CS 5002: Discrete Structures **Fall 2018**

Lecture 1: September 6, 2018

Lecturer: Adrienne Slaughter, Tamara Bonaci *Scribes: scribe-name1,2,3*

1.1 Overview

1. Logic: Operators, propositions, equivalences, predicates, quantifiers.
2. Number representation and base expansion
3. Why all this matters

1.2 Introduction

In today's lecture, we will introduce the field of mathematical logic, a subfield of mathematics that explores the applications of formal logic to mathematics. We will then consider one of its applications - number systems and number representations.

1.3 Logic, mathematical logic, and logic in computer science

> Logic is a science of the necessary laws of thought, without which no employment of the understanding and the reason takes place. Immanuel Kant

It is often considered that the roots of the modern logic stem from a Greek philosopher Aristotle, who developed the first collection of rules of deductive reasoning that were intended to serve as a basis for the study of every branch of knowledge. Aristotle's ideas, however, remained dormant until the seventeenth century, when the German philosopher and mathematician Gottfried Leibniz proposed the idea of using symbols to simplify the process of deductive reasoning, in a similar way as algebraic notation is being used to reason about numbers and their relationship.

Leibniz's idea was realized by the English mathematicians George Boole and Augustus De Morgan in the nineteenth century, when they invented the field of symbolic logic. From there, mathematical logic was born. The expressive power of formal systems, and the deductive power of formal proof system is often considered the unifying theme of mathematical logic.

Mathematical logic is often divided into the fields of set theory, model theory, recursion theory, and proof theory, and in this course, we will talk about all of these theories. Today, however, we will start by focusing on logical form and logical equivalence.

1.3.1 Definitions

Definition: Proposition

A ***proposition*** is a statement that is true or false, but not both.

Examples of Propositions:

- $2 + 2 = 4$ **TRUE**
- Seattle is in Washington. **True**
- $3 \times 6 = 15$ **False**
- Seattle is in California. **False**

Practice

Which of the following are propositions?

1. What's your favorite movie? **No**
2. Read the newspaper. **No**
3. $x^2 + y^2 = z^2$ **Yes**
4. Seattle is in California. **Yes**

Just like with algebra, letters are used to represent propositions:

- p: $2 + 2 = 4$
- q: Seattle is in California.

Definition: Truth Value

The ***truth value*** of a proposition is true (**T**) if it is a true proposition and false (**F**) if it is false.

Definition: Truth Table

A ***truth table*** displays the relationship between the truth values of propositions.

p	$\neg p$
T	F
F	T

This truth table introduces the ***negation*** operator: $\neg$

1.3.2 Logic Operators

Definition: Negation

Represented by the symbol: $\neg$

The proposition $\neg p$ is false when p is true, and true when p is false.

p	$\neg p$
T	F
F	T

A ***conjunction*** is a fancy word for ***and***.

Represented by the symbol: $\wedge$

The proposition p and q is $p \wedge q$ and is *true* when p and q are true, and false otherwise.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

p: Sam left the party.

q: Pat arrived at the party.

$p \wedge q$

p	q	$p \wedge q$	
T	T	T	Sam left the party and Pat arrived at the party.
T	F	F	~~Sam left the party; Pat did not arrive at the party.~~
F	T	F	~~Sam did not leave the party; Pat arrived at the party.~~
F	F	F	~~Sam did not leave the party; Pat did not arrive at the party.~~

Disjunction is a fancy word for ***or***.

Represented by the symbol: $\vee$

The proposition p or q (written by $p \vee q$) is *false* when p and q are false, and true otherwise.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

OR example

p: It is Wednesday.

q: I am preparing lecture.

$p \vee q$

p	q	$p \vee q$	
T	T	T	It is Wednesday and I am preparing lecture.
T	F	T	It is Wednesday but I am not preparing lecture.
F	T	T	It is not Wednesday, but I am preparing lecture.
F	F	F	~~It is not Wednesday, and I am not preparing lecture.~~

Sometimes I don't prepare lecture on Wednesdays, but if it's not Wednesday, then I am preparing lecture.

This is also known as ***inclusive OR***: this, or that, or both.

Sometimes it's necessary to denote the notion that something is true if p XOR q is true. This is called **exclusive or**: this or that, but not both.

