

Review

This is a review of what you need to be able to do for the final exam:

1. Prove a property of the natural numbers by *simple induction*:
 - (a) Declare clearly what the $S(n)$ is.
 - (b) State what you are proving, that is, $S(n)$ for what values?
 - (c) Prove that $S(n)$ holds for the base case.
 - (d) State what you are assuming about $S(n)$ in the induction step.
 - (e) Prove that $S(n + 1)$ holds.
2. Prove a property of the natural numbers by *complete induction*:
 - (a) Declare clearly what the $S(n)$ is.
 - (b) State what you are proving, that is, $S(n)$ for what values?
 - (c) State what you are assuming about n .
 - (d) State what you are assuming about numbers $i < n$ and $S(i)$.
 - (e) Prove that $S(n)$ holds.
 - You will likely need to split cases.
 - Make sure that all the cases for n are handled.
 - For cases where the induction hypothesis is needed, make sure that the assumption made in step (d) covers the case.
3. Given an iterative program, prove that some property (involving the variables) in the program is a *loop invariant*.
 - (a) Formulate this as property of the natural numbers: for all $n \in \mathbb{N}$, if the program does n iterations, then some property holds.
 - (b) Prove this by simple induction.
 - For the base case, use the initial values of the variables.
 - Otherwise, use the new values of the variables after going through the body of the loop in terms of the previous values.
4. Given an *iterative* program, prove it *partially correct* for some precondition and postcondition. This means proving that if the precondition holds and the program terminates then the postcondition holds.
 - (a) State that you are assuming that the program terminates.
 - (b) State what you can conclude about the variables in the program by inspecting the termination condition on the loop.
 - (c) Use this and the precondition and any given or proven loop invariants to prove that the postcondition holds.

5. Given an *iterative* program, prove that the program *terminates*.
 - (a) Find an expression E_n in terms of the variables in the program.
 - (b) Prove that if the program does n iterations, $E_n \in \mathbb{N}$.
 - (c) Prove that if the program does $(n + 1)$ iterations, $E_{n+1} < E_n$.

For (b) and (c), use the precondition and any given or proven loop invariants.
6. Given a *recursive* program, prove it *correct* for some precondition and postcondition.
 - (a) Prove that the recursive procedure always terminates and returns a suitable value.
 - This will usually be a proof by simple or complete induction over the arguments to the procedure, depending on the nature of the recursive calls.
 - For procedures with more than one argument, the induction might be on one of the arguments or on the sum or difference of the arguments.
 - (b) Using this and the given precondition, prove that the program terminates and satisfies the postcondition.
7. Given a recurrence relation, solve it using the method of *repeated substitutions*:
 - (a) Express $T(n)$ in terms of T of smaller and smaller values until some sort of pattern emerges. This may be $T(n/2)$, $T(n/4)$. . . , or $T(n - 1)$, $T(n - 2)$. . . , or something else.
 - (b) Find a value (or values) of n to remove the $T(n)$ from the RHS.
 - (c) If necessary, prove the solution correct using induction.
8. Analyze the *time complexity* of a *recursive* program:
 - (a) Given a recursive procedure, write its time complexity as a recurrence relation.
 - (b) Apply the Divide and Conquer Theorem to get tight bounds on the time complexity, when the recurrence relation has the appropriate form.
Note: It is not necessary to memorize the Theorem. It will be given, if needed.
9. Prove that a property holds of every string in an inductively defined language \mathcal{L} by *structural induction*:
 - (a) Declare clearly what the $P(u)$ is. You may need to define a function from \mathcal{L} to \mathbb{N} .
 - (b) State what you are proving, that is, $P(u)$ for what values?
 - (c) Prove that P holds for the base case(s) of \mathcal{L} .
 - (d) State what you are assuming about P in the induction step.
 - (e) Prove that P holds for the inductive case(s) of \mathcal{L} .
10. For a given first-order logical language \mathcal{L} , relate *sentences* to *structures*, as follows:
 - Given a sentence F in \mathcal{L} and a structure \mathcal{S} , state whether F is true/false in \mathcal{S} .
 - Given a sentence F in \mathcal{L} , show a structure \mathcal{S} where F is true/false in \mathcal{S} .
 - Given a structure \mathcal{S} , write a sentence F that is true/false in \mathcal{S} .

For all of these, you need to be able to do the following:

- (a) Construct sentences as formulas with no free variables.
- (b) Specify a structure by specifying a domain, an interpretation for the constant symbols, and an interpretation for the predicate symbols (using the superscript notation).
- (c) Decide if a sentence is true or false in a structure using the rules for truth for the logical connectives and the quantifiers.

11. For a given first-order logical language \mathcal{L} , do the following:

- Prove that a sentence F is not logically equivalent to another sentence G .
- Prove that a set of sentences T does not logically imply another sentence G .

For these, you need to be able to do the following:

- (a) To prove non-equivalence, find a counterexample structure where one sentence is true and the other is false.
- (b) To prove non-implication, find a counterexample structure where all the sentences in T are true, but G is false.

12. For a given first-order logical language \mathcal{L} , do the following:

- Prove that a sentence F is logically equivalent to another sentence G .
- Prove that a set of sentences T logically implies another sentence G .

For these, you need to be able to do the following:

- (a) For any sentence in the language, translate the truth or falsity of that sentence into a condition on a structure.
- (b) For equivalence, prove that for an arbitrary structure, the condition for one sentence holds iff the condition for the other sentence holds.
- (c) For implication, prove that for an arbitrary structure, if the conditions for all the sentences in T hold, then the condition for G holds.

13. To relate automata to languages, do the following:

- Draw a DFSA or a NFSA M such that $\mathcal{L}(M)$ is a given language.
- Write a regular expressions R such that $\mathcal{L}(R)$ is a given language.
- Describe $\mathcal{L}(M)$ for a given DFSA or NFSA M .
- Describe $\mathcal{L}(R)$ for a given regular expression R .
- Use the subset construction to construct a DFSA that accepts the same language as a given NFSA.

14. For a given DFSA M , prove that M accepts a given language:

- (a) Determine what the extended transition function δ^* is from the initial state q_0 . Write this as an equation for $\delta^*(q_0, x)$. There must be exactly one case that applies for each possible value of x .
- (b) Prove by induction that the δ^* satisfies this definition. This can be by induction on the length of x or a structural induction over strings.
- (c) Use this the property of δ^* to conclude that $x \in \mathcal{L}(M)$ iff x is in the given language.