**Part 1:**

**Prove a DFA correct:**

* See the ‘DFA-correctness’ handout.
* Outline the ‘status’ (properties) of each state.
* Complete a proof by induction on all possible combinations of symbol and state to justify the transition function.
* Alternatively, state minimization can be shown through a set of pairwise distinguishable strings (one state per string for a minimized DFA)

**Prove a language is not regular:**

* Create an infinite set of pairwise distinguishable strings, such that all strings in the set are either in the language or not in the language.
* Then, for any two strings of the set, show a generalized suffix such that one string is in the language and the other is not.

**Parameterized (generalized) DFA construction:**

* Do some examples with actual numbers to understand the pattern. Generally, these problems use some sort of “pair” or “counting” system to parameterize the number of states.

**Construction of NFAs taking advantage of nondeterminism:**

* Using nondeterminism should attempt to keep as many branches open as possible at a time.

**Closure under operations in Regular Languages:**

* Show that the operation can be expressed as a DFA (dictionary, transition-function, etc.)
  + This usually involves a composition of states, counters, and transitions)
* Then prove that the DFA recognizes the language of interest.

The main idea here is that we prove closure if we can create a DFA that does the operation in question.

**Convert NFA to DFA:**

* Use subset construction (shown in class and homework)
* Recall that epsilon-transitions allow for instant movement between states (good for “optional” portions of regular expressions or looping)

**Part 2:**

**Prove a language is undecidable:**

* Use problem reduction on a previously known undecidable language and conduct a contradiction proof.

**Prove a language is decidable:**

* Construct a halting TM to show that the language is decidable.
* Make sure to highlight that the TM halts on every input (finite set, etc.)

**Prove a language is recognizable:**

* Construct a TM for some language.
  + The TM should implement an enumeration over the operation provided in some manner
  + The TM is usually non-haling, which means it has to have two sets of loops, one over some integer k which indicates how many steps to run the machine.
* If it is non-halting for certain inputs highlight this as part of the explanation.

**Prove a language is recognizable-but-undecidable:**

* First show the language is recognizable.
* Then show the language is undecidable.

Note that the complement to this language is unrecognizable.

**Four-part language classification:**

* Does the solution imply the problem is decidable: maybe if the solution is halting.
* Does the solution imply the problem is recognizable: yes, as the solution (TM) is valid, the problem is halting.
* Does the solution imply the problem is undecidable: no, as there could potentially be a halting solution that is not covered.
* Is the problem decidable: maybe if you can design a halting TM for the solution (typically it involves using the halting or finite properties of DFAs in meta context on a TM)

**Proving close under a operation on a recognizable language:**

* Design a TM which executes the operation on some string.
* Iterate through all k positive integers (outer loop)
* For each k, execute k steps of the original TM on each string of the operation.
* Justify how this machine recognizes success but is potentially non-halting.

**Basic language properties:**

* Undecidable languages: uncountable set
  + Ex: Irrational numbers
* Set difference: closed under decidable languages, not closed under recognizable languages.
* Polynomial problem: usually undecidable but read closely.
* Program termination on all values: Inherently undecidable, discussed in class.
* NFAs = DFAs and they are always halting. Therefore a TM can find the set of all accepted strings on an NFA/DFA
* Regular languages are closed under intersection and complement
  + Lecture slides on closure have more properties, and so does the textbook

**Construct a TM:**

* Essentially just think of an algorithm and then write it out into a TM
* Recall some basic TM ‘modules’ to use:
  + Such as shifting all elements to the right or left by one space (4-state setup with swappers)
  + Use ticked (‘) versions of characters to indicate some operation or action ahs been performed on them (they have already been scanned)

**Outline the buggy statement:**

* Usually appears on some kind of proof by reduction problem
* The buggy statement is almost always (n = 2) the declaration of results from R accepting/rejecting M’ which overestimates or neglects the two possibilities on M’ (i.e M accepts w and M rejects w, what happens to M’ in each case)

A close up of text

Description automatically generated

Decidable languages are closed under ∪, °, \*, ∩, and complement

A white background with black text

Description automatically generated

**Part 3:**

**Prove a language is in P:**

* Create a basic polynomial-time (and space) algorithm to solve a problem

**Prove a language is in NP:**

* Show the existence of a polynomial time verifier that accepts a polynomial-space certificate to resolve the problem (recall that P is a subset of NP and therefore this would also work for a language in P)
  + Alternatively may use the nondeterministic guess() function to keep polynomial time with “guessing” (iterating through nondeterministic options)

**Prove a language is NP-Complete:**

* Prove that the language is in NP
* Prove that the language is in NP-hard via reduction