Represented by the symbol: $\oplus$

The proposition p xor q (written by $p \oplus q$) is *true* when p is true OR q is true, but not when p and q are either both true or both false.

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

XOR example

p: It is Wednesday.

q: I am preparing lecture.

$p \oplus q$

p	q	$p \oplus q$	
T	T	F	~~It is Wednesday and I am preparing lecture.~~
T	F	T	It is Wednesday but I am not preparing lecture.
F	T	T	It is not Wednesday, but I am preparing lecture.
F	F	F	~~It is not Wednesday, and I am not preparing lecture.~~

It is either Wednesday, or I am preparing lecture; it is never the case that I am preparing lecture on Wednesday.

Conditional (Implication)

The proposition $p \to q$ is false when p is true and q is false, and true otherwise.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

p is sometimes considered ***hypothesis***, *antecedent, or premise.* q is sometimes called the ***conclusion*** *or consequence.*

Implication examples

p: The weather report says rain.

q: I wear my rain coat.

$p \to q$

p	q	$p \to q$	
T	T	T	The report said rain, so I wore my coat.
T	F	F	~~The report said rain, so I didn't wear my coat.~~
F	T	T	The report said sun, but I wore my coat.
F	F	T	The report said sun, and I didn't wear my coat.

When p is true, q is true.

When p is false, q might be either true or false, and it doesn't matter.

In this example, when the report says rain, I *always* wear my rain coat; but if the report does not say rain, I may or may not wear my rain coat.

Biconditional: The proposition $p \leftrightarrow q$ is true when p is true and q is true,
or p is false and q is false, false otherwise.

p	q	$p \to q$
T	T	T
T	F	F
F	T	F
F	F	T

Biconditional example

p: The weather report says rain.

q: I wear my rain coat.

$p \leftrightarrow q$

p	q	$p \leftrightarrow q$	
T	T	T	The report said rain, so I wore my coat.
T	F	F	~~The report said rain, so I didn't wear my coat.~~
F	T	F	~~The report said sun, but I wore my coat.~~
F	F	T	The report said sun, and I didn't wear my coat.

When p is true, q is true.

When p is false, q is false.

In this example, when the report says rain, I *always* wear my rain coat; but if the report does not say rain, I *never* wear my rain coat.

This is an if and only if: I wear my coat if and only if the report says rain.

1.3.3 Evaluating Propositions

Grouping and Order of Operations

Parentheses are used to specify grouping. Innermost statements are evaluated first. The negation operator is applied before all other logical operators.

$\neg p \wedge q$ is the same as $(\neg p) \wedge q$

NOT: $\neg(p \wedge q)$

Compound Propositions

The proposition $q \rightarrow p$ is the ***converse*** of $p \rightarrow q$.

The ***contrapositive*** of $p \rightarrow q$ is $\neg q \rightarrow \neg p$.

Example: Given the implication:

"If today is Thursday, then I have a test today"

The converse is: "If I have a test today, then today is Thursday"

The contrapositive is: "If I do not have a test today, then today is not Thursday"

1.3.4 Summary: Logical Operators

Symbol	said	name	description
$\neg p$	not p	negation	true when p is false.
$p \wedge q$	p and q	conjunction	true when p and q are true.
$p \vee q$	p or q	disjunction	false when p and q are false.
$p \rightarrow q$	p implies q	implication	true when p is false or q is true.
$p \leftrightarrow q$	p if and only if q	biconditional	true when both p and q are the same.
$p \oplus q$	p xor q	exclusive or	true when p and q are different.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \oplus q$
T	T	F	T	T	T	T	F
T	F		F	T	F	F	T
F	T	T	F	T	T	F	T
F	F		F	F	T	T	F

1.3.5 Logical Equivalences

Logical Equivalence

We can evaluate the ***logical equivalence*** of two propositions by comparing the truth tables of each statement.

Prove the following two statements are logically equivalent:

$\neg p \vee q$ and $p \rightarrow q$

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Example: Prove the following two statements are logically equivalent:

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

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

1.3.6 Predicates and Quantifiers

Is this a proposition?

$p : x^2 + y^2 = z^2$

Well... kinda.

$P(x, y, z) : x^2 + y^2 = z^2$

Back at the beginning, I said this is a proposition. It's not really– it's actually something called a predicate.

