same thing:

- The conclusion *logically follows* from the premises.
- The conclusion is a *logical consequence* of the premises.
- The premises **entail** the conclusion.

The argument above is valid. The following is invalid:

Example

Premises:	Frank took the Ford or the Toyota.
	If Frank took the Ford he will be late.
	Frank is late.
Conclusion:	Frank took the Ford.

For arguments in propositional logic, we can capture validity as follows:
Let $\phi_1, \dots, \phi_n$ and ϕ be formulae of propositional logic. Draw a truth table with columns for each of $\phi_1, \dots, \phi_n$ and ϕ.
The argument with premises $\phi_1, \dots, \phi_n$ and conclusion ϕ is valid, denoted

$$\phi_1, \dots, \phi_n \models \phi$$

if in every row of the truth table where $\phi_1, \dots, \phi_n$ are all true, ϕ is true also.

We mark only true locations (blank = F)

Frd	$Tyta$	$Late$	$Frd \lor Tyta$	$Frd \to Late$	$\neg Late$	$Tyta$
F	F	F		T	T	
F	F	T		T		
F	T	F	T	T	T	T
F	T	T	T	T		T
T	F	F	T		T	
T	F	T	T	T		
T	T	F	T		T	T
T	T	T	T	T		T

This shows $Frd \lor Tyta,\ Frd \to Late,\ \neg Late \models Tyta$

The following row shows $Frd \vee Tyta$, $Frd \rightarrow Late$, $Late \not\models Frd$

Frd	$Tyta$	$Late$	$Frd \vee Tyta$	$Frd \rightarrow Late$	$Late$	Frd
F	T	T	T	T	T	F

Applications III: Reasoning About Requirements/Specifications

Suppose a set of English language requirements R for a software/hardware system can be formalised by a set of formulae $\{\phi_1, \dots \phi_n\}$.
Suppose C is a statement formalised by a formula ψ. Then

1. The requirements cannot be implemented if $\phi_1 \wedge \dots \wedge \phi_n$ is not satisfiable.
2. If $\phi_1, \dots \phi_n \models \psi$ then every correct implementation of the requirements R will be such that C is always true in the resulting system.
3. If $\phi_1, \dots \phi_{n-1} \models \phi_n$, then the condition ϕ_n of the specification is redundant and need not be stated in the specification.

Example

Requirements R: A burglar alarm system for a house is to operate as follows. The alarm should not sound unless the system has been armed or there is a fire. If the system has been armed and a door is disturbed, the alarm should ring. Irrespective of whether the system has been armed, the alarm should go off when there is a fire.
Conclusion C: If the alarm is ringing and there is no fire, then the system must have been armed.
Questions

1. Will every system correctly implementing requirements R satisfy C?
2. Is the final sentence of the requirements redundant?

Expressing the requirements as formulas of propositional logic, with

- S = the alarm sounds = the alarm rings
- A = the system is armed
- D = a door is disturbed
- F = there is a fire

we get
Requirements:

1. $S \rightarrow (A \vee F)$
2. $(A \wedge D) \rightarrow S$
3. $F \rightarrow S$

Conclusion: $(S \wedge \neg F) \rightarrow A$

Our two questions then correspond to

1. Does $S \to (A \vee F)$, $(A \wedge D) \to S$, $F \to S \models (S \wedge \neg F) \to A$?
2. Does $S \to (A \vee F)$, $(A \wedge D) \to S \models F \to S$?

Answers: problem set 2, exercise 2

Validity of Formulas

A formula ϕ is **valid**, or a **tautology**, denoted $\models \phi$, if it evaluates to T for *all* assignments of truth values to its basic propositions.

Example

A	B	$(A \to B) \to (\neg B \to \neg A)$
F	F	T
F	T	T
T	F	T
T	T	T

Validity, Equivalence and Entailment

Theorem

The following are equivalent:

- $\phi_1, \dots \phi_n \models \psi$
- $\models (\phi_1 \wedge \dots \wedge \phi_n) \to \psi$
- $\models \phi_1 \to (\phi_2 \to \dots (\phi_n \to \psi) \dots)$

Theorem

$\phi \equiv \psi$ *if and only if* $\models \phi \leftrightarrow \psi$

Quantifiers

We've made quite a few statements of the kind

"If there exists a satisfying assignment ..."

or

"Every natural number greater than 2 ..."

without formally capturing these quantitative aspects.

