1 Monday, January 12

Announcements: Course Syllabus, Homework 01 Due Jan 15.

Reading: Rosen: 1.1, 1.2, 1.3. LLM: 1.1–1.2, 3.1–3.4

1.1 Rosen 1.1 — Propositional Logic

Def: A *proposition* is a declarative sentence (a sentence that declares a fact) that is either true or false, never both.

Ex: "Ames is a city in Iowa." "Today is Monday." "$2 \times 3 = 6$." "$2 \times 2 = 5$."

Non-Ex: "How are you doing?" "Go outside, for once!" "$3x = 6$." "$x + y = z$."

Use letters (such as $x, y, z, p, q, r, s, \dots$) to denote *propositional variables*, which symbolize some proposition (and whether or not it is true).

The *truth value* is either *true* (**T**) or *false* (**F**).

Def: The *propositional calculus* or *propositional logic* is a way to combine propositions to create other propositions.

Def: If p is a proposition, then the *negation* of p, denoted $\neg p$ or $\overline{p}$, is the statement

"It is not the case that p."

Def: A *truth table* expresses the truth value of a compound proposition given every possible combination of truth values for its propositional variables.

p	$\neg p$
T	**F**
F	**T**

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

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

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

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

Table: Truth tables for $\neg p$, $p \wedge q$, $p \vee q$, $p \oplus q$, and $p \to q$.

Def: Let p and q be propositions. The *conjunction* of p and q, denoted $p \wedge q$ and said "p and q," is true exactly when p and q are both true. The *disjunction* of p and q, denoted $p \vee q$ and said "p or q," is true exactly when at least one of p or q is true.

Note: If you say "Will you have the soup or the salad?" in English, you are typically saying you can have *exactly one* of the soup or the salad. However, in the disjunction $p \vee q$, it is still true if BOTH p and q are true. This is called *inclusive or*. When you want to specify the statement "exactly one of p and q is true," then you want the *exclusive or*, denoted $p \oplus q$.

Def: Let p and q be propositions. The *conditional statement* $p \to q$ is the proposition "if p, then q." This proposition is true only when q is true whenever p is true. There are many ways to describe this relationship (see Rosen, p. 6 for a list).

A conditional statement is sometimes called an *implication*.

"If your final weighted score is above 90%, then you will receive an A in this course."

The idea of "$p \to q$" is that p is a condition, and only when that condition is true can you say something for sure about q! If p is false, then the conditional statement "$p \to q$" is *vacuously true*.

"If pigs have wings, then pigs can fly."

"If I am a hat, then I am the president."

Note: Do not confuse the "if p then q" phrasing too much with the if/then constructions in programming (but they are related!).

Given a conditional statement "$p \to q$" there are a few ways to modify it (that are worth remembering):

1. *converse* : "$q \to p$" (sometimes written "$p \leftarrow q$").
2. *contrapositive* : "$\neg q \to \neg p$
3. *inverse* : "$\neg p \to \neg q$

See the following truth table to see that these do not always agree with the conditional statement.

		Conditional	Converse	Contrapositive	Inverse	Biconditional
p	q	$p \to q$	$q \to p$	$\neg q \to \neg p$	$\neg p \to \neg q$	$p \leftrightarrow q$
T	**T**	**T**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**T**	**F**
F	**T**	**T**	**F**	**T**	**F**	**F**
F	**F**	**T**	**T**	**T**	**T**	**T**

Table: Truth table for variations on conditional statements. Observe that these conditionals each false in exactly one situation: where p and q differ (and which of these two cases is different in some situations).

Def: Let p and q be propositions. The *biconditional* statement $p \leftrightarrow q$ is the proposition "p if and only if q" and is true when p and q have the same truth value.

1.1.1 Recommended Homework

Rosen 1.1: 1–5, 8–15, 22–28.

2 Wednesday, January 14

Announcement: Homework 01 due Thursday, Jan 15th.

Correction/Reminder: AND ($\wedge$) is a *conjunction*, OR ($\vee$) is a *disjunction*. Write out truth tables of all symbols.

Now that we are armed with our basic operators, we can make more complicated propositions, called *compound propositions*. We can blindly construct these, and their truth value depends on the truth values of its constituent propositions. (We use parentheses to nest these operations, so you know which inner propositions are grouped.)

Example: Construct the truth table for the compound proposition $(p \to q) \wedge (\neg p \to \neg q)$.

Example: Construct the truth table for the compound proposition $(p \to q) \vee (\neg q \to p)$.