Predicates:

$$P(x) : x > 3$$

- If $x = 5$, $P(x) = P(5) = 5 > 3 \Rightarrow true$
- If $x = 1$, $P(x) = P(1) = 1 > 3 \Rightarrow false$

At this point, you should be thinking: But, we can define some rules that define when P is true and false.

Quantifiers:

Universal quantifier: For All

For every value in the *universe of discourse*, the predicate P is true. $P(x)$ is true for every value of x is in the *universe of discourse.*

$\forall x P(x)$

Existential quantifier: There exists

There exists a value in the universe of discourse such that P is true.

$P(x)$ is true for some value of x in the universe of discourse.

$\exists x P(x)$

Quantifiers: Example

"All students in this class are in the ALIGN program. "

Let: $P(x)$: x is in the ALIGN program.

Define the **universe of discourse** as "students in this class", or $S(x)$.

$\Rightarrow \forall x S(x) \rightarrow P(x)$

Quantifiers: Example

"Some students in this class live in Seattle"

Let: $P(x)$: x lives in Seattle.

Define the **universe of discourse** as "students in this class", or $S(x)$'.

$\Rightarrow \exists x S(x) \rightarrow P(x)$

Quantifiers: Example

Going back to our example of $P(x) : x > 3$

Define the **universe of discourse** as "integers", or $N(x)$.

$\Rightarrow \exists x N(x) \rightarrow P(x)$

An Example[1]

$$P(x) : x \text{ is a professor}$$
$$Q(x) : x\text{is ignorant}$$
$$R(x) : x\text{is vain}$$

Express each of the statements using quantifiers and logical operators, where the universe of discourse is the set of all people.

[1]Based on *Symbolic Logic* by Lewis Carroll

1. No professors are ignorant. $\forall x(P(x) \rightarrow \neg Q(x))$
2. All ignorant people are vain.$\forall x(Q(x) \rightarrow R(x))$
3. No professors are vain. $\forall x(P(x) \rightarrow \neg R(x))$
4. Does (3) follow from (1) and (2)?
   No. P implies not Q; P implies not R; and Q implies R, but no statement has been made about whether P implies R. The statements do not rule out the possibility of other vain people that are not ignorant.

Another Example[2]

$$\begin{aligned} P(x) &: x \text{ is a baby} \\ Q(x) &: x \text{ is logical} \\ R(x) &: x\text{is able to manage a crocodile} \\ S(x) &: x \text{ is despised} \end{aligned}$$

Express each of the statements using quantifiers, logical connectives and the statements, where the universe of discourse is the set of all people.

1. Babies are illogical.$\forall x(P(x) \rightarrow \neg Q(x))$
2. Nobody is despised who can manage a crocodile. $\forall x(R(x) \rightarrow \neg S(x))$
3. Illogical persons are despised. $\forall x(\neg Q(x) \rightarrow S(x))$
4. Babies cannot manage crocodiles. $\forall x(P(x) \rightarrow \neg R(x))$
5. Does (4) follow from (1), (2) and (3)?
   Yes.
   If x is a baby, then by (1), x is illogical. By (3), x is despised. (2) says that if x could manage a crocodile, then x would not be despised. Therefore, x cannot manage a crocodile.

Summary

- Propositions
- Truth Values
- Truth Tables
- Logical operators (sometimes called Logic Connectives), to create compound statements
- Combining/ordering of operators
- Predicates
- Quantifiers: when predicates are true or false
- Universe of discourse

1.4 Number Representations

Generally, we use *decimal* notation to express integers. In decimal notation, a number 1024 can be expressed as:

$$(1 \cdot 1000) + (0 \cdot 100) + (2 \cdot 10) + (4 \cdot 1) \text{ or } (1 \cdot 10^3) + (0 \cdot 10^2) + (2 \cdot 10^1) + (4 \cdot 10^0)$$

As a review, *decimal* is ***base 10***.

