CS 2800, Logic and Computation — Fall 2009
Pete Manolios — Propositional Logic Lectures

1 Propositional Logic

The study of logic was initiated by the ancient Greeks, who were concerned with analyzing the laws of reasoning. They wanted to fully understand what *conclusions* could be derived from a given set of *premises*. Logic was considered to be a part of philosophy for thousands of years. In fact, until the late 1800's, no significant progress was made in the field since the time of the ancient Greeks. But then, the field of modern mathematical logic was born and a stream of powerful, important, and surprising results were obtained. For example, to answer foundational questions about the mathematics, logicians had to essentially create what later because the foundations of computer science. In this class, we'll explore some of the many connections between logic and computer science.

We'll start with propositional logic, a simple, but surprisingly powerful fragment of logic. Expressions in propositional logic can only have one of two values. We'll use T and F to denote the two values, but other choices are possible, *e.g.*, 1 and 0 are sometimes used.

The expressions of propositional logic include:

1. The *constant expressions true* and *false*: they always evaluate to T and F, respectively.

2. The *propositional atoms*, or more succinctly, *atoms*. We will use p, q, and r to denote propositional atoms. Atoms range over the values T and F.

Propositional expressions can be combined together with the propositional connectives, which we include the following.

The simplest connective is negation. Negation, $\neg$, is a *unary* connective, meaning that it is applied to a single expression. For example $\neg p$ is the negation of atom p. Since p (or any propositional expression) can only have one of two values, we can fully define the meaning of negation by specifying what it does to the value of p in these two cases. We do that with the aid of the following truth table.

p	$\neg p$
T	F
F	T

What the truth table tells us is that if we negate T we get F and if we negate F we get T.

Negation is the only unary propositional connective we are going to consider. Next we consider *binary* (2-argument) propositional connectives, starting with *conjunction*, $\wedge$. The conjunction (and) of p and q is denoted $p \wedge q$ and its meaning is given by the following truth table.

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

Each row in a truth table corresponds to an *assignment*, one possible way of assigning values (T or F) to the atoms of a formula. The truth table allows us to explore all relevant assignments. If we have two atoms, there are 4 possibilities, but in general, if we have n atoms, there are 2^n possible assignments we have to consider.

In one sense, that's all there is to propositional logic, because every other connective we are going to consider can be expressed in terms of $\neg$ and $\wedge$, and almost every question we are going to consider can be answered by the construction of a truth table.

Next, we consider *disjunction*. The disjunction (or) of p and q is denoted $p \vee q$ and its meaning is given by the following truth table.

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

In English usage, "p or q" often means p or q, but not both. Consider the mother who tells her child:

> You can have ice cream or a cookie.

The child is correct in assuming this means that she can have ice cream or a cookie, but not both.

As you can see from the truth table for disjunction, in logic ”or” always means at least one.

We can write more complex formulas by using several connectives. For example, $\neg p \vee \neg q$ and we can construct truth tables for such expressions quite easily. First, determine how many distinct atoms there are. In this case there are two; that means we have four rows in our truth table. Next we create a column for each atom and for each connective. Finally, we fill in the truth table, using the truth tables that specify the meaning of the connectives.

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

Next, we consider implication, $\Rightarrow$. This is called logical (or material) implication. In $p \Rightarrow q$, p is the antecedent and q is the consequent. Implication is often confusing to students because the way it is used in English is quite complicated and subtle. For example, consider the following sentences.

> If Obama is 75 years old, then the inhabitants of this city are all dragons.

Is it true?
What about the following?

> If Obama is president, then the inhabitants of this city are all dragons.

Logically, only the first is true, but most English speakers will say that if there is no connection between the antecedent and consequent, then the implication is false.

Why is the first logically true? Because here is the truth table for implication.

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Here are two ways of remembering this truth table. First, $p \Rightarrow q$ is equivalent to $\neg p \vee q$. Second, $p \Rightarrow q$ is false only when p is T, but q is F. This is because you should think of $p \Rightarrow q$ as claiming that if p holds, so does q. That claim is true when p is F. The claim can only be be invalidated if p holds, but q does not.

