In this programming assignment, you will take the data type of propositions defined in class and write simple programs to manipulate them.

**type** prop = Letter **of** string | T | F

    | Not **of** prop | And **of** prop \* prop

    | Or **of** prop \* prop | Impl**of** prop \* prop  | Iff **of** prop \* prop ;;

Already implemented for you are:

1. ht: prop -> int, which returns the height of a proposition (syntax tree).
2. size: prop -> int, which returns the number of nodes in a proposition (syntax tree)
3. truth: prop -> (string -> bool) -> bool, which evaluates a proposition with respect to a given truth assignment to propositional letters.
4. nnf: prop -> prop, which converts a proposition into negation normal form.
5. subst: prop -> (string -> prop) -> prop, which applies a substitution to a prop.
6. kleisli:  (string -> prop) -> (string -> prop) -> prop -> prop, which composes two propositions.

In light of the Relevance Lemma, you may wish to choose a compact representation of **valuations**, instead of simply string -> bool.

The functions you need to implement are:

Syntactic

1. vars: prop -> string set, which returns the set of propositional letters that appear in a proposition.
2. cnf: prop -> prop set set, which converts a proposition into conjunctive normal form (POS) as a set of clauses, each of which is a set of **literals** (a special subset of prop, comprising only letters and negations of letters).
3. dnf: prop -> prop set set,  which converts a proposition into disjunctive normal form (SOP) as a set of terms, each of which is a set of literals.

Semantic

1. isSatisfiable: prop -> bool, which checks if a proposition is satisfiable,
2. satisfier: prop -> valuation, which returns a satisfying truth assignment if it exists.
3. isFalsifiable: prop -> bool, which checks if a proposition is satisfiable
4. falsifier: prop -> valuation, which returns a falsifying truth assignment if it exists.
5. models: prop -> valuation set, which returns the set of ``all valuations'' that satisfy a given prop.
6. isTautology: prop -> bool, which checks if a proposition is a tautology.
7. isContradiction: prop -> bool, which checks if a proposition is a tautology.
8. isEquivalent: prop -> prop -> bool, which checks if two propositions are logically equivalent.
9. entails: prop set -> prop -> bool, which checks if a given proposition is a logical consequence of a given set of propositions.

Wherever possible, use the list functions map, filter, foldr (or foldl).

For each of your functions, indicate the TIME and SPACE complexity in terms of the size of the proposition and number of distinct letters.