Arbitrary base: Base 5

[2]Also based on *Symbolic Logic* by Lewis Carroll

$$(1 \cdot 10^3) + (0 \cdot 10^2) + (2 \cdot 10^1) + (4 \cdot 10^0)$$

$$\begin{aligned}(1 \cdot 5^3) + (0 \cdot 5^2) + (2 \cdot 5^1) + (4 \cdot 5^0) &= (125) + (10) + 4 \\ &= (139)_5\end{aligned}$$

Counting in Different bases:

Decimal	Binary	Base 5	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	10	5
6	110	11	6
7	111	12	7
8	1110	13	8
9	1000	14	9
10	1001	20	A
11	1010	21	B
12	1011	22	C
13	1100	23	D
14	1101	24	E
15	1110	30	F
16	10000	31	10
17	10001	32	11
18	10010	33	12

1.4.1 Base Expansion

Base b expansion of n

Theorem 1.1 *If b is a positive integer greater than 1, and n is a positive integer, it can be expressed in the form:*

$$n_{10} = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0$$

where k is a nonnegative integer, the number of digits in n

In decimal notation, a number 1024 can be expressed as:

$$(1 \cdot 1000) + (0 \cdot 100) + (2 \cdot 10) + (4 \cdot 1) \text{ or } (1 \cdot 10^3) + (0 \cdot 10^2) + (2 \cdot 10^1) + (4 \cdot 10^0)$$

As a review, *decimal* is ***base 10***.

(This is what we use to go FROM base b TO base 10.)

Bases

Binary: Base 2

Hexadecimal: Base 16

Octal: Base 8

When base $b > 10$, we need more symbols than the digits $0 - 9$ to represent each value. In hexadecimal, we use the letters A-F to represent the values 10-15.

Binary expansion of integer n

When base $b = 2$, it is called *binary* expansion.

If n is a positive integer, it can be expressed in the form:

$$n_{10} = a_k 2^k + a_{k-1} 2^{k-1} + ... + a_1 2 + a_0$$

where k is a nonnegative integer, which is the number of digits in n.

Integers in base 2 use only the digits 1 and 0.

(This is what we use to go FROM base 2 TO base 10.)

Hexadecimal expansion of n

When base b is 16 and n is positive, it can be expressed in the form:

$$n_{10} = a_k 16^k + a_{k-1} 16^{k-1} + ... + a_1 16 + a_0$$

where k is a nonnegative integer, which is the number of digits in n. Integers in base 16 use the digits 0-9 and the letters A-F.

(This is what we use to go FROM base 16 TO base 10.)

Example: Hexadecimal Expansion

Problem: Express the number $(2A0F1)_{16}$ in decimal.

Solution: Use the hexadecimal expansion formula.

$$\begin{gathered}(2 \cdot 16^4) + (A \cdot 16^3) + (0 \cdot 16^2) + (F \cdot 16^1) + (1 \cdot 16^0) = \\ (2 \cdot 65536) + (A \cdot 4096) + (0 \cdot 256) + (F \cdot 16) + (1 \cdot 1) = \\ (2 \cdot 65536) + (10 \cdot 4096) + (0 \cdot 256) + (15 \cdot 16) + (1 \cdot 1) = (172273)_{10}\end{gathered}$$

Going from Base 10 to Base b

To construct the base b expansion of integer n, first divide n by b to obtain a quotient and remainder. That is:

$$n = bq_0 + a_0, 0 \leq a_0 < b$$

The remainder (a_0) is the rightmost (least significant) digit in the base b expansion on n. Next, divide q_0 by b to obtain

$$q_0 = bq_1 + a_1, 0 \leq a_1 < b$$

a_1 is the second digit from the right in the base b expansion of n. Continue this until you obtain a quotient equal to 0.

Problem: Convert $(12345)_{10}$ to base 8.

Solution: First, divide 12345 by 8 to obtain:

$$\begin{aligned}
12345 &= 8 \times 1543 + 1 \textit{(The right-most digit)}\\
1543 &= 8 \times 192 + 7\\
192 &= 8 \times 24 + 0\\
24 &= 8 \times 3 + 0\\
3 &= 8 \times 0 + 3 \textit{(The left-most digit)}
\end{aligned}$$

This results in:

$(12345)_{10} = (30071)_8$

We've talked about different bases. One significant effect of base value is the number of values that can be represented by a given number of digits.

Number of values represented

For base b and digit length k, the maximum number of values that can be represented is b^k. We'll go into more of the math behind this later this semester.

Example: Limits

Assume we have a positive integer of length 4.

