DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

CSC 202 NOTES
Spring 2010: Amber Settle

Week 1: Monday, March 29, 2010

Announcements

- Discussion of the syllabus
  - Textbook
  - Tour of COL site
  - Questions
    - E-mail
    - Skype
    - In person
    - Ask early and ask often!
  - Online section
- Class roster
- First assignment is due Monday, April 5th

Course overview

In this course, we will learn the basic concepts of discrete mathematics that are central to computer science. This includes (but is not limited to):

1. Propositional logic
2. Sets and set operations
3. Relations and functions
4. Quantified statements
5. Graph theory
6. Basic probability
7. Basic combinatorics

Whenever possible we will learn this in the context of computer science problems so that you can see the relevance of what will be somewhat abstract concepts.

Above all else I hope that you will gain knowledge of how to approach and solve problems that arise in computer science and an appreciation of the beauty of computing.

Solving problems

A colleague once shared a quote from *Grundriß der Logik* written by Maass several centuries ago. The English translation[1] is worth considering as we start this class:

[1] My thanks to Marcus Schäfer for the original quote and the translation

When given a question, one should first of all 1) determine its meaning as clearly and accurately as possible; then one may 2)replace terms with other terms of the same meaning if this simplifies finding the solution: 3) often, the question can be split into several questions, which can, similarly, simplify finding the answer; 4) it is often helpful to consider a special, more particular, version of the question; 5) sometimes the question can be reduced to another question which is easier to answer; 6) occasionally one obtains a solution by turning the question into a statement, and assuming its negation. On studying the negation one will often discover it is false, which obtains the answer to the original question.

> This quote gives us a lot of information that is useful for your work in this class. We will consider the quote point by point.

Determine the meaning of a question

While this seems so obvious so as to be meaningless, it is actually quite profound and important.

> This information is the single most important piece of advice I would share with new graduate students, since it saved me repeatedly from handing in assignments that were nonsense.

Questions can be unclear for a number of reasons:

1. The person asking the question doesn't know exactly what they mean or want
2. The question is phrased in a natural language, and natural languages are inherently ambiguous
3. Questions phrased in mathematical formalism must be translated from a natural language, a process that can be done incorrectly or misunderstood

To follow this advice we should:

1. Read the question carefully

2. Understand all parts of the question
3. Make sure that the meaning of all of the parts of the question are clear

Replace terms with equivalent terms

This advice is really telling you to make sure that you're clear on the definition of terms introduced in the question.

For example, if a question asks you to show that a number is prime, you need to understand that a prime number is one that is divisible only by 1 and itself.

This step is important because **the definition of a problem may also show you how to approach solving the problem**.

For example, if you determine that all the numbers between 2 and 10 do not divide 11, then you know that 11 is prime.

This, of course, isn't the most efficient way to test primality. Computer scientists, unlike some mathematicians, are very concerned with efficiency of solutions.

Challenge question: Why do you only need to check the values from 2 to $\sqrt{x}$ in order to show that x is prime? Why is this more efficient?

Divide and recombine

One of the core techniques in problem solving is to:

1. Break a problem into smaller pieces
2. Solve those pieces separately
3. Combine the solutions to the smaller pieces into a solution for the original problem

Smaller problems are often easier to understand and may be easier to solve. If the combination of the solutions is also straightforward, then the problem has been made easier by dissecting it.

Special cases

There are two ways to read this advice:

1. **If the problem has some free variables, fix some of them**.

   For example, suppose that someone asks you to determine if it's true that if a | b and a | c then a | (b+c). (Here a | b means that a divides b evenly).

   You may not know where to start until you try some values.

Suppose a = 2, b = 4 and c = 6. Then 2 | 4 is true and 2 | 6 is true. But 2 | (4+6) is the same as 2 | 10 which is true.

Trying some more values may convince you that the statement is true and give you some insight into how to prove it.

2. **If the problem has a fixed variable, change it** (usually to a smaller value).

   For example, if you are given a problem on an 8 x 8 chessboard, you may consider a 2 x 2 or 4 x 4 chessboard. While you may not get the solution from the smaller boards, you can gain insight into the smaller problem which may help with the solution to the original problem.

The goal in both cases is to consider smaller problems in order to gain insight.