As a final example of the difference between logical implication (whose meaning is given by the above truth table) and implication as commonly used, consider the mother telling her child:

> If you behave, I'll get you ice cream.

The child rightly expects to get ice cream if she behaves, but also expects to *not* get ice cream if she doesn't: there is an implied threat here.

The point is that while the English language is subtle and open for interpretation. In order to avoid misunderstanding mathematical fields, like Computer Science tend to use what is often called "mathematical English," a very constrained version of English, where the meaning of all connectives is clear.

Above we said that $p \Rightarrow q$ is equivalent to $\neg p \vee q$. This is the first indication that we can often reduce propositional expressions to simpler forms. In your homework, simpler means less connectives, so which of the above is simpler?

Can we express the equivalence in propositional logic? Yes, using equality of Booleans, $\equiv$, as follows $(p \Rightarrow q) \equiv (\neg p \vee q)$.

Here is the truth table for $\equiv$.

p	q	$p \equiv q$
T	T	T
T	F	F
F	T	F
F	F	T

How would you simplify the following?

1. $p \wedge \neg p$
2. $p \vee \neg p$
3. $p \equiv p$

Here is one way.

1. $(p \wedge \neg p) \equiv \mathit{false}$
2. $(p \vee \neg p) \equiv \mathit{true}$
3. $(p \equiv p) \equiv \mathit{true}$

The final binary connective we will consider is $\oplus$, xor. There are two ways to think about xor. First, note that xor is e$x$clusive or, meaning that exactly one of its arguments is true. Second, note that xor is just the Boolean version of not equal. Here is the truth table for $\oplus$.

p	q	$p \equiv q$
T	T	F
T	F	T
F	T	T
F	F	F

We will also consider a *ternary* connective, *i.e.*, a connective with three arguments. The connective is *ite*, which stands for "if-then-else," and means just that: if the first argument holds, return the second (the then branch), else return the third (the else branch). Since there are three arguments, there are eight rows in the truth table.

p	q	r	$ite(p, q, r)$
T	T	T	T
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	F

Here are some very useful ways of characterizing propositional formulas. Start by constructing a truth table for the formula and look at the column of values obtained. We say that the formula is:

- *satisfiable* if there is at least one T

- *unsatisfiable* if it is not satisfiable, *i.e.*, all entries are F
- *falsifiable* if there is at least one F
- *valid* if it is not falsifiable, *i.e.*, all entries are T

We have see examples of all of the above. For example, $p \wedge q$ is satisfiable, since the assignment that makes p and q T makes $p \wedge q$ T. This example is also falsifiable, as evidenced by the assignment that makes p F and q T. An example of an unsatisfiable formula is $p \wedge \neg p$. If you construct the truth table for it, you will notice that every assignment makes it F (so it is falsifiable too). Finally, an example of a valid formula is $p \vee \neg p$.

Notice that if a formula is valid, then it is also satisfiable. In addition, if a formula is unsatisfiable, then it is also falsifiable.

Validity turns out to be really important. A valid formula, often also called a *theorem*, corresponds to a correct logical argument, an argument that is true regardless of the values of its atoms. For example $p \Rightarrow p$ is valid. No matter what p is, $p \Rightarrow p$ always holds.

2 The Power of Xor

Let us take a short detour, I'll call "the power of xor."

Suppose that you work for a secret government agency and you want to communicate with your counterparts in Europe. You want the ability to send messages to each other using the Internet, but you know that other spy agencies are going to be able to read the messages as they travel from here to Europe.

How do you solve the problem?

Well, one way is to have a shared secret: a long sequence of F's and T's (0's and 1's if you prefer), in say a code book that only you and your counterparts have. Now, all messages are really just sequences of bits, which we can think of as sequences of F's and T's, so you take your original message m and xor it, bit by bit, with your secret s. That gives rise to coded message c, where $c \equiv m \oplus s$. Notice that here we are applying $\equiv$ and $\oplus$ to sequences of Boolean values, often called *bit-vectors*.

