CS 173, Lecture B
September 1, 2015
Tandy Warnow

1 Overview of today's lecture

- Simplifying logical expressions, and seeing when two logical expressions are equivalent
- Determining if a logical expression can be *satisfied*
- Expressing English statements in logic

2 Simplifying logical expressions.

Objectives:

- Remove all unnecessary parentheses
- Remove all $\rightarrow$ or $\leftrightarrow$

Hence, you need to be able to simplify expressions like

$$\neg(a \rightarrow b) \vee (\neg b)$$

When A and B are logical expressions, and you say $A \equiv B$, you mean that they have the same truth values. (You can also write this as $A \leftrightarrow B$.)

For example:

- $\neg\neg x \equiv x$ (obvious)
- $x \vee (x \wedge y) \equiv x$

  Similarly, you can write $x \vee (x \wedge y) \leftrightarrow x$. In other words, $x \vee (x \wedge y)$ is true if and only if x is true.
- $x \vee \neg x \equiv T$

  In other words, $x \vee \neg x$ is always true, no matter what x is.
- $x \wedge \neg x \equiv F$

  In other words, $x \wedge \neg x$ is never true, no matter what x is.

Distribution. Another trick that can be very helpful is to use *distribution*:

- $(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$
- $(P \vee (Q \wedge R)) \equiv ((P \vee Q) \wedge (P \vee R))$

Truth Tables. We (sometimes) use truth tables to check our analyses. Here's an example of a very simple truth table for the expression $A \wedge B$:

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

A more complicated truth table. Consider the expression

$$[(A \to B) \wedge \neg B] \to A$$

Is this always true? Sometimes true and sometimes false? Always false? Let's use a truth table to answer this.

A	B	$(A \to B) \wedge \neg B$	$[(A \to B) \wedge \neg B] \to A$
T	T	F	T
T	F	F	T
F	T	F	T
F	F	T	F

So the answer is that it is sometimes true and sometimes false. Note that we also showed $[(A \to B) \wedge \neg B] \to A \equiv A \vee B$.

De Morgan's Laws.

- Negation of $A \wedge B$: $\neg A \vee \neg B$

  This is also written as

  $$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

  or as

  $$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$

- Negation of $A \vee B$: $\neg A \wedge \neg B$

  This is also written as

  $$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

  or as

  $$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$$

Negation, warm up with quantifiers.

- Negation of $\forall x \in S, P(x)$:

  $\exists x \in S$ s.t. $\neg P(x)$

  Negation of $\exists x \in S$ s.t. $P(x)$

  - $\forall x \in S, \neg P(x)$

Negating Logical Expressions. Consider the expression

$$A \to B$$

To negate this, we have:

$$\begin{aligned} &\neg(A \to B) \\ &\equiv \neg(\neg A \vee B) \\ &\equiv \neg\neg A \wedge \neg B \\ &\equiv A \wedge \neg B \end{aligned}$$

Now let's negate: $(x \to y) \wedge \neg x$
First Solution:

$$\begin{aligned} &\neg[(x \to y) \wedge \neg x] \\ &\equiv \neg(x \to y) \vee \neg\neg x \\ &\equiv \neg(\neg x \vee y) \vee x \\ &\equiv (\neg\neg x \wedge \neg y) \vee x \\ &\equiv (x \wedge \neg y) \vee x \\ &\equiv x \end{aligned}$$

Second Solution: We begin by simplifying the expression above before negating it. Note that

$$x \to y \equiv \neg x \vee y$$

Hence

$$\begin{aligned} &(x \to y) \wedge \neg x \\ &\equiv (\neg x \vee y) \wedge \neg x \\ &\equiv (\neg x \wedge \neg x) \vee (y \wedge \neg x) \\ &\equiv \neg x \vee (y \wedge \neg x) \\ &\equiv \neg x \end{aligned}$$

Therefore,

$$\neg[(x \to y) \wedge \neg x] \equiv \neg\neg x \equiv x$$

3 Satisfiability

Some logical expressions can never be true, some are always true, and some depend on the values of their variables. **T** and **F** refer to the logical constants True and False, respectively. Examples:

1. $A \vee \neg A$ (always true)
2. $A \wedge \neg A$ (never true)
3. $A \vee B$ (sometimes true and sometimes false, depends on A and B)
4. $A \wedge F$ (never true)

Statements that are always true are called *tautologies*. Statements that can be true (or are always true) are said to be *satisfiable*, and otherwise they are said to be *unsatisfiable*.

Examples. For each of the following expressions, determine if it is satisfiable or not satisfiable. If it is satisfiable, determine if it is a tautology.

1. $(A \wedge B) \rightarrow A$
   (Answer: tautology)
2. $(A \wedge B) \rightarrow \neg A$
   (Answer: satisfiable (A = B = **F**) but not a tautology (A = B = **T**))
3. $(A \wedge B) \leftrightarrow A$
   (Answer: satisfiable (A = B = **T**) but not a tautology (A = **T** and B = **F**)
4. $(A \rightarrow B) \wedge A \wedge \neg B$
   (Answer: not satisfiable, so never true)
5. $A \rightarrow \neg A$
   (Answer: satisfiable (A = **F**) but not a tautology (A = **T**))

Conjunctive Normal Form (CNF). A logical expression of the form

$$A_1 \vee A_2 \vee A_3 \vee ... \vee A_k$$

where the A_i are literals (statement letters or their negations) is called a *disjunctive clause*.

Then

$$C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_p,$$

where each C_i is a disjunctive clause, is said to be in *conjunctive normal form*, or CNF.

CNF is very popular in computer science!

Two-satisfiability (2-SAT): A special case of CNF is where each clause has at most two literals! That is, expression that are written in the form $(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \ldots (A_k \vee B_k)$.

Which of the following CNF expressions are satisfiable?

1. $(x \vee y) \wedge (\neg x \vee \neg y)$
2. $(x \vee y) \wedge (\neg x \vee \neg y) \wedge x$
3. $(x \vee y) \wedge (\neg x \vee \neg y) \wedge x \wedge y$
4. $(x \vee y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee z) \wedge (\neg x \vee z)$
5. $(\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg z \vee x) \wedge (x \vee z)$

2-SAT is the problem where the input is one of these CNF expressions with at most two literals per clause, and the question is whether the expression is satisfiable. Since the answer is either YES or NO, this is called a **decision problem**.

It is not that easy to see, but this problem *can* be solved in polynomial time! On the other hand, 3-SAT, the corresponding problem where the clauses have up to three literals per clause, seems harder. In particular, no one has found a polynomial time algorithm for 3-SAT.