Reduction

Reducing one problem to another means **rephrasing the problem so that it looks like a problem you can solve**.

For example, you might know how to get to a location from your house. If you don't have directions to the location you want to reach, you can go home and then go to the new location.

(I have seen my 5-year-old daughter apply this principle in public places). Note that this may not be the most efficient solution, but it will work.

Negation

This approach is particularly helpful for logic problems and particularly when dealing with a statement of the form "for all x, P(x)".

Showing that a statement P(x) is true for an arbitrary x can be very difficult.

But if we assume that there is an x that makes P(x) false, then we have a specific x to work with and we can show that such an x doesn't exist.

Propositional logic

A proposition is a declarative sentence that is either true or false, but not both.

Examples:

- 2 + 2 = 4
- It rained in Chicago yesterday.

- There is sentient life on planets other than Earth in the universe.

Each of these is either true or false.

Not propositions:

- Read this carefully.
- $x + 1 = 2$

The truth value of this statement depends on the value of x

Generic propositions will usually be represented by a lower case p, q, r, s, etc.

Compound propositions

Simple propositions can be combined into more complicated ("compound") propositions using various connectives.

Given two propositions p and q, the four usual connectives are:

- $p \wedge q$: the **conjunction** of p and q
- $p \vee q$: the **disjunction** of p and q
- $\bar{p}$: the **negation** of p
- $p \rightarrow q$: the **conditional proposition**, with hypothesis p and conclusion q

Since p and q have truth values, compound propositions involving p and q must also have truth values. The value of the compound proposition depends on the values of the simple propositions it contains.

The truth value of a compound proposition is given by its **truth table**, which lists the truth value of the compound proposition for all possible combinations of truth values of its constituent simple propositions.

The following is the truth table for each of the connectives we have introduced:

p	q	$p \wedge q$	$p \vee q$	$\bar{p}$	$p \rightarrow q$
T	T	T	T	F	T
T	F	F	T	F	F
F	T	F	T	T	T
F	F	F	F	T	T

Comments:

- The compound proposition $p \wedge q$ is true exactly when both of p and q are true, which is why it is often stated as "p and q".
- Similarly, $p \vee q$ is true when either of p or q is true (or both), and for this reason it is often called "p or q".
- Negation is often termed "not p" because it has the opposite truth value from p.
- The compound proposition $p \rightarrow q$ is sometimes called "p implies q". This is because in most instances it does what we understand by implication.
  - If both p and q are true, then the truth of p has implied the truth of q, so the proposition is true.
  - If p is true, but q is false, then the truth of p has not implied the truth of q, so the proposition is false.
  - If p is false, then the truth value of the proposition should not depend on the truth value of q. We simply define $p \rightarrow q$ to be true when p is false.

Evaluating expressions

We can use the truth tables for the compound statements to evaluate the truth value of more complex compound propositions.

Examples: Suppose for p = T, q = F, and r = T

1. $(p \wedge q) \vee \bar{r}$
   $(T \wedge F)$ is F
   F $\vee$ F is F

2. $(p \vee q) \vee \bar{r}$ is T (Why?)
3. $p \wedge (q \rightarrow r)$ is T (Why?)

Unfortunately, without some concrete examples these concepts can be abstract and difficult to digest.

We will now discuss the context that will motivate many of our early examples in this course.

An introduction to databases

You may or may not be familiar with databases, and this course is not designed to teach you about databases.

However, they provide an excellent example for many of the concepts we will see in this course, and we will use databases as our motivating examples for logic, sets, relations, and functions.

For this reason, we will take some time to understand some basic database concepts.

A **database** is a collection of tables, where a **table** consists of multiple **rows** also known as **records**.

Each row records a point of data.
The columns of the table correspond to the different **fields** or **attributes** of a **record**, that is, the properties or characteristics of the record that are being stored.

A **relational database** allows us to store and organize many tables that contain related information.

The H2 database

For simplicity, I will use the H2 database engine in this course.

You can use Microsoft Access or MySQL but neither will handle all of the examples we will see in class. H2 is very easy to use and install and is compatible across multiple platforms.

There is a tutorial posted in the Useful Links that discusses how to install and use the database.
See: http://facweb.cs.depaul.edu/asettle/csc202/info/usingH2.htm

Let's step through the tutorial now.