Anyone can read c, but they will have no idea what the original message was, since s effectively scrambled it. In fact, with no knowledge of s, an eavesdropper can extract no information about the contents of m from c.

But, how will your counterparts in Europe decode the message? Notice that some propositional reasoning shows that $m = c \oplus s$, so armed with your shared secret, they can determine what the message is.

3 Complete Boolean Bases

Next, think about this claim: you can represent all the Boolean connectives using just *ite* (and the constants *false*, *true*).

Also, consider the claim that for any Boolean formula, there is an equivalent formula consisting of only the following connectives:

1. $\wedge, \neg$
2. $\vee, \neg$

Here is a partial answer.

$$\neg p \equiv \mathit{ite}(p, \mathit{false}, \mathit{true})$$

$$p \vee q \equiv \mathit{ite}(p, \mathit{true}, q)$$
$$p \wedge q \equiv \mathit{ite}(p, q, \mathit{false})$$
$$p \Rightarrow q \equiv \mathit{ite}(p, q, \mathit{true})$$
$$p \equiv q \equiv \mathit{ite}(p, q, \neg q)$$
$$p \oplus q \equiv \mathit{ite}(p, \neg q, q)$$

4 Propositional Logic in ACL2s

This class is about logic from a computational point of view and our vehicle for exploring computation is ACL2. ACL2s has *ite*: it is just `if`!

A difference is that ACL2s really has generalized booleans: instead of *false* and *true* we have: `nil` and non-`nil`.

For example the way that ACL2s evaluates `(if a b c)` is it checks if `a`=`nil`; if so, it returns `c`, else it returns `b`.

What does ACL2s do on the following?

- `(if t 2 3)`

- `(if 3 4 5)`

So `if` can return non-Booleans and even the test need not be Boolean. Remember that in the test of an `if`, every object in the ACL2s universe that is non-`nil` is coerced to `t`.

In ACL2s we have following correspondence.

- *ite*: `if`
- $\wedge$: `and`
- $\vee$: `or`
- $\Rightarrow$: `implies`
- $\equiv$: `iff` (or `equal`)
- $\oplus$: `xor`

We saw how to evaluate `if`. What about the rest of the connectives? They are all defined in terms of `if`, so try the following commands in ACL2s.

- `:trans (and a b)`. This tells us that `(and a b)` is really `(if a b nil)`.
- `:trans (or a b)`. This tells us that `(and a b)` is really `(if a a b)`.
- `:pe implies`. This tells us that the `implies` is defined to be `(if p (if q t nil) t))`.
- `:pe iff`. This tells us that `iff` is defined to be `(if p (if q t nil) (if q nil t)))`.
- `:pe xor`. This tells us that `xor` is defined to be `(if p (if q nil t) (if q t nil))))`.

One more thing to note is that `and` and `or` take an arbitrary number of arguments. Here are some examples.

- `(and)` is `t`
- `(or)` is `nil`

- `(and p)` is `p`
- `(or p)` is `p`
- `(and p q r)` is `(if p (if q r nil) nil)`
- `(or p q r)` is `(if p p (if q q r))`

Here is a design guideline for using Boolean connectives.

Design guideline: do not use **`and`**, `or`, `implies`, `iff` on non-Booleans.

Remember that ACL2s is in the business of proving theorems. Since propositional logic is used everywhere, it would be great if we could use ACL2s to reason about propositional logic. In fact, we can.

