CS70 Course Notes

Anmol Parande

Fall 2019 - Professors Alistair Sinclair and Yun Song

Disclaimer: These notes reflect CS60 when I took the course (Fall 2019). They may not accurately reflect current course content, so use at your own risk. If you find any typos, errors, etc, please report them on the GitHub repository.

Contents

1 Logic and Proofs

1.1 Propositional Logic

Definition 1 *A proposition P is a statement that is either true or false*

Propositions can depend on one or more variables. This is denoted by $P(x, y, ...)$

Definition 2 *A connective is an operator which joins two or more propositions together in some way*

Connectives are fully defined by a **Truth Table** which enumerates the values of the connective given all possible combination of inputs.

Definition 3 *The base connectives are those which can be combined to create any other connective.*

- $\wedge$*: P AND Q*
- $\vee$*: P OR Q*
- $\neg$*: NOT P*

One important connective is the **implies** connective.

P	Q	$P \implies Q$
T	T	T
T	F	F
F	T	T
F	F	T

Notice that $P \implies Q$ has the same truth table as $\neg P \vee Q$. This means they are equivalent.

Definition 4 *The contrapositive of $P \implies Q$ is $\neg Q \implies \neg P$*

Notice: The contrapositive is logically equivalent to the original statement

Definition 5 *The converse of $P \implies Q$ is $Q \implies P$*

Notice: This is not always equivalent to the original statement The **if and only if** connective $P \iff Q$ is equivalent to $(P \implies Q) \wedge (Q \implies P)$

P	Q	$P \iff Q$
T	T	T
T	F	F
F	T	F
F	F	T

Quantifiers help introduce variables into our propositions.

- $\forall$: For all

- $\exists$: There exists

Important: The order of quantifiers matters.

- $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
- $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
- $\neg(\forall x P(x)) \equiv \exists x(\neg P(x))$
- $\neg(\exists x P(x)) \equiv \forall x(\neg P(x))$

1.2 Proofs

Definition 6 *A proof is a sequency of statements, each of which follows from the preceding ones by a valid law of reasoning.*

Definition 7 *An Axiom is a basic fact which can be assumed without proof*

1.2.1 Direct Proof

Goal: Prove $P \implies Q$
Approach:

1. Assume P
2. Deduce Q through logical steps

1.2.2 Proof by Contraposition

Since the contrapositive of a statement is logically equivalent to the original statement, we can prove the original statement by proving the contrapositive.
Goal: To prove $P \implies Q$
Approach:

1. Assume $\neg Q$
2. Deduce $\neg P$ through logical steps

Theorem 1 *The pigeonhole principle says that if n objects are placed in k boxes and $n > k$, then some box contains ≥ 1 object.*

1.2.3 Proof by Contradiction

Goal: Prove $P \implies Q$ **Approach:**

1. Assume $\neg P$
2. Deduce $\neg P \implies (R \neg R)$

R and $\neg R$ are two facts which can be deduced from assumign $\neg P$

1.2.4 Proof by Cases

Goal: Prove $P \implies Q$
Approach:

1. Break P into cases $1...n$
2. Prove P holds in all cases

1.2.5 Proof by Induction

Goal: Prove $\forall n, P(n)$ **Approach:**

1. Prove a based case
2. Assume P(k) (Induction hypothesis)
3. Prove $P(k) \implies P(k+1)$ (Inductive Step)

1.2.6 Strong Induction

Because induction requires that $P(k)$ be true to prove $P(k+1)$, it might be easier to just assume $P(0) \wedge P(1) \wedge ... \wedge P(k)$ when proving $P(k+1)$.

1.3 Case Study: Stable Marriage Algorithm

1.3.1 The algorithm

The stable marriage algorithm uses a list of preferences to make pairings from two disjoint groups of people.

```
Input:
    - n men, n women
    - For each man, a ranked list of the women
    - For each woman, a ranked list of the men
Goal: Pair men and women up in a way such that
      two people from different pairings would be
      happier together if they switched partners

Algorithm:

    Each Day:
        1. Each man proposes to the first woman on
           his list who has not yet rejected him
        2. Each woman says "maybe" to the man she
           likes best among her proposals and
           rejects the others
        3. Each man crosses off his list the woman
           who just rejected him
    Repeat until there are no rejections
```

1.3.2 Proving the Algorithm

There are four things we must prove in order to make sure that the stable marriage algorithm works.

