

## The Anatomy of a Formal Proof

Think of every construction-based proof as having these five essential sections. Following this template will ensure your arguments are complete and easy to follow. We give examples based on proofs for regular languages, but the same logic holds for all topics in this class.

### 1. State Your Goal and Strategy (The "What" and "How")

Begin with a clear, one-sentence statement of what you intend to prove and the method you will use. This acts as a thesis statement for your proof.

- What to write: "We will prove that the class of regular languages is closed under the [Operation Name] operation. We will do this by construction, showing how to create a new automaton for the resulting language."
- Common Strategies: "Proof by Construction," "Proof by Induction," "Proof using Closure Properties."

### 2. The Setup (The "Given")

Formally define what you're starting with. If a prompt says "Let L be a regular language," your setup should translate that into a formal object you can work with.

- What to write: You are given a regular language L. By definition, this means there is a machine that recognizes it. Define that machine using its formal 5-tuple notation.
- Example:  
Let L be a regular language over an alphabet  $\Sigma$ . By definition, there exists a Deterministic Finite Automaton (DFA), M, that recognizes it. We define this machine as:  $M=(Q,\Sigma,\delta,q_0,F)$  where  $L(M)=L$ .

### 3. The Construction (The "Recipe")

This is the creative core of the proof. You must describe exactly how to build the new machine that recognizes the target language.

- What to write:
  - Start by stating you are constructing a new machine (e.g., an NFA named N or a DFA named M').
  - Define the new machine using its 5-tuple notation.
  - Crucially, define every component of your new machine (its states, alphabet, transitions, start state, and accept states) in terms of the components of the original machine from the setup.
- Example:  
We construct a new DFA, M', that recognizes the target language. We define M' as:  $M'=(Q',\Sigma,\delta',q'_0,F')$  where:

- $Q' = \dots$  (defined in terms of  $Q$ )
- $\delta'(q, a) = \dots$  (defined in terms of  $\delta$ )
- $q_0' = \dots$  (defined in terms of  $q_0$ )
- $F' = \dots$  (defined in terms of  $F$ )

#### 4. Proof of Correctness (The "Why It Works")

You can't just provide the construction; you must prove that it works. This requires showing that the machine you built accepts the language you claim it does, and only that language. This is a two-part argument.

- What to write: You need to show that "a string  $w$  is accepted by your new machine if and only if  $w$  is in the target language." This means proving two directions:
  1. Direction 1 ( $\Rightarrow$ ): Assume an arbitrary string  $w$  is in the target language. Then, using the definition of the target language, show how your constructed machine follows a path to an accept state.
  2. Direction 2 ( $\Leftarrow$ ): Assume your machine accepts an arbitrary string  $w$ . Then, using the definition of your machine's components, show that the string  $w$  must satisfy the properties of the target language.

#### 5. Conclusion (The "Q.E.D.")

End with a final, declarative sentence that restates what you have proven. This wraps up the argument neatly.

- What to write:
  - "Therefore, the resulting language is regular, and the class of regular languages is closed under the [Operation Name] operation."
  - "Since we have successfully constructed an automaton for the language, the language is regular."

#### Tips for Excellent Proofs

- Add an "Intuition" Section: Before the formal construction, it's very helpful to include a brief, plain-English paragraph explaining your core idea. This shows the reader you understand the concept behind the formalism.
- Cite Previous Results: If you use a property that was proven earlier in the course (e.g., that finite languages are regular, or that regular languages are closed under union), state it explicitly. This is a key skill in more complex proofs.
- Handle Special Cases: Always consider the empty string ( $\epsilon$ ). Ask yourself if your construction correctly accepts or rejects  $\epsilon$  based on the language's definition. Making a small note about this shows thoroughness.