Notation: $\forall$ means "for all" and $\exists$ means "there exist(s)"

Example

Goldbach's conjecture

$$\forall n \in 2\mathbb{N} \, (n > 2 \to \exists p, q \in \mathbb{N} \, (p, q \in \textsc{Primes} \wedge n = p + q))$$

Proof Rules and Methods: Proof of the Contrapositive

We want to prove $A \to B$.
To prove it, we show $\neg B \to \neg A$ and invoke the equivalence $(A \to B) \equiv (\neg B \to \neg A)$.

Example

$\forall m, n \in \mathbb{N} \, (m + n \geq 73 \;\; \to \;\; m \geq 37 \vee n \geq 37)$

Proof Rules and Methods: Proof by Contradiction

We want to prove A.
To prove it, we assume $\neg A$, and derive both B and $\neg B$ for some proposition B.
(Hard part: working out what B should be.)

Examples

- $\sqrt{2}$ is irrational
- There exist an infinite number of primes

Proof Rules and Methods: Proof by Cases

We want to prove that A. To prove it, we find a set of cases $B_1, B_2, \ldots, B_n$ such that

1. $B_1 \vee \ldots \vee B_n$, and
2. $B_i \to A$ for each $i = 1..n$.

(Hard Part: working out what the B_i should be.)
(Comment: often $n = 2$ and $B_2 = \neg B_1$, so $B_1 \vee B_2 = B_1 \vee \neg B_1$ holds trivially.)

Example

$|x + y| \leq |x| + |y|$ for all $x, y \in \mathbb{R}$.
Recall:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

Substitution

Substitution is the process of replacing every occurrence of some symbol by an expression.

Examples

The result of substituting 3 for x in

$$x^2 + 7y = 2xz$$

is

$$3^2 + 7y = 2 \cdot 3 \cdot z$$

The result of substituting $2k + 3$ for x in

$$x^2 + 7y = 2xz$$

is

$$(2k + 3)^2 + 7y = 2 \cdot (2k + 3) \cdot z$$

We can substitute logical expressions for logical variables:

Example

The result of substituting $P \wedge Q$ for A in

$$(A \wedge B) \rightarrow A$$

is

$$((P \wedge Q) \wedge B) \rightarrow (P \wedge Q)$$

Substitution Rules

(a) If we substitute an expression for *all* occurrences of a logical variable in a tautology then the result is still a tautology.
If $\models \phi(P)$ then $\models \phi(\alpha)$.

Examples

$\models P \rightarrow (P \vee Q)$, so

$$\models (A \vee B) \rightarrow ((A \vee B) \vee Q)$$

$\boxed{2.5.7}$
$\models \neg Q \rightarrow (Q \rightarrow P)$, so

$$\models \neg(P \rightarrow Q) \rightarrow ((P \rightarrow Q) \rightarrow P)$$

(b) If a logical formula ϕ contains a formula α, and we replace (an occurrence of) α by a logically equivalent formula β, then the result is logically equivalent to ϕ.
If $\alpha \equiv \beta$ then $\phi(\alpha) \equiv \phi(\beta)$.

Example

$P \rightarrow Q \equiv \neg P \vee Q$, so

$$Q \rightarrow (P \rightarrow Q) \equiv Q \rightarrow (\neg P \vee Q)$$

COMP9020 Lecture 6
Session 2, 2017
Logic cont'd

Lecture 5 recap: Logical connectives

- AND — conjunction, $\wedge$, &
- OR — disjunction, $\vee$, ||
- NOT — negation
- Implication, $\rightarrow$, $\supset$ (IF-THEN)
- Bi-implication, $\leftrightarrow$ (IF AND ONLY IF)

A	B	A $\wedge$ B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A $\vee$ B
F	F	F
F	T	T
T	F	T
T	T	T

A	$\neg$ A
F	T
T	F

Lecture 5 recap: Propositional formulae

The set of **formulae of propositional logic** is the smallest set of words over Σ such that

- $\top$, $\bot$ and all elements of *Prop* are formulae
- If ϕ is a formula, then so is $\neg\phi$
- If ϕ and ψ are formulae, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$.