Consider trying to prove that a propositional formula is valid. We would do that now, by constructing a truth table. We can also just ask ACL2s. For example, to check whether the following is valid

$$(p \Rightarrow q) \equiv (\neg p \vee q)$$

We can ask ACL2s the following query

```
(thm (iff (implies p q) (or (not p) q)))
```

ACL2s responds with the output shown in ACL2s Output Figure **??**.

```
ACL2 >QUERY (thm (iff (implies p q) (or (not p) q)))

<< Starting proof tree logging >>

But we reduce the conjecture to T, by case analysis.

Q.E.D.

Summary
Form:  ( THM ...)
Rules: NIL
Warnings:  None
Time:  0.00 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, ...)

Proof succeeded.
```

ACL2s Output Figure 1: Successful proof

In fact, if a propositional formula is valid (that is, it is a theorem) then ACL2s will definitely prove it. We say that ACL2s is a decision procedure for propositional validity. A *decision procedure* for propositional validity is program that given a formula can decide whether or not it is valid. We saw that ACL2s indicates that it has determined that a formula is valid with "Q.E.D." [1] How does ACL2 indicate that a formula is not valid? Let's try checking whether the following is valid:

$$(p \oplus q) \equiv (p \vee q)$$

We can ask ACL2s the following query

```
(thm (iff (xor p q) (or p q)))
```

ACL2s responds with the output shown in ACL2s Output Figure **??**.

5 Decision Procedures

When a formula is not valid, it is falsifiable, so there exists an assignment that makes it false. Such an assignment is often called a *counterexample* and can be very useful for debugging purposes. ACL2s does not provide counterexamples. However, since ACL2s is a programming language, we can use it to write our own decision procedure that does provide counterexamples to validity. I showed you such a program in class. Try writing your own.

While we are on the topic of decision procedures, it is worth pointing that we characterized formulas as satisfiable, unsatisfiable, valid, or falsifiable. Let's say we have a decision procedure for one of these four characterizations. Then, we can, rather trivially, get a decision procedure for any of the other characterizations.

Why?

Well, consider the following.

Proof

[1] Q.E.D. is abbreviation for "quod erat demonstrandum," Latin for "that which was to be demonstrated."

```
ACL2 >QUERY (thm (iff (xor p q) (or p q)))

<< Starting proof tree logging >>

By case analysis we reduce the conjecture to

Goal'
(COND ((XOR P Q) (OR P Q))
      ((OR P Q) NIL)
      (T T)).

This simplifies, using the :definition XOR, to

Goal''
(IMPLIES P (NOT Q)).

This simplifies, using trivial observations, to

Goal'''
NIL.
^^^ Checkpoint Goal''' ^^^


Summary
Form:  ( THM ...)
Rules: ((:DEFINITION IFF) (:DEFINITION XOR))
Warnings:  None
Time:  0.00 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, ...)
******** FAILED ********
```

ACL2s Output Figure 2: Failed proof

Unsat f

$\equiv$ { By the definition of sat, unsat }

not (Sat f)

$\equiv$ { By definition of Sat, Valid }

Valid $\neg f$

$\equiv$ { By definition of valid, falsifiable }

not (Falsifiable $\neg f$) $\Box$

How do we use these equalities to obtain a decision procedure for either of unsat, sat, valid, falsifiable, given a decision procedure for the other?

Well, let's consider an example. Say we want a decision procedure for validity given a decision procedure for satisfiability.

Valid f

$\equiv$ { not (Sat f) $\equiv$ *Valid*$\neg f$, by above }

not (Sat $\neg f$)

What justifies this step? Propositional reasoning and instantiation.

Let p denote "(Sat f)" and q denote "(Valid $\neg f$)." The above equations tell us $\neg p \equiv q$, so $p \equiv \neg q$.

If more explanation is required, note that $(\neg p \equiv q) \equiv (p \equiv \neg q)$ is valid. That is, you can transfer the negation of one argument of an equality to the other.

Make sure you can do this for all 12 combinations of starting with a decision procedure for sat, unsat, valid, falsifiable, and finding decision procedures for the other three characterizations.

There are two interesting things to notice here.

First, we took advantage of the following equality:

$$(\neg p \equiv q) \equiv (p \equiv \neg q)$$

There are lots of equalities like this that you should know about, so I'll give you a list.

