**Project 2: Boolean Inference Checker**

**1. Data Model**

A ***logical variable*** is a string of one or more lowercase letters *a,...,z,* not beginning with *v.* For example, the following are logical variables: "p", "foo", "qrs".

A ***symbol*** is either a logical variable or one of the following eleven strings:  "T", "F", "^", "v", "~", "=>", "<=>", "(", ")", ":.", "," Note that, by comparison with the Boolean expression evaluator, the new symbols are logical variables, the comma, and the *therefore* symbol ":.".

A ***symbol* *string***is the concatenation of zero or more symbols and/or spaces. For example, the following are symbol strings:

* "T :.  "
* "=> Tv~  F(( F"
* "T foo  => F ^ (F pr :. ) v F)"
* ""
* "  "
* "Foo"

and the following are not symbols strings

* "T X  "
* "  : .F => T"
* "TF p\*^)"
* "FOO"

A ***Boolean expression*** is a vector of symbols defined as in the [previous assignment](https://docs.google.com/document/d/1xDkMRJYBLOVNuZPxWJMcsI3X-ft4M5SnUFnzpBPmFIM/edit?usp=sharing), with the exception that, in addition to other grammar rules, logical variables are also considered to be unbreakable expressions. So the only rule that chances is the rule for unbreakable expressions:

*U* ::= *Const* |  "(" B ")" | *Lvar*

An ***Boolean inferenc***e consists of one or more Boolean expressions separated by commas, followed by a *therefore* sign ":.", followed by a Boolean expression. Formally,

*Ps*  ::= *Bexp* | *Bexp* "," *Ps*

*Inf*  ::=  *Ps* ":." *Bexp*

An AST is as in the previous assignment.

An ***inference*** is a struct with two attributes:

* *premises --* a vector of AST's.
* *conclusion --* an AST

**2. Functions**

The Boolean expressions *p*1,...*pn* are the ***premises*** of the Boolean inference

*p*1,...,*pn* :. *q*

and *q* is its ***conclusion***.

A Boolean inference is ***invalid***if there is no consistent substitution of truth values ("T" and "F") for its logical variables that makes all of its premises true and its conclusion false. Otherwise, it is ***valid.***

For example, the following strings represent valid inferences:

* "p => q, ~q :. ~p"
* "p ^ ~q => r ^ ~r :. ~p v q"
* "p => q, q => r, r => s v t, ~s, p :. t"

and the following strings represent invalid inferences:

* "p => q, ~p :. ~q"
* "p => q, q => r  :. q => p ^ r"

The following eight functions shall be written for this assignment:

list<string> Insert(string s, list<string> \*L)

* *Precondition: L* is a list of distinct strings, sorted alphabetically
* *Effects*:
  + if *s* is already in *L,* there is no effect.
  + If *s* is not already in *L*, it is added to *L* so that the length of *L* increases by 1 and L remains sorted.
* *performance*: for strings of length less than 100, this function should run in *O*(log *n*) time, where n is the length of L.

list<string> vars(AST T)

* If *T* is an AST of a Boolean expression, then *vars*(*T*) is a list of all variables occurring in *T*, sorted in alphabetical order, with duplicates eliminated.

list<string> vars(vector<AST> Ts)

* If *Ts* is a list of AST's of Boolean expressions, then *vars*(*Ts*) is a list of all variables occurring in the members of *Ts*, sorted in alphabetical order, with duplicates eliminated.

list<bool> bits(int i, int N)

* if *N* > 0 and *i* is an integer in {0,...2*N*-1}, then *bits*(*i,N*) is the *N-*bit representation of *i* as an integer. For example *bits*(0,3) = [0,0,0]; *bits*(3,3) = [0,1,1], *bits*(6,3) = [1,1,0].

AST substitute(list<bool> vals, list<string> vars, AST Exp)

* If *vars* and *vals* are the same length,  then *substitute*(*vals, vars, Exp*) is the AST obtained from *Exp* by substituting "T" for *vars*[i] *w*henever *vals*[*i*]=1, and "F" for *vars*[*i*] *whenever vals*[*i*] = 0.

bool witnessInvalid(list<bool> vals, list<string> vars, inference I)

* If *eval*(*substitute*(*vals,vars,p)*) is true for every premise *p* of *I,* and *eval*(*substitute*(*vals, vars, I.conclusion*)) is false, then *witnessInvalid*(*vals, vars, I*) is true;
* Otherwise, *witnessInvalid*(*vals, vars, I*) is false

bool valid(inference I)

* *valid*(*I*) is *true* if *I* is valid, and *false* otherwise.
* That is, if *Vars* is a list of the distinct variables occurring in *I,* and *N* = *length*(*Vars*), then
  + if there is an *i* in [0...2*N*-1] such that *witnessInvalid*(*bits*(*i,N*),*Vars,I*) is true, then *valid*(*I*) is false.
  + otherwise, *valid*(*I*) is true.

string validInference(string s)

* if *s* is not a symbol string, *validInference(s*) is "symbol error"
* if *s* is a symbol string, but the tokenization of s does not parse as an inference, *validInference*(*s*) is "grammar error"
* If *s* is a symbol string that parses as an inference, then *validInference*(*s*) is "valid" if the inference is valid, and "invalid" otherwise.
* For example,
  + *validInference*("=> u F \*") is "symbol error"
  + *validInference*("u => :. :. F pr") is "grammar error"
  + *validInference*("u => p, u :. p") is "valid"
  + *validInference*("p v q, ~q :. p") is "valid"
  + *validInference*("pa v q, ~pa v ~q :. pa <=> q") is "invalid".