Lecture 5 recap: Truth tables

- Row for every **truth assignment** — assignment of T/F to elements of *Prop*
- Columns for subformulae
- Truth assignments can also map formulae to T/F: be careful!

p	q	$\neg p$	$\neg p \wedge q$	$p \vee (\neg p \wedge q)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

Lecture 5 recap: Truth tables

- Row for every **truth assignment** — assignment of T/F to elements of *Prop*
- Columns for subformulae
- Truth assignments can also map formulae to T/F: be careful!

p	q	$\neg p$	$\neg p \wedge q$	$p \vee (\neg p \wedge q)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

Lecture 5 recap: Truth assignments

A **truth assignment** is a function $\nu : Prop \to \{T, F\}$.

Can extend truth assignments to formulae:

- $\nu(\top) = T$, $\nu(\bot) = F$
- $\nu(\neg\varphi) = \neg\nu(\varphi)$,
- $\nu(\varphi \wedge \psi) = \nu(\varphi) \wedge \nu(psi)$, ...

Lecture 5 recap: Logical equivalence and entailment

Two formulas, φ and ψ, are **logically equivalent**, $\varphi \equiv \psi$, if $\nu(\varphi) = \nu(\psi)$ for all truth assignments ν.

A list of formulae, $\varphi_1, \ldots, \varphi_n$ **entail** a single formula, ψ, written $\varphi_1, \ldots, \varphi_n \models \psi$ if $\nu(\psi) = T$ for all truth assignments ν where $\nu(\varphi_1) = \ldots = \nu(\varphi_n) = T$. If the list is empty, we say ψ is a **tautology**.

Theorem

- *$\phi_1, \ldots \phi_n \models \psi$ if, and only if, $\models (\phi_1 \wedge \ldots \wedge \phi_n) \to \psi$*
- *$\phi \equiv \psi$ if, and only if, $\models \phi \leftrightarrow \psi$*

Lecture 5 recap: Proof techniques

- Proof by contrapositive: show $A \Rightarrow B$ by showing $\neg B \Rightarrow \neg A$
- Proof by contradiction: Assume $\neg A$. Show $B \wedge \neg B$. $\neg A \models \bot$ if and only if $\models A$
- Proof by cases: Given $B_1 \vee \cdots \vee B_n$, show $B_i \models A$ for all i.

Boolean Functions

Formulae can be viewed as **Boolean functions** mapping valuations of their propositional letters to truth values.

A Boolean function of one variable is also called **unary**.
A function of two variables is called **binary**.
A function of n input variables is called **n-ary**.

Question

How many unary Boolean functions are there?
How many binary functions? n-ary?

Question

What connectives do we need to express all of them?

Boolean Arithmetic

Consider truth values with operations $\wedge, \vee, \neg$ as an **algebraic structure**:

- $\mathbb{B} = \{0, 1\}$ with 'Boolean' arithmetic

$$a \wedge b, \ \ a \vee b, \ \ \bar{a} = 1 - a$$

NB

We often write pq for $p \wedge q$.
In electrical and computer engineering, the notation $\overline{p}$ is more common than p', which is often used in mathematics.
Observe that using $\overline{(\cdot)}$ obviates the need for some parentheses.

Applications IV: Digital Circuits

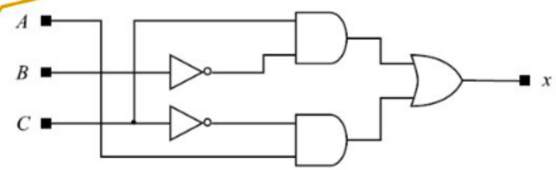
A formula can be viewed as defining a digital circuit, which computes a Boolean function of the input propositions. The function is given by the truth table of the formula.

