MTT is intended to be used as a universal Tarski meta-language including a meta-language to itself.
Because MTT has its own provability operator: "⊢" provability can be analyzed directly within the deductive inference model instead indirectly through diagonalization. This allows us to see exactly why an expression of language can be neither proved nor disproved, details that diagonalization cannot provide. **All of the symbolic logic operators retain their conventional semantic meaning from mathematical logic.**

```
%left  IDENTIFIER         //  Letter+ (Letter | Digit)*  // Letter includes UTF-8
%left  SUBSET_OF          //  ⊆
%left  ELEMENT_OF         //  ∈
%left  FOR_ALL            //  ∀
%left  THERE_EXISTS       //  ∃
%left  IMPLIES            //  →
%left  PROVES             //  ⊢
%left  IFF                //  ↔
%left  AND                //  ∧
%left  OR                 //  ∨
%left  NOT                //  ~
%left  ASSIGN_ALIAS       //  :=  LHS is assigned as an alias name for the RHS (macro substitution)
%%                        //  An alias named expression is treated syntactically as a propositional
                          //  variable in the next higher level of logic specifying HOL using FOL syntax.
                          //  This alias name is then treated semantically as if it was macro expanded.
sentence
      : atomic_sentence
      | '~' sentence %prec NOT
      | '(' sentence ')'
      | sentence   IMPLIES      sentence
      | sentence   IFF          sentence
      | sentence   AND          sentence
      | sentence   OR           sentence
      | quantifier IDENTIFIER   sentence
      | quantifier IDENTIFIER   type_of IDENTIFIER sentence  // Enhancement to FOL
      | sentence   PROVES       sentence                     // Enhancement to FOL
      | IDENTIFIER ASSIGN_ALIAS sentence                     // Enhancement to FOL
      ;

atomic_sentence
      : IDENTIFIER '(' term_list ')' // ATOMIC PREDICATE
      | IDENTIFIER                   // SENTENTIAL VARIABLE  // Enhancement to FOL
      ;

term
      : IDENTIFIER '(' term_list ')' // FUNCTION
      | IDENTIFIER                   // CONSTANT or VARIABLE
      ;

term_list
      : term_list ',' term
      | term
      ;

type_of
     : ELEMENT_OF                                           // Enhancement to FOL
     | SUBSET_OF                                            // Enhancement to FOL
     ;

quantifier
     : THERE_EXISTS
     | FOR_ALL
     ;
```