General Proof Techniques and Formats

Please put your proofs in these formats from now on including the subsection labels: Given, Want to Show (WTS), Conclusion etc.

Direct Proof (Proof by Construction)

In a constructive proof one attempts to demonstrate $P$ to $Q$ directly. In other words you are proving a statement by constructing something that verifies it. Here is the format for this proof:

- **Given**
  Name set variables, machines, etc

- **Want to Show (WTS)**
  State goal and outline plan

- **Construction**
  Using objects previously defined new tools, work toward the goal. Give formal definitions and explain.

- **Correctness**
  Here, if you constructed a machine $M$, you must prove that it actually accepts a language $L$. You have to (1) show that $M$ accepts all strings in language $L$ (2) show that $M$ rejects all strings not in language $L$. (Check out how to formally show acceptance via a path through a machine on the bottom of page 40 in the US textbook)

- **Conclusion**
  Recap what you proved.

Proof by Contradiction

The proof by contradiction is grounded in the fact that any proposition must be either true or false, but not both true and false at the same time. We arrive at a contradiction when we are able to demonstrate that a statement is both simultaneously true and false, showing that our assumptions are inconsistent. Here is the format:

- **Given**
  Statements, set variable names, machines, etc
**WTS**
State goal and outline plan

**Contradiction**
If you're trying to prove $P \rightarrow Q$:
1. Assume that $P$ is true.
2. Assume that not $Q$ is true.
3. Use the above to demonstrate a contradiction.

**Conclusion**
Recap what you proved.

---

**Proof by Induction**

Proof by induction is a very powerful method in which we use recursion to demonstrate an infinite number of facts in a finite amount of space. The most basic form of mathematical induction is where we first create a propositional form whose truth is determined by an integer function. If we are able to show that the propositional form is true for some integer value then we may argue from that basis that the propositional form must be true for all integers. Format:

- **Given**
  - Statements, variable names, equations, etc

- **Hypothesis or WTS**
  - State goal or statement we are trying to prove.

- **Basis Step**
  - Show that the hypothesis is true for the base value.

- **Induction Step**
  - Assuming that our hypothesis is true for any arbitrary step $k$, show that our hypothesis holds for the next step: $H(k + 1) = H(k) + (k + 1)$

- **Conclusion**
  - Recap what you proved.

---

**HW1 Common Issues**

Note: For this homework, we were instructed to be very lax and take minimal points away for small mistakes. Dr. Pennycuff has reiterated that this will not be the case starting with homework 3.

**Problem 1: Induction Proof**

Since we didn’t mention any formal proof formats, here we were looking for 2 main things:
1. The basis step was explicitly solved at $n=1$
2. In the induction step, $F(k+1) = F(k) + (k+1)$ and the simplification was shown.
The easiest, most succinct way of doing the induction step is setting up the LHS and RHS to be \( C(k+1) \) and \( C(k) + (k+1) \), respectively, then simplifying each side to a point where equivalence is clear.

**Problem 2: Proof by Contradiction**

One of the biggest issues with this problem was it was not always clear how someone was trying to prove this. Formalize your proof’s using the structure above and EXPLICITLY state which proof method you are using at the beginning. Reading through a solution where you begin explaining logic without knowing how you are trying to prove something can be difficult for us to understand.

Informal proofs aside, we were looking for 2 main points here:
1. Recognize that the set of possible degrees is \([0,n-1]\) since we are proving this for general graphs.
2. Show that it is a contradiction to have an unconnected node with degree 0 in the same graph that has a fully connected node with degree \( n-1 \).

**Problem 3: Designing Finite Automata**

Most people did well on this section. The common problem was confusion over the formal definition of your machine for each problem. Remember, you need to both visually draw the automata while also correctly stating its formal tuple definition.