NB

Common usage: $+$ for or, $\cdot$ for and, $\overline{x}$ for $\neg x$

A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$x = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} = \overline{B}C + A\overline{C}$$



Definition: Boolean Algebra

Every structure consisting of a set T with operations *join*: $a, b \mapsto a \vee b$, *meet*: $a, b \mapsto a \wedge b$ and *complementation*: $a \mapsto \bar{a}$, and distinct elements 0 and 1, is called a **Boolean algebra** if it satisfies the following laws, for all $x, y, z \in T$:

commutative:
- $x \vee y = y \vee x$
- $x \wedge y = y \wedge x$

associative:
- $(x \vee y) \vee z = x \vee (y \vee z)$
- $(x \wedge y) \wedge z = x \wedge (y \wedge z)$

distributive:
- $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
- $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

identity: $x \vee 0 = x, \quad x \wedge 1 = x$

complementation: $x \vee \overline{x} = 1, \quad x \wedge \overline{x} = 0$

Terminology and Rules

- A **literal** is an expression p or $\overline{p}$, where p is a propositional atom.
- An expression is in CNF (conjunctive normal form) if it has the form

$$\bigwedge_i C_i$$

  where each **clause** C_i is a disjunction of literals e.g. $p \vee q \vee \overline{r}$.
- An expression is in DNF (disjunctive normal form) if it has the form

$$\bigvee_i C_i$$

  where each clause C_i is a conjunction of literals e.g. $p \wedge q \wedge \overline{r}$.

- CNF and DNF are named after their top level operators; no deeper nesting of $\wedge$ or $\vee$ is permitted.
- We can assume in every clause (disjunct for the CNF, conjunct for the DNF) any given variable (literal) appears only once; preferably, no literal and its negation together.
  - $x \vee x = x, \quad x \wedge x = x$
  - $x \wedge \overline{x} = 0, \quad x \vee \overline{x} = 1$
  - $x \wedge 0 = 0$, $x \wedge 1 = x$, $x \vee 0 = x$, $x \vee 1 = 1$
- A preferred form for an expression is DNF, with as few terms as possible. In deriving such minimal simplifications the two basic rules are **absorption** and **combining the opposites**.

Fact

1. $x \vee (x \wedge y) = x$ *(absorption)*
2. $(x \wedge y) \vee (x \wedge \overline{y}) = x$ *(combining the opposites)*

Theorem

For every Boolean expression ϕ, there exists an equivalent expression in conjunctive normal form and an equivalent expression in disjunctive normal form.

Proof.

We show how to apply the equivalences already introduced to convert any given formula to an equivalent one in CNF, DNF is similar. □

Step 1: Push Negations Down

Using **De Morgan's** laws and the **double negation** rule

$$\overline{x \vee y} = \overline{x} \wedge \overline{y}$$
$$\overline{x \wedge y} = \overline{x} \vee \overline{y}$$
$$\overline{\overline{x}} = x$$

we push negations down towards the atoms until we obtain a formula that is formed from literals using only $\wedge$ and $\vee$.

Step 2: Use Distribution to Convert to CNF

Using the distribution rules

$$x \vee (y_1 \wedge \ldots \wedge y_n) = (x \vee y_1) \wedge \ldots \wedge (x \vee y_n)$$
$$(y_1 \wedge \ldots \wedge y_n) \vee x = (y_1 \vee x) \wedge \ldots \wedge (y_n \vee x)$$

we obtain a CNF formula.

CNF/DNF in Propositional Logic

Using the equivalence

$$A \rightarrow B \quad \equiv \quad \neg A \vee B$$

we first eliminate all occurrences of $\rightarrow$

Example

$$\neg(\neg p \wedge ((r \wedge s) \rightarrow q)) \equiv \neg(\neg p \wedge (\neg(r \wedge s) \vee q))$$

Step 1:

Example

$$\begin{aligned}\overline{p(\overline{rs} \vee q)} &= \overline{p} \vee \overline{\overline{rs} \vee q} \\ &= p \vee \overline{\overline{rs}} \wedge \overline{q} \\ &= p \vee rs\overline{q}\end{aligned}$$

Step 2:

Example

$$\begin{aligned}p \vee rs\overline{q} &= (p \vee r)(p \vee s\overline{q}) \\ &= (p \vee r)(p \vee s)(p \vee \overline{q}) \qquad \text{CNF}\end{aligned}$$

Canonical Form DNF

Given a Boolean expression E, we can construct an equivalent DNF E^{dnf} from the lines of the truth table where E is true:
Given an assignment π of $0, 1$ to variables $x_1 \ldots x_i$, define the literal

$$\ell_i = \begin{cases} x_i & \text{if } \pi(x_i) = 1 \\ \overline{x_i} & \text{if } \pi(x_i) = 0 \end{cases}$$

and a product $t_\pi = \ell_1 \cdot \ell_2 \cdot \ldots \cdot \ell_n$.