In decimal:

- What is the maximum value we can represent? 9999
- What is the minimum value we can represent? 0000
- $\Rightarrow$ 10,000 values, or 10^4 unique values.

In binary:

- 16, or 2^4 unique values.

In hexadecimal:

- 65,536, or 16^4 unique values.

Hopefully the decimal example helps. Also, you can work with a 3-digit number in base 2 (2^3 or 8 unique values) to convince yourself.

Summary: Number Representation

Not including details on adding and multiplying in various bases.

- Number representations
- Base 10 (decimal), base 2 (binary), base 16 (hexadecimal)
- Going from Base 10 to base b
- Going from Base b to base 10
- Converting between base 2 and base 16 (binary and hexadecimal)
- Maximum number of values that can be represented

1.5 Relevance to Programming

1.5.1 Binary in Computers

Binary is important in CS because numbers are all represented in binary. Data is stored, transmitted and processed via a series of electrical pulses; high for 1, low for 0.

8- vs 16-bit numbers: What's the largest value an 8-bit binary number can have?

In this case, $k = 8$, and our formula gives us:

$$n = 2^8 = 256$$

Binary in Computers

These days, data is usually stored in a 16-bit binary number. For display and discussion purposes, I'll be using an 8-bit binary number. But so far, we have only talked about positive integers. If this is the case, how do we represent negative numbers?

In math, or the non-computer world, we just put a negative sign in front. But in the machine world, what's the negative sign?

I could provide a longer explanation about how computers are built out of circuits and how we could use various protocols and such, but really it's just easier to encode the negative sign within the number representation itself.

2's Complement: Definition

This is where 2's complement comes in:

Definition: 2's complement

In 2's complement representation of binary numbers, the *most significant bit* is considered negative:

$$n = -a_k 2^k + a_{k-1} 2^{k-1} + ... + a_1 2 + a_0$$

Where k is the number of bits in the number.

2's Complement: Explanation

Using the definition from the previous slide, an 8-bit binary number using 2's complement will have the value:

$$n = -a_7 2^7 + a_6 2^6 + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2 + a_0$$

Recall, since this is binary, each a_k is either 1 or 0. That means, each term is either present or not.

If all the terms are present (that is, all the a_k are 1, or the binary number is 11111111), the value is:

$$\begin{aligned} n &= -1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1 \\ &= -1 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \\ &= -128 + 127 \\ &= -1 \end{aligned}$$

2's Complement: Practice

Spend a few minutes and answer the following questions:

- In 2's complement, which of the numbers below has the ***greatest*** value?
  1. 1111 1111
  2. 1000 0000
  3. 0000 0000
  4. 0111 1111
- In 2's complement, which of the numbers below has the ***least*** value?
  1. 1111 1111
  2. 1000 0000
  3. 0000 0000
  4. 0111 1111

1.5.2 Representing Negative Numbers

2's Complement: Relevance to Python

In Python, when you're looking at documentation, you'll see that different types have different MAX and MIN values. While the values vary according to implementation and machine, here are some representative values:

type	storage size (# bits)	min value	max value
char	8 bits (1 byte)	-127	128
unsigned char	8 bits	0	255
signed char	8 bits	-127	128
int	16 bits[3]	-32,768	32,767
unsigned int	16 bits[4]	0	65,535
short	16 bits (2 bytes)	-32,768	32,767
unsigned short	16 bits	0	65,535
long	32 bits (4 bytes)	-2,147,483,648	2,147,483,647
unsigned long	32 bits	0	4,294,967,295

1.5.3 Logic in Computers

Logic and Bit Operators

bit operations

p	q	NOT p	p AND q	p OR q	p XOR q
1	1	0	1	1	0
1	0		0	1	1
0	1	1	0	1	1
0	0		0	0	0

T/F and Bits

OR, AND, XOR

Logic and Bit Operators

bitwise OR, bitwise AND, bitwise XOR

107	0110	1011	
206	1100	1110	
239	1110	1111	bitwise OR
74	0100	1010	bitwise AND
165	1010	0101	bitwise XOR
148	1001	0100	bitwise NOT

Summary: Relevance to Computing

- Why we use binary in computing
- How to represent negative numbers in binary
- How number representation impacts integer values in C
- Logic and bits: AND, OR, NOT, XOR