Second, we saw that it was useful to extract the propositional skeleton from an argument. We'll look at examples of doing that. Initially this will involve word problems, but later it will involve reasoning about programs.

6 Useful Equalities

Here are some simple equalities involving the constant *true*.

1. $p \vee true \equiv true$

2. $p \wedge true \equiv p$

3. $p \Rightarrow true \equiv true$

4. $true \Rightarrow p \equiv p$

5. $p \equiv true \equiv p$

6. $p \oplus \mathit{true} \equiv \neg p$

Here are some simple equalities involving the constant *false*.

1. $p \vee \mathit{false} \equiv p$
2. $p \wedge \mathit{false} \equiv \mathit{false}$
3. $p \Rightarrow \mathit{false} \equiv \neg p$
4. $\mathit{false} \Rightarrow p \equiv \mathit{true}$
5. $p \equiv \mathit{false} \equiv \neg p$
6. $p \oplus \mathit{false} \equiv p$

Why do we have separate entries for $p \Rightarrow \mathit{false}$ and $\mathit{false} \Rightarrow p$, above, but not for both $p \vee \mathit{false}$ and $\mathit{false} \vee p$? Because $\vee$ is commutative. Here are some equalities involving commutativity.

1. $p \vee q \equiv q \vee p$
2. $p \wedge q \equiv q \wedge p$
3. $p \equiv q \equiv q \equiv p$
4. $p \oplus q \equiv q \oplus p$

What about $\Rightarrow$. Is it commutative? Is $p \Rightarrow q \equiv q \Rightarrow p$ valid? No. By the way, the righthand side of the previous equality is called the *converse*: it is obtained by swapping the antecedent and consequent.

A related notion is the *inverse*. The inverse of $p \Rightarrow q$ is $\neg p \Rightarrow \neg q$. Note that the inverse and converse of an implication are equivalent.

Even though a conditional is not equivalent to its inverse, it is equivalent to its *contrapositive*:

$$(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$$

The contrapositive is obtained by negating the antecedent and consequent and then swapping them.

While we're discussing implication, a very useful equality involving implication is:

$$(p \Rightarrow q) \equiv (\neg p \vee q)$$

Also, we often want to replace $\equiv$ by $\Rightarrow$, which is possible due to the following equality:

$$(p \equiv q) \equiv [(p \Rightarrow q) \wedge (q \Rightarrow p)]$$

Here are more equalities.

1. $\neg\neg p \equiv p$
2. $\neg \mathit{true} \equiv \mathit{false}$
3. $\neg \mathit{false} \equiv \mathit{true}$
4. $p \wedge p \equiv p$
5. $p \vee p \equiv p$
6. $p \Rightarrow p \equiv \mathit{true}$
7. $p \equiv p \equiv \mathit{true}$
8. $p \oplus p \equiv \mathit{false}$
9. $p \wedge \neg p \equiv \mathit{false}$
10. $p \vee \neg p \equiv \mathit{true}$
11. $p \Rightarrow \neg p \equiv \neg p$
12. $\neg p \Rightarrow p \equiv p$
13. $p \equiv \neg p \equiv \mathit{false}$
14. $p \oplus \neg p \equiv \mathit{true}$

Here's one set of equalities you have probably already seen: DeMorgan's Laws.

1. $\neg(p \wedge q) \equiv \neg p \vee \neg q$
2. $\neg(p \vee q) \equiv \neg p \wedge \neg q$

Here's another property: associativity.