1. SMA terminates
2. SMA must output a pairing
3. The pairing outputted by SMA is stable

Beginning with proving the SMA terminates,

Proof 1 (SMA Terminates) *On each iteration, when SMA doesn't halt, some man got rejected. If a man gets rejected, he crossed a woman off his list. If every man gets rejected by every woman on his list, then there is a maximum of n^2 rejections. This will take a maximum of n^2 days, therefore the algorithm terminates after n^2 days.*

Before proving that the SMA must output a pairing, it is first useful to prove something called the improvement lemma

Lemma 1 (Improvement lemma) *Suppose M proposes to W on iteration K. On every day $J \geq K$, W has said "maybe" to a man she likes at least as much as M*

Proof 2 (Improvement Lemma) *Proceed with induction on the number of days k*

Base Case: *$J = K$*
M proposes to W, so the statement holds because even if M is her worst proposal, she will only say "maybe" to a better man.

Induction Hypothesis: *Assume the lemma is true for day J*
Inductive Step:
On day J, W has said "maybe" to M' she likes at least as much as M. M' will propose again on day $J+1$, so no matter who else proposes to W, she will at least have M' who is better than M

Proof 3 (SMA outputs a pairing) *Suppose the algirhtm produces no pairing. Then there exists M who is rejected by all n women when the algorithm terminates. By the improvement lemma, all women at the end of the algorithm have a men whom they've said maybe to. So n women can only have rejected $n-1$ men which is a contradiction, therefore SMA must output a pairing.*

Before proving that the output of SMA is stable, we should define what stability means.

Definition 8 *A pairing outputted by SMA is stable if there is not a man and a woman who would rather be with each other than their current partners (a rogue couple).*

Proof 4 (SMA outputs a stable pairing) *Suppose the pairing outputted by SMA is* $\{...(M, W)...(M', W')\}$ *and* M *likes* W' *more than* W. *Since* M *likes* W' *more than* W, M *must have proposed to* W'. *By the improvement lemma,* M' *must be better than* M, *so* (M, W') *is not a rogue couple, so there are no rogue couples and the pairing is stable.*

Definition 9 *For a given* M *the optimal* W *is the highest woman on his list that he is paired with in any stable pairing*

Definition 10 *The male optimal pairing is the one in which every man is paired with his optimal woman.*

Proof 5 (SMA is male-optimal) *Proceed by induction on k (the number of days).* $\forall k \geq 0$, *on day* k, *no man gets rejected by his optimal woman.*

Base Case: $k = 0$
Nothing to prove, no rejections have happened

Inductive Step:
Assume for all $0 \leq k \leq j$, *no man is rejected by his optimal woman. Suppose for contradiction that on day* $j + 1$ M *gets rejected by his optimal woman* W^* *meaning some other man was accepted* M^*. *Since* W^* *is* M*'s optimal woman, there exists a stable pairing* $\{...(M, W^*)...(M^*, W')\}$ *since* W^* *prefers* M^* *to* M *because she rejected* M. *By the inductive hypothesis,* M^* *has not yet been rejected by his optimal woman, so he likes* W^* *at least as much as* W' *because there is a stabe pairing with* W'. *Hence* (M^*, W^*) *is rogue because they like each other better than their partners.*

Theorem 2 *The male-optimal pairing is the female-pessimal pairing.*

2 Polynomials

Definition 11 *A polynomial of a single variable* x *with degree d is a function of the from*

$$P(x) = \sum_{i=0}^{d} a_i x^i$$

In simplest terms, a polynomial is simply a function. The degree of the polynomial is its highest exponent. The a_i are called the coefficients of the polynomial.

Definition 12 *A value* $a \in \mathbb{R}$ *is a root of the polynomial* $P(x)$ *if* $P(a) = 0$

We say that two polynomials are equal when they are equal at all values x ($P(x) = Q(x)\ \forall x$)

2.1 Properties of Polynomials

1. A nonzero polynomial of degree d has at most d roots
2. Given $d+1$ pairs $(x_1, y_1)...(x_{d+1}, y_{d+1})$ with the x_i being distinct, there exists a unique polynomial of degree at most d such that $P(x_i) = y_i$, $\forall i \in [1, d+1]$

While seemingly complex, the second property tells us something very simple about polynomials: each polynomial of degree d is uniquely defined by $d+1$ points. Thus given a set of points, we can find the polynomial which yielded those points. This can be done by regression or solving a system of linear equations. A "faster" technique, however, is Lagrange interpolation.