The formal definition for a machine \( M \) is: \( M = (Q, \Sigma, \delta, q_0, F) \) where \( Q, \Sigma, \) and \( F \) are SETS denoted by \{..\}

Example: \( M = (\{q_0,q_1,q_2,q_3, sink\}, \{a,b,c\}, \delta, q_0, (q_2,q_3)) \)

\( \delta = ..table.. \)

**HW2 Common Issues**

**Problem 1: Closure Proof**

**Operations on Regular Expressions**

Many people who tried to approach this problem from the perspective of regular expressions had a lot of good ideas (\( A \) is a regular language, therefore \( A \) is described by a regular expression, etc), but didn’t really specify an operation on regular expressions that took advantage of the recursive nature of regular expressions. In general, when something is defined recursively (like regular expressions), doing an operation on it is often best done by defining each case, recursively.

For example, if you wanted to define an operation \( \text{Double} \) on regular languages that for any language \( L \), it would hold that

\[ \text{Double}(L) = \{w_0w_0w_1w_1 \cdots w_nw_n \mid w_0w_1 \cdots w_n \in L\} \]

and you wanted to prove that the regular languages were closed under this operation, (and you wanted to prove this using regular expressions) you might proceed as follows:

“We aim to show \( \text{Double}(A) \) is regular if \( A \) is regular. We know a language \( A \) is regular iff it is recognized by a regular expression. Let’s assume \( R_1 \) is a regular expression such that \( \mathcal{L}(R_1) = A \). In order to show that \( \text{Double}(A) \) is regular, it is sufficient to describe an operation \( f \) on regular expressions such that for any regular expression \( R \), \( \mathcal{L}(f(R)) = \text{Double}(\mathcal{L}(R)) \). \( f(R) \) is defined recursively as follows:

- If \( R \) is \( a \) for some \( a \in \Sigma \), \( f(R) = a \)
If \( R \) is \( \epsilon \), \( f(R) = \epsilon \)

If \( R \) is \( \emptyset \), \( f(R) = \emptyset \)

If \( R \) is \( R_1 \cup R_2 \) for some regular expressions \( R_1 \) and \( R_2 \), \( f(R) = f(R_1) \cup f(R_2) \)

If \( R \) is \( R_1 \circ R_2 \) for some regular expressions \( R_1 \) and \( R_2 \), \( f(R) = f(R_2) \circ f(R_1) \)

If \( R \) is \( (R_1)^* \) for some regular expression \( R_1 \), \( f(R) = (f(R_1))^* \)

(I’ll omit proving that \( L(f(R)) = \text{DOUBLE}(L(R)) \), and usually you can as well if what you’re doing is pretty self explanatory.)

Since for any regular language \( L \), you can create a regular expression recognizing \( \text{DOUBLE}(L) \), the regular languages are closed under \( \text{DOUBLE} \).

Note how the construction very much follows the recursive definition (definition 1.52) of regular expressions on page 64 of the Sipser textbook (US edition).

Operations on Automata

Please follow the proof layout for construction above. Many people just constructed a finite automata without a proof of correctness.

For this problem, we were looking for 5 things:

- State that an automata accepts the language \( L \)
- Reverse transitions (next time you should formalize using the delta function)
- Explain the new accept state
- Explain the new start state (including the epsilon transitions to all old accept states)
- Correctness: discuss the paths/connectivity of the automata

Problem 2: Proving a Regular Language

Again, please formalize proofs. Most people understood the automata construction very well.

Some people are confused by the fact that the alphabet contains characters that are size 3 vectors and drew automata with transitions with \( \{0,1\} \). Please review the definition of finite automata and \( \Sigma \).

Problem 3: Proving a Regular Language

Main Issues:

- Formalize proofs with the construction format above.
- Just tracking carry in this problem is not enough (this deals with the addition part of \( 2x + x \) but not the bit-shift for \( 2x \)).

Problem 4: NFA and DFA

Main Issues:

- Many students did not take into account all edge cases of a regular expression. For example, in \( L2 = \{ w \mid w = 0^n(01)^* \} \), you must consider all cases: \( \epsilon, 0, 00, 01, 0101, \ldots, 01, 00, 010101, \ldots \)
• Arbitrary amount of repeating digits were not taken care of in NFAs for a number of students.
• Many accept states were missed while considering the edge cases above.