1. $((p \vee q) \vee r) \equiv (p \vee (q \vee r))$
2. $((p \wedge q) \wedge r) \equiv (p \wedge (q \wedge r))$
3. $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$
4. $((p \oplus q) \oplus r) \equiv (p \oplus (q \oplus r))$

We also have distributivity:

1. $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
2. $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

We also have transitivity:

1. $[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$
2. $[(p \equiv q) \wedge (q \equiv r)] \Rightarrow (p \equiv r)$

Last, but not least, we have absorption:

1. $p \wedge (p \vee q) \equiv p$
2. $p \vee (p \wedge q) \equiv p$

Let's consider absorption more carefully. Here is a simple calculation:
Proof

$p \wedge (p \vee q)$

$\equiv$ { Distribute $\wedge$ over $\vee$ }

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

$\equiv$ { $(p \wedge p) \equiv p$ }

$p \vee (p \wedge q)$ $\Box$

The above proof shows that $p \wedge (p \vee q) \equiv p \vee (p \wedge q)$, so if we show that $p \wedge (p \vee q) \equiv p$, we will have also shown that $p \vee (p \wedge q) \equiv p$.

Let's try to show that $p \wedge (p \vee q) \equiv p$. We will do this using a proof technique called case analysis.

Case analysis: If f is a formula and p is an atom, then f is valid iff both $f|_{\langle p,true\rangle}$ and $f|_{\langle p,false\rangle}$ are valid. By $f|_{\langle p,x\rangle}$ we mean, substitute x for p in f.

Proof

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

$\equiv$ { Case analysis }

$\mathit{true} \wedge (\mathit{true} \vee q) \equiv \mathit{true}$ and $\mathit{false} \wedge (\mathit{false} \vee q) \equiv \mathit{false}$

$\equiv$ { Basic Boolean equalities }

$\mathit{true} \equiv \mathit{true}$ and $\mathit{false} \equiv \mathit{false}$

$\equiv$ { Basic Boolean equalities }

true $\square$

7 Word Problems

Next, we consider how to formalize word problems using propositional logic.

Consider formalizing and analyzing the following.

> Tom likes Jane if and only if Jane likes Tom. Jane likes Bill. Therefore, Tom does not like Jane.

Here's the kind of answer I expect you to give.

> Let p denote "Tom likes Jane"; let q denote "Jane likes Tom"; let r denote "Jane likes Bill."
>
> The first sentence can then be formalized as $p \equiv q$.
>
> We denote the second sentence by r.
>
> The third sentence contains the claim we are to analyze, which can be formalized as $((p \equiv q) \wedge r) \Rightarrow \neg p$.
>
> This is not a valid claim. A truth table shows that the claim is violated by the assignment that makes p, q, and r *true*. This makes sense because r (that Jane likes Bill) does not rule out q (that "Jane likes Tom"), but q requires p (that "Tom likes Jane").

Consider another example.

> A grade will be given if and only if the test is taken. The test has been taken. Was a grade given?

Anything of the form "*a* iff *b*" is formalized as $a \equiv b$. The problem now become easy to analyze.

> John is going to the party if Mary goes. Mary is not going. Therefore, John isn't going either.

How do we formalize "*a* if *b*"? Simple: $b \Rightarrow a$. Finish the analysis.

> John is going to the party only if Mary goes. Mary is not going. Therefore, John isn't going either.

How do we formalize "only if"? A simple way to remember is that "if" is one direction of and "if and only if" and "only if" is the other direction. Thus, "*a* only if *b*" is formalized as $a \Rightarrow b$.

Try this one.

> John is going to the party only if Mary goes. Mary is going. Therefore, John is going too.

One more.

> Paul is not going to sleep unless he finishes the carrot hunt on Final Fantasy XII. Paul went to sleep. Therefore, he finished the carrot hunt on Final Fantasy XII.

How do we formalize "*a* unless *b*"? It is $\neg b \Rightarrow a$. Why? Because "*a* unless *b*" says that *a* has to be *true*, except when (unless) *b* is *true*, so when *b* is *true*, *a* can be anything. The only assignment that violates "*a* unless *b*" is when *a* is *false* and *b* is *false*. So, notice that "*a* unless *b*" is equivalent to "*a* or *b*".

One more example of unless.

> You will not get into NEU unless you apply.

is the same as

> You will not get into NEU if you do not apply.

which is the same as

> You will not get into NEU or you will apply.

So, the hard part here is formalizing the problem. After that, even ACL2s can figure out if the argument is valid.