Example

If $\pi(x_1) = 1$ and $\pi(x_2) = 0$ then $t_\pi = x_1 \cdot \overline{x_2}$

The **canonical DNF** of E is

$$E^{dnf} = \sum_{E(\pi)=1} t_\pi$$

Example

If E is defined by

x	y	E
0	0	1
0	1	0
1	0	1
1	1	1

then $E^{dnf} = \overline{x}\overline{y} + x\overline{y} + xy$
Note that this can be simplified to either

$$\overline{y} + xy$$

or

$$\overline{xy} + x$$

Exercise

$\boxed{10.2.3}$ Find the canonical DNF form of each of the following expressions in variables x, y, z

- xy
- $\bar{z}$
- $xy + \bar{z}$
- 1

Exercise

$\boxed{10.2.3}$ Find the canonical DNF form of the following expressions Remember that these are meant as expressions in three variables x, y, z.

$$\begin{aligned} xy &= xy \cdot 1 = xy \cdot (z + \bar{z}) = xyz + xy\bar{z} \\ \bar{z} &= xy\bar{z} + x\bar{y}\bar{z} + \bar{x}y\bar{z} + \overline{xyz} \\ xy + \bar{z} &= \text{combine the 6 product terms above} \\ 1 &= \text{sum of all 8 possible product terms: } xyz + \bar{x}yz + \ldots + \overline{xyz} \end{aligned}$$

NB

Obviously, preferred in practice are the expressions with as few terms as possible.
However, the existence of a uniform representation as the sum of (quite a few) product terms is important for proving the properties of Boolean expressions.

Karnaugh Maps

For up to four variables (propositional symbols) a diagrammatic method of simplification called **Karnaugh maps** works quite well. For every propositional function of $k = 2, 3, 4$ variables we construct a rectangular array of 2^k cells. We mark the squares corresponding to the value 1 with eg "+" and try to cover these squares with as few rectangles with sides 1 or 2 or 4 as possible.

Example

$\boxed{10.4.2}$ Use a K-map to find an optimised form.

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x	+	+		+
$\bar{x}$	+		+	+

For optimisation, the idea is to cover the + squares with the minimum number of rectangles. One *cannot* cover any empty cells (they indicate where $f(w, x, y, z)$ is 0).

- The rectangles can go 'around the corner'/the actual map should be seen as a *torus*.
- Rectangles must have sides of 1, 2 or 4 squares (three adjacent cells are useless).

Example

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x	+	+		+
$\bar{x}$	+		+	+

$f = xy + \bar{x}\bar{y} + z$

Canonical form would consist of writing all cells separately:
$xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z$

For optimisation, the idea is to cover the $+$ squares with the minimum number of rectangles. One *cannot* cover any empty cells (they indicate where $f(w, x, y, z)$ is 0).

- The rectangles can go 'around the corner'/the actual map should be seen as a *torus*.
- Rectangles must have sides of 1, 2 or 4 squares (three adjacent cells are useless).

Example

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
x	$+$	$+$		$+$
$\bar{x}$	$+$		$+$	$+$

$f = xy + \bar{x}\bar{y} + z$

Canonical form would consist of writing all cells separately:
$xyz + xy\bar{z} + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z$

Supplementary Exercise

$\boxed{10.6.6(c)}$

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx	$+$	$+$		$+$
$w\bar{x}$	$+$	$+$	$+$	$+$
$\bar{w}\bar{x}$			$+$	$+$
$\bar{w}x$	$+$			$+$

Supplementary Exercise

$\boxed{10.6.6(c)}$

	yz	$y\bar{z}$	$\bar{y}\bar{z}$	$\bar{y}z$
wx	$+$	$+$		$+$
$w\bar{x}$	$+$	$+$	$+$	$+$
$\bar{w}\bar{x}$			$+$	$+$
$\bar{w}x$	$+$			$+$

$f = wy + \bar{x}\bar{y} + xz$

Note: trying to use $w\bar{x}$ or $\bar{y}z$ doesn't give as good a solution

Boolean Algebras in Computer Science

Several data structures have natural operations following essentially the same rules as logical $\wedge$, $\vee$ and $\neg$.