Given points $(x_1, y_1), ..., (x_{d+1}, y_{d+1})$,

$$P(x) = \sum_{i=1}^{d+1} y_i \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$$

This works because we need $P(x_i) = y_i$. Looking at our formula, whenever $x = x_i$, the fraction term becomes 1 only for the ith term and 0 otherwise (because x_i is a root for the numerator for all terms except the ith one). That forces $P(x_i) = y_i$. We can use Lagrange Interpolation to prove other interesting facts about polynomials. As an example,

Lemma 2 *If a is a root of a degree of polynomial p, then we can write $p(x) = (x-a)q(x)$ for a polynomial $q(x)$ with degree $d-1$.*

2.2 Polynomials over Finite Fields

Normally, we treat polynomials over the real numbers. However, we can also consider polynomials over a finite field. An particularly useful field is a Galois Field, which is the integers modulo p. This is denoted by $GF(p)$. This field contains only the values between 0 and $p-1$.

Definition 13 *A polynomial $P(x)$ over a finite field $GF(p)$ is equivalent the function $Q(x) = P(x)$ mod p*

Notice a couple things about polynomials about finite fields.

- There are p^{d+1} polynomials of degree $\leq d$ over $GF(p)$ because there are $d+1$ points and we havce p choices for the values.
- Given $k \leq d+1$ points, we can find p^{d+1-k} unique polynomials which go through these points because we have $d+1-k$ points for which we can choose values, and there are p values to choose

2.2.1 Case Study: Secret Sharing

One interesting use of polynomials over finite fields is to force people to collaborate in order to obtain a secret. Suppose that we have n generals with whom we want to share a secret encoded by an integer s. If a single general obtained the secret, they could destroy the other generals, so we want to make sure that at least k generals must work together to learn the secret (because then its mutually assured destruction). If any group of less than k generals works together, they should get no information about s.

```
  - Choose a large prime number p: p > s, p > n
  - Construct a "random" polynomial q(x) w/
    degree (k-1) mod p such that q(0) = s
  - Give the ith general the value q(i) mod p
```

With this scheme, if k generals get together, then they can use lagrange interpolation to recover $q(x)$ and find s by computing $q(0)$. However if $d < k$ polynomials get together, there are going to be p^{k-d} possible polynomials, so any secret value will be possible.

2.2.2 Case Study: Error-Correcting Codes

Another use for polynomials over finite fields is in creating error-correcting codes. Suppose we have a message which is a sequence of integers $m_1, m_2, ..., m_n$. We are going to send this message over an unreliable channel, so we need to transmit some redundant information. Lets send code words $c_1, c_2, ..., c_m$ where $m > n$ to create the redundancy we need. Some of these packets may be dropped or modified, so we will receive a message $r_1, r_2, ..., r_p$.

Suppose we know the following about the channel:

- Up to k of the c_i will get erased, and we will know which ones do
- Up to k of the c_i will be corrupted and we don't know which ones.
- The maximum number of errors is k

Since we are tolerating k errors, lets send $n + 2k$ codewords over the channel. That way we are guaranteed $n + k$ of the r_i are equal to the corresponding c_i.

1. Define $q > n + 2k$ to be a large prime so we can encode our message as integers mod q
2. Let $P(x)$ be a unique degree $n - 1$ polynomial over $GF(q)$ such that $P(i) = m_i,\ i \in [1, n]$
3. Compute $2k$ extra values $p(n + 1)...p(n + 2k)$
4. Send all points $c_i = p(i),\ i \in [1, n + 2k]$

We know we will receive at least $n+k$ of our packets, but some of them may be modified. There must a degree $n-1$ polynomial p which goes through at least $n+k$ of these points (i, r_i). Suppose the location of the errors are $e_1, e_2, ..., e_k$ We can define the error locator polynomial

$$E(x) = (x-e_1)(x-e_2)...(x-e_k) = x^k + \sum_{i=1}^{k-1} b_i x^i$$

Now notice that $P(i)E(i) = r_i E(i)$ since $P(i) = r_i$ for the non-error points, and $E(i) = 0$ at the error points. Now define

$$Q(x) = P(x)E(x) = \sum_{i=0}^{n+k-1} a_i x^i$$

Since $Q(i) = r_i E(i)$, this gives us $n+2k$ equations which we can use to find the coefficients of $Q(i)$ and $E(i)$. We can now simply find $P(x) = \frac{Q(x)}{E(x)}$.