Note that the sample database we will use is found in the university.sql file posted in the Documents on the COL site.

The file contains historical information about students and courses taken at DePaul. Note that prior to 2008, CDM was known as CTI and since the database is historical it uses the old name for the institution.

SQL

Data in a database is typically accessed using SQL (Structured Query Language).

A simple **SQL select query** consists of three clauses:

1. The SELECT clause specifying what information we want to retrieve
2. The FROM clause, specifying which table or tables we want to retrieve the information from, and
3. The WHERE clause, in which we can restrict what information we want to see.

Examples:

- SELECT *
  FROM Student;

  Returns all the records in the Student table

- SELECT LastName
  FROM Student;

  Returns the last names of all students in the Student table

  Note: SQL does not remove duplicates in its output, so if two students have the same last name, then that name will appear twice in the output

  To remove duplicates you can write:
  SELECT DISTINCT LastName
  FROM Student;

- SELECT LastName, FirstName, SSN, Career
  FROM Student;

  Returns multiple attributes from the Student table

- SELECT SID, LastName, FirstName, SSN
  FROM Student
  WHERE Career = 'UGRD';

  Returns all undergraduate student information since the WHERE clause limits the output of the query to records that have UGRD in the Career field

Exercises: What do the following queries return? Describe the elements returned in English.

Query 1:

```
SELECT SSN
FROM Student
WHERE Career = 'GRD';
```

Query 2:

```
SELECT FirstName
FROM Student
WHERE LastName = 'Patel';
```

Query 3:

```
SELECT FirstName, LastName, SID
FROM Student
WHERE Started < 2004;
```

Keys

A **key** is a set of attributes in a table that uniquely identifies a single row, but does not contain any null values and does not contain any superfluous attributes.

A key is crucial for accessing and manipulating a database, for example, for finding a list of all students found in our sample database.

Some combinations of attributes are unacceptable as keys:

- LastName and FirstName
- SSN

Why are each of the above unacceptable as keys?

If a good key does not exist, then the best solution is to introduce a key.

The typical solution at a university is to create a unique student ID (SID) for each student, which is both guaranteed to exist and to be unique.

More complex queries

The utility of a database is in its ability to provide us with answers to general questions about that data contained within it.

For example: Which courses have very high or very low enrollments? This information is needed to make scheduling decisions for future quarters.

It is in answering more complex queries that we begin to see the connection between logic and databases.

Example: Write a query that returns the name and ID number of all students who are not undergraduates.

Answer:

```
SELECT LastName, FirstName, SID
FROM Student
WHERE NOT Career = 'UGRD';
```

How does this work?

1. Opens the table specified in the FROM clause

2. Looks at each row of the table and checks the condition in the WHERE clause
3. If the condition is true, then the row is added to the output of the query

Example: Write a query that returns all students (e.g. the name and ID number) who are undergraduate computer science students.

Answer:
SELECT LastName, FirstName, SID
FROM Student
WHERE Career = 'UGRD' AND Program = 'COMP-SCI';

Example: Write a query that returns all Chicago graduate students who have a social security number.

Answer:
SELECT LastName, FirstName, SID
FROM Student
WHERE Career = 'GRD' AND City = 'Chicago' AND NOT SSN is null;

Note that we can write the conditions in any order – we could put the check for null first.

Example: Write a query that returns all students in the computer science and information systems degrees.

Answer:
SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'INFO-SYS' OR Program = 'COMP-SCI';

Why is this not the following?

SELECT LastName, FirstName, SID
FROM Student
WHERE Program = 'INFO-SYS' AND Program = 'COMP-SCI';

Back to propositional logic

Why are we talking about databases in a discrete math class?

Because when each query is executed, we are adding a row to the results returned if a proposition (possibly a compound proposition) is true and excluding the row if a proposition is false.

So the ability to write and understand propositions (most often compound ones) is crucial to being able to write a database query.

Logical equivalence

It is possible for two compound propositions that look very different to have the same truth table.

Two such compound propositions will have the exact same meaning, and are said to be **logically equivalent**.