- n-tuples of 0's and 1's with Boolean operations, e.g.

$$\begin{aligned} \textit{join:} \quad & (1,0,0,1) \vee (1,1,0,0) = (1,1,0,1) \\ \textit{meet:} \quad & (1,0,0,1) \wedge (1,1,0,0) = (1,0,0,0) \\ \textit{complement:} \quad & \overline{(1,0,0,1)} = (0,1,1,0) \end{aligned}$$

- $\mathrm{Pow}(S)$ — subsets of S

$$\textit{join: } A \cup B, \quad \textit{meet: } A \cap B, \quad \textit{complement: } A^c = S \setminus A$$

Example

$\boxed{10.1.1}$ Define a Boolean algebra for the power set $\mathsf{Pow}(S)$ of $S = \{a, b, c\}$

join: $X, Y \mapsto X \cup Y$
meet: $X, Y \mapsto X \cap Y$
complementation: $X \mapsto \{a, b, c\} \setminus X$
$0 \stackrel{\text{def}}{=} \emptyset$
$1 \stackrel{\text{def}}{=} \{a, b, c\}$

Exercise:
Verify that all Boolean algebra laws (cf. slide 65) hold for $X, Y, Z \in \mathsf{Pow}(\{a, b, c\})$

Example

$\boxed{10.1.1}$ Define a Boolean algebra for the power set $\mathsf{Pow}(S)$ of $S = \{a, b, c\}$

join: $X, Y \mapsto X \cup Y$
meet: $X, Y \mapsto X \cap Y$
complementation: $X \mapsto \{a, b, c\} \setminus X$
$0 \stackrel{\text{def}}{=} \emptyset$
$1 \stackrel{\text{def}}{=} \{a, b, c\}$

Exercise:
Verify that all Boolean algebra laws (cf. slide 65) hold for $X, Y, Z \in \mathsf{Pow}(\{a, b, c\})$

More Examples of Boolean Algebras in CS

- Functions from any set S to $\mathbb{B}$; their set is denoted $\mathsf{Map}(S, \mathbb{B})$

  If $f, g : S \longrightarrow \mathbb{B}$ then
  - $(f \vee g) : S \longrightarrow \mathbb{B}$ is defined by $s \mapsto f(s) \vee g(s)$
  - $(f \wedge g) : S \longrightarrow \mathbb{B}$ is defined by $s \mapsto f(s) \wedge g(s)$
  - $\overline{f} : S \longrightarrow \mathbb{B}$ is defined by $s \mapsto \overline{f(s)}$

  There are 2^n such functions for $|S| = n$

- All Boolean functions of n variables, e.g.

$$(p_1, p_2, p_3) \mapsto (p_1 \vee \overline{p_2}) \wedge (p_1 \vee p_3) \wedge \overline{p_2 \vee p_3}$$

  There are 2^{2^n} of them; their collection is denoted $\textsc{bool}(n)$

Every finite Boolean algebra satisfies: $|T| = 2^k$ for some k.
All algebras with the same number of elements are **isomorphic**, i.e. “structurally similar”, written $\simeq$. Therefore, studying one such algebra describes properties of all.
A cartesian product of Boolean algebras is again a Boolean algebra. We write

$$\mathbb{B}^k = \mathbb{B} \times \ldots \times \mathbb{B}$$

The algebras mentioned above are all of this form

- n-tuples $\simeq \mathbb{B}^n$
- $\mathsf{Pow}(S) \simeq \mathbb{B}^{|S|}$
- $\mathsf{Map}(S, \mathbb{B}) \simeq \mathbb{B}^{|S|}$
- $\textsc{bool}(n) \simeq \mathbb{B}^{2^n}$

NB
Boolean algebra as the calculus of two values is fundamental to computer circuits and computer programming.
Example: Encoding subsets as bit vectors.

Summary

- Logic: syntax, truth tables; $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\top$, $\bot$
- Valid formulae (tautologies), satisfiable formulae
- Entailment $\models$, equivalence $\equiv$
  some well-known equivalences (slides 23 and 24)
- Proof methods: contrapositive, by contradiction, by cases
- Boolean algebra
- CNF, DNF, canonical form

Supplementary reading [LLM]

- Ch. 1, Sec. 1.5-1.9 (more about good proofs)
- Ch. 3, Sec. 3.3 (more about proving equivalences of formulae)