Example: Construct the truth table for the compound proposition $(\neg p \wedge q) \vee (p \wedge \neg q)$.

Note: When you construct a truth table with n propositional variables, you will have $2^n = \underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}}$ rows. This is because there are two options for each of the n variables, and these choices are made "independently." See this video on constructing truth tables, specifically how to generate all combinations of truth values.

Precedence of Logical Operators: Just as $-$ (negation), $+$ and $-$ (addition and subtraction), $*$ (multiplication), and / (division) have different orders of importance when considering a complicated algebraic phrase, so too do logical operators. The orders can be confusing, so we will use parentheses as often as possible to prevent this being an issue.

Order of precedence: $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$.

Logic and Binary Representation: One last thing: In a computer, we use 0 and 1 as a "bit". This is the most basic computing element. We can treat 1 as **T** and 0 as **F** and find an equivalence in terms. But more importantly, if we perform bit-wise operations on a string of bits, then we are determining the compound proposition for each combination of bits.

Example:

$$\begin{array}{r}
\mathbf{x} = 101101 \\
\mathbf{y} = 011001 \\
\hline
\mathbf{x} \wedge \mathbf{y} = 001001 \\
\mathbf{x} \vee \mathbf{y} = 111101 \\
\mathbf{x} \oplus \mathbf{y} = 110100
\end{array}$$

2.0.2 Recommended Homework

Roesn 1.1: 31–39, 43–44.

2.1 Rosen 1.2 — Applications of Propositional Logic

Application 1: Translating English Sentences. (Important for lawyers in particular!)

Application 2: Systems Specifications. (Don't forget about testing!)

Application 3: Logic Puzzles.

http://www.logic-puzzles.org/init.php

The Knights and Knaves (see http://en.wikipedia.org/wiki/Knights_and_Knaves) problem is about two types of people: Knights always say the truth, and Knaves always lie. Based on your interaction, you need to determine if they are a Knight or a Knave and also possibly some other information.

> You come to a fork in the road. The two ways are guarded by two people. There is a sign saying "Exactly one of these guards is a Knight, and the other is a Knave. Exactly one path leads to your goal, the other leads to certain death."
>
> "Would you answer "Yes" if I asked you "Does your path lead to freedom?""
>
> "Would the other one answer "Yes" if I asked them "Does your path lead to freedom?""

The Boxes from http://www.folj.com/puzzles/

> There are three boxes. One is labeled "APPLES" another is labeled "ORANGES". The last one is labeled "APPLES AND ORANGES". You know that each is labeled incorrectly. You may ask me to pick one fruit from one box which you choose.
>
> How can you label the boxes correctly?

Example: See book for "Muddy Children Puzzle"

Application 4: Paradoxes.

Liar Paradox: Suppose a Knight or a Knave says "I always lie." Is the person a Knight or a Knave?

Alt 1: "This sentence is false."

Alt 2: "The next sentence is false." "The previous sentence is true."

Application 5: Logical Circuits.

Using NOT, OR, and AND gates, we can construct circuits to compute compound propositions.

Can we make our circuits more efficient? Fewer gates? Smaller depth? We should not modify the behavior while we do this! This means we must develop methods to *demonstrate equivalence between logical statements*.

2.1.1 Recommended Homework

Rosen 1.2: 1–5, 7–12, 15–18, 19–23, 24–31, 40–43.

Explore http://en.wikipedia.org/wiki/Category:Logic_puzzles

3 Friday, January 16

3.1 Rosen 1.3 — Propositional Equivalences

Question: Given two complex propositional statements, can you tell if they always evaluate to the same value?

Alternative viewpoint: Given two logic circuits, do they always output the same value?

One method: Exhaustively list the truth table. Very difficult in general!

Def: A compound proposition that is always true (no matter what truth values of the propositional variables) is a *tautology.*

Examples: $b \vee \neg b$. $(p \wedge \neg p) \to (q \wedge \neg q)$.

Def: A compound proposition that is always false (no matter what truth values of the propositional variables) is a *contradiction.*

Examples: $b \wedge \neg b$. $(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$.

Def: A proposition that is neither a tautology or a contradiction is a *contingency.*

Def: The compound propositions p and q are called *logically equivalent* if $p \leftrightarrow q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.