Example: Consider the two propositions $\overline{p \wedge q}$ and $(\bar{p} \vee \bar{q})$.

p	q	$p \wedge q$	$\overline{p \wedge q}$	$\bar{p}$	$\bar{q}$	$\bar{p} \vee \bar{q}$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

For every possible combination of truth values for p and q, the two propositions have the same truth values.

Therefore the two propositions $\overline{p \wedge q}$ and $(\bar{p} \vee \bar{q})$ are logically equivalent, denoted by $\equiv$

The fact that these two propositions are logically equivalent is known as one of **DeMorgan's Laws**. We will return to DeMorgan's Law a bit later.

Precedence

As with arithmetic operators there is a specific order with which you are supposed to apply the logical operators.

The logical operators are applied in the following order (when parentheses are not present):

1. Negation
2. And ($\wedge$)
3. Or ($\vee$)

This means that $\bar{p} \vee q \wedge \bar{r}$ is equivalent to $(\bar{p}) \vee (q \wedge (\bar{r}))$

Note: In general it pays to add parentheses to avoid confusion!

It should be noted that **parentheses can make a difference in the value** of a logical expression (or we wouldn't worry about precedence).

Example: $(p \vee q) \wedge \bar{r}$ is not the same as $p \vee (q \wedge \bar{r})$

How can we verify this?
By constructing a truth table for the two expressions!

p	q	r	$\bar{r}$	$p \vee q$	$(p \vee q) \wedge \bar{r}$	$(q \wedge \bar{r})$	$p \vee (q \wedge \bar{r})$
T	T	T	F	T	F	F	T
T	F	T	F	T	F	F	T
F	T	T	F	T	F	F	F
F	F	T	F	F	F	F	F
T	T	F	T	T	T	T	T
T	F	F	T	T	T	F	T
F	T	F	T	T	T	T	T
F	F	F	T	F	F	F	F

Tautologies and falsehoods

Consider the formula $p \vee \bar{p}$

There is no way to make this compound proposition false. If p is F then $\bar{p}$ is T and the proposition is true. On the other hand, if p is T then the proposition is true.

Formulas that are always true regardless of the values of the propositions contained within them are called **tautologies**.

Along the same lines, formulas that are always false regardless of the values of their propositions are called **falsehoods**.

A compound proposition p is a falsehood if $\bar{p}$ is a tautology.

Note: Falsehoods and tautologies are useful for proving things about other, more complex formulas.

However, they are not very useful in SQL since a tautology in the WHERE clause will return all the records in the table and a falsehood in the WHERE clause will return no records from the table.

DeMorgan's Law redux

Recall what we showed earlier about $\overline{p \wedge q}$ and $(\bar{p} \vee \bar{q})$.

We showed that the two propositions are equivalent by constructing truth tables for them.

p	q	$p \wedge q$	$\overline{p \wedge q}$	$\bar{p}$	$\bar{q}$	$\bar{p} \vee \bar{q}$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

This means that two propositions $\overline{p \wedge q}$ and $(\bar{p} \vee \bar{q})$ are logically equivalent, denoted by $\equiv$

The fact that these two propositions are logically equivalent is known as one of **DeMorgan's Laws**.

The other of DeMorgan's Laws states that $\overline{p \vee q} \equiv (\bar{p} \wedge \bar{q})$.

Exercise: Show this by constructing a truth table.

How is DeMorgan's Law useful?

When considering database queries, it can allow us to simplify queries to make them more understandable (and possibly easier to verify for correctness).

Example: Suppose we want to write a query that lists students who are neither in computer science nor in information systems.

```
SELECT LastName, FirstName, SID
FROM Student
WHERE NOT Program = 'COMP-SCI' AND NOT Program =
'INFO-SYS';
```

This can be written as:

```
SELECT LastName, FirstName, SID
FROM Student
WHERE NOT (Program = 'COMP-SCI' OR Program = 'INFO-
SYS');
```

The first query has to evaluate the truth of two propositions and then perform 3 operations (two NOT and one AND).

The second query has to evaluate the two propositions but then only performs two operations (one OR and one NOT).

In a large database, the difference between performing 3 operations and performing 2 operations can have a significant impact on the efficiency of the query.

And as computer scientists we care very much about efficiency!

It might be the case that the first query is easier to read. This is why query optimizers are created that can look for equivalent, but more efficient, queries.