DeMorgan's Laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Extended DeMorgan's Laws
$\neg(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \equiv \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n$
$\neg(p_1 \vee p_2 \vee \ldots \vee p_n) \equiv \neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n$

Extended DeMorgan's Laws (Concise)
$\neg \bigwedge_{i=1}^n p_i \equiv \bigvee_{i=1}^n (\neg p_i)$
$\neg \bigvee_{i=1}^n p_i \equiv \bigwedge_{i=1}^n (\neg p_i)$

We will use the notation

$$\bigvee_{i=1}^n p_i \text{ to denote } p_1 \vee p_2 \vee \ldots \vee p_n,$$

and

$$\bigwedge_{i=1}^n p_i \text{ to denote } p_1 \wedge p_2 \wedge \ldots \wedge p_n,$$

the same way we use

$$\sum_{i=1}^n x_i \text{ to denote } x_1 + x_2 + \cdots + x_n.$$

For more logical equivalences, see the Logic Cheatsheet!

Constructing New Logical Equivalences

We can create logical equivalences by brute-force using the truth table, but that becomes very hard and boring. Let's instead use a very limited form of *proof!*

Essentially, we use our list of logical equivalences to translate from one statement to the other!

Example: Demonstrate the equivalence $(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$.

$$\begin{aligned}
(p \to r) \lor (q \to r) &\equiv (\neg p \lor r) \lor (q \to r) && \text{Logical equivalence of conditional statement}\\
&\equiv (\neg p \lor r) \lor (\neg q \lor r) && \text{Logical equivalence of Conditional statement}\\
&\equiv (\neg p \lor \neg q) \lor (r \lor r) && \text{Commutative law}\\
&\equiv (\neg p \lor \neg q) \lor r && \text{Idempotent law}\\
&\equiv \neg(p \land q) \lor r && \text{DeMorgan's law}\\
&\equiv (p \land q) \to r && \text{Logical equivalence of conditional statement}
\end{aligned}$$

Let p be a compound proposition. We could demonstrate that p is a tautology by demonstrating the equivalence $p \equiv \mathbf{T}$. We could demonstrate that p is a contradiction by demonstrating the equivalence $p \equiv \mathbf{F}$.

Example: Demonstrate that $(a \lor b) \land (\neg a \lor c) \to (b \lor c)$ is a tautology.

$$\begin{aligned}
(a \lor b) \land (\neg a \lor c) \to (b \lor c) &\equiv \neg\,[(a \lor b) \land (\neg a \lor c)] \lor (b \lor c) && \text{Logical equivalence of conditional statement}\\
&\equiv \neg(a \lor b) \lor \neg(\neg a \lor c) \lor (b \lor c) && \text{DeMorgan's Law}\\
&\equiv (\neg a \land \neg b) \lor \neg(\neg a \lor c) \lor (b \lor c) && \text{DeMorgan's Law}\\
&\equiv (\neg a \land \neg b) \lor (\neg(\neg a) \land \neg c) \lor (b \lor c) && \text{DeMorgan's Law}\\
&\equiv (\neg a \land \neg b) \lor (a \land \neg c) \lor (b \lor c) && \text{Double Negation Law}\\
&\equiv (\neg a \land \neg b) \lor b \lor (a \land \neg c) \lor c && \text{Commutative Law}\\
&\equiv [(\neg a \lor b) \land (\neg b \lor b)] \lor (a \land \neg c) \lor c && \text{Distributive Law}\\
&\equiv [(\neg a \lor b) \land (\neg b \lor b)] \lor [(a \lor c) \land (\neg c \lor c)] && \text{Distributive Law}\\
&\equiv [(\neg a \lor b) \land \mathbf{T}] \lor [(a \lor c) \land (\neg c \lor c)] && \text{Negation Law}\\
&\equiv [(\neg a \lor b) \land \mathbf{T}] \lor [(a \lor c) \land \mathbf{T}] && \text{Negation Law}\\
&\equiv (\neg a \lor b) \lor [(a \lor c) \land \mathbf{T}] && \text{Identity Law}\\
&\equiv (\neg a \lor b) \lor (a \lor c) && \text{Identity Law}\\
&\equiv (\neg a \lor a) \lor (b \lor c) && \text{Commutative Law}\\
&\equiv \mathbf{T} \lor (b \lor c) && \text{Negation Law}\\
&\equiv \mathbf{T} && \text{Domination Law}
\end{aligned}$$

Propositional Satisfiability

Given a compound proposition, we can demonstrate it is NOT a contradiction by simply giving one example assignment of truth values to the propositional variables such that the compound proposition is true.

Def: A compound proposition is *satisfiable* if there is an assignment of truth values to its variables that makes it true; such an assignment is a *solution.* Otherwise, the proposition is *unsatisfiable* (which means it is a contradiction!).

Example: Determine if the compound proposition $(p \lor q) \land (\neg p \lor \neg q)$ is satisfiable. [Hint: the first clause says that one of p and q is true, while the second says that one is false.]

Example: Determine if the compound proposition $(p \lor q) \land (\neg p \lor \neg q) \land (\neg p \lor q)$ is satisfiable.

Example: Determine if the compound proposition $(p \lor q) \land (\neg p \lor \neg q) \land (\neg p \lor q) \land (p \lor \neg q)$ is satisfiable.

$$\begin{aligned}
(p \vee q) \wedge (\neg p \vee \neg q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) &\equiv (p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \to q) \wedge (q \to p) && \text{Logical equivalence of conditional statement} \\
&\equiv (p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \leftrightarrow q) && \text{Logical equivalence using biconditional statement} \\
&\equiv (\neg p \to q) \wedge (q \to \neg p) \wedge (p \leftrightarrow q) && \text{Logical equivalence of conditional statement} \\
&\equiv (\neg p \leftrightarrow q) \wedge (p \leftrightarrow q) && \text{Logical equivalence using biconditional statement} \\
&\equiv \neg(p \leftrightarrow q) \wedge (p \leftrightarrow q) && \text{Logical equivalence using biconditional statement} \\
&\equiv \neg r \wedge r && \textit{Substitution } r = p \leftrightarrow q \\
&\equiv \mathbf{F} && \text{Negation law}
\end{aligned}$$

Applications

Book has Sudoku. Let's talk about KenKen. But first: Latin Squares!

Def: Let n be a positive integer. An $n \times n$ *latin square* is an $n \times n$ table where every row and column contains the numbers $1, \ldots, n$ exactly once.

We can create a proposition that is true if we have a latin square. We first need to create propositional variables.

Let $p(i, j, k)$ be true if the i, j position has value k.

For i, j, the compound proposition

$$\bigwedge_{k=1}^{n} [p(i,j,k) \to \neg p(i,j,k')]$$

guarantees that at most one number is placed in the i, j position. For integers i and k, the compound proposition

$$\bigvee_{j=1}^{n} p(i,j,k)$$

guarantees that the number k appears somewhere in the ith row. Similarly,

$$\bigvee_{i=1}^{n} p(i,j,k)$$

guarantees that the number k appears somewhere in the jth column. To take the conjunction of all of these statements, we find a compound proposition that is true if and only if the values $p(i, j, k)$ correspond to a latin square.

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=1}^{n} \bigwedge_{k' \neq k} [p(i,j,k) \to \neg p(i,j,k')] \wedge \bigwedge_{i=1}^{n} \bigwedge_{k=1}^{n} \bigvee_{j=1}^{n} p(i,j,k) \wedge \bigwedge_{j=1}^{n} \bigwedge_{k=1}^{n} \bigvee_{i=1}^{n} p(i,j,k).$$

Exercise: Determine why we do not need to add propositions for the following properties:
1. The i, j position contains at least one number.
2. The ith row contains the number k at most once.
3. The jth column contains the number k at most once.

KenKen is a puzzle that starts with a partially-complete latin square [that is, some of the propositional variables $p(i, j, k)$ are set to $\mathbf{T}$; e.g. $p(2, 1, 5)$ is part of our conjunction]. But also, other constraints are added! Specifically, some contiguous blocks of the square are grouped together and an algebraic operation ($+$, $-$, $\times$, or $/$) is specified along with the desired output when applying that operation to those numbers. These can be specified using our logic, although in a more complicated way!

Example: Suppose that in a 7×7 KenKen, the pair of positions $(3,4)$ and $(4,4)$ have the constraint "$\times 21$" Then, we explore the possible values that can multiply to find 21. These are exactly 3×7 and 7×3. So, we add to our list of constraints the disjunction:

$$[[p(3,4,3) \wedge p(3,4,7)] \vee [p(3,4,7) \wedge p(3,4,3)]]$$

Example: Suppose that in a 7×7 KenKen, the pair of positions $(3,4)$ and $(4,4)$ have the constraint "/2" Then, we explore the possible values that can divide to form 2. These are $2/1$, $4/2$ and $6/3$. So, we add to our list of constraints the disjunction:

$$[[p(3,4,1) \wedge p(3,4,2)] \vee [p(3,4,2) \wedge p(3,4,1)] \vee [p(3,4,2) \wedge p(3,4,4)] \vee [p(3,4,4) \wedge p(3,4,2)] \vee [p(3,4,3) \wedge p(3,4,6)] \vee [p(3,4,6) \wedge p(3,4,3)]]$$