More logical equivalences

Consider the following compound proposition: $(p \wedge q) \vee (\overline{p} \wedge \overline{q})$

What is its truth table?

p	q	$\overline{p}$	$\overline{q}$	$(p \wedge q)$	$(\overline{p} \wedge \overline{q})$	$(p \wedge q) \vee (\overline{p} \wedge \overline{q})$
T	T	F	F	T	F	T
T	F	F	T	F	F	F
F	T	T	F	F	F	F
F	F	T	T	F	T	T

This is a very useful proposition, so useful that it has a specific name. It is called the **biconditional**, $p \leftrightarrow q$. Its (simplified) truth table is given below:

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

The biconditional is true precisely when p and q have the same truth value (either T or F). For this reason, it is often called "if and only if".

We can construct the biconditional from other connectives. $p \leftrightarrow q$ is equivalent to $(p \rightarrow q) \wedge (q \rightarrow p)$.

Exercise: Show this using a truth table.

Truth functions

A **truth function** $f(p_1, p_2, \ldots, p_n)$ is a proposition whose truth depends on the truth of a set of propositions $p_1, p_2, \ldots, p_n$.

Example: $p \wedge \bar{q}$ is a truth function of p and q.

In general, we can define a truth function through a truth table.

Example: f(p, q, r) is defined by the truth table below

p	q	r	f(p, q, r)
T	T	T	F
T	T	F	F
T	F	T	F
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	T
F	F	F	F

f(p, q, r) expresses that exactly one of p, q, and r is true.

Given the truth table for a truth function, can we **rewrite the function using the propositions** upon which it depends?

Yes, and there is a general approach for doing so:

1. Find all the cases where the formula is true (since we don't care about the cases where it's false).
2. Construct a case by taking the conjunction (and) of the variables, with the proposition represented as is when it is true and the negation of the proposition when it is false.
3. Take the disjunction (or) of all the cases where the function has to be true.

Example: There are three cases where f(p, q, r) needs to be true

- $p \wedge \bar{q} \wedge \bar{r}$
- $\bar{p} \wedge q \wedge \bar{r}$
- $\bar{p} \wedge \bar{q} \wedge \bar{r}$

So the disjunction of these clauses is the formula for f(p, q, r).

Example: Recall the truth table for $f(p, q) = p \leftrightarrow q$

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

There are two cases that make $p \leftrightarrow q$ true:

1. $\bar{p} \wedge \bar{q}$
2. $p \wedge q$

So the disjunction of these two clauses is the equivalent for $p \leftrightarrow q$, which is the formula we saw previously.

Exercise: Write a formula that is true if and only if an even number of the variables p, q, and r is true.

Normal forms

The formulas we derived in the previous section all have the same format.

To describe that format we need some definitions:

- A **literal** is a proposition or the negation of a proposition
- A **clause** is a conjunction or disjunction of literals

The formulas we derived are a disjunction of clauses, which are themselves conjunctions of literals. Formulas with this format are said to be in **disjunctive normal form** (DNF).

That means that our algorithm in the previous section proves the following **theorem**: Every truth function can be written as a formula in disjunctive-normal form.

There is a similar format, called **conjunctive normal form** (CNF), in which the formula is a conjunction of disjunctions of literals.

This means that the following **corollary** is true: Every truth function can be written in conjunction normal form.

Why? Take any truth function or formula α. Then by the previous theorem, $\bar{\alpha}$ can be written as a formula β in DNF. But then $\bar{\beta}$ is in CNF using DeMorgan's Law.

In-class exercises

As time permits, let's do some additional exercises to make sure that we are comfortable with the material presented so far.

Exercises:

1. Write an SQL query that will list the complete name and SID of each student in the university database who is in computer science

2. Write an SQL query to list the last name, first name, and SID of all undergraduate students who started before 2003.
3. Write an SQL query to list the last name, first name, and SID of all students who are not from Chicago and are not in computer science
4. Write an SQL query to list the last name, first name, and SID of all graduate students without social security numbers
5. Create a table with all possible truth assignments for the formulas p and $p \vee (q \wedge \overline{p})$. Do they differ? If so, give a truth assignment to p and q that yields different truth values for the two propositions.
6. Write a formula f(p, q, r) that is true if and only if exactly one of the variables p, q, or r is false.