

Formal specification of Minimal Type Theory

This is the formal YACC BNF specification for Minimal Type Theory (MTT). MTT was created by augmenting the syntax of First Order Logic (FOL) to specify Higher Order Logic (HOL) expressions using FOL syntax.

This second-order sentence $\forall P \forall x (Px \vee \neg Px)$ says that for every formula P, and every individual x, either Px is true or not(Px) is true (this is the principle of bivalence).¹

Here it is encoded using first order logic syntax and the definition operator:

(a) $S1 := \forall x (Px \vee \neg Px)$

(b) $\forall P(S1)$

Every instance of the left-hand-side of a definition is to be expanded into its right-hand-side.

Thus the above two lines specify: $\forall P(\forall x (Px \vee \neg Px))$.

MTT can also correctly parse second order logic expressions directly: $\forall P \forall x (P(x) \vee \sim P(x))$

```
<sentence_4 token="FOR_ALL">
  <sentence_4 token="IDENTIFIER" value="P"/>
  <sentence_4 token="FOR_ALL">
    <sentence_4 token="IDENTIFIER" value="x"/>
    <sentence_13 token="OR">
      <atomic_sentence_1 token="IDENTIFIER" value="P">
        <term_2 token="IDENTIFIER" value="x"/>
      </atomic_sentence_1>
      <atomic_sentence_1>
        <sentence_2 token="NOT">
          <atomic_sentence_1 token="IDENTIFIER" value="P">
            <term_2 token="IDENTIFIER" value="x"/>
          </atomic_sentence_1>
        </sentence_2>
      </atomic_sentence_1>
    </sentence_13>
  </sentence_4>
</sentence_4>
```

Another key use of the definition operator is to properly formalize self-referential expressions.

“This sentence is not true.” would be formalized as:

LP := $\sim \text{True}(LP)$ which expands into: $\sim \text{True}(\sim \text{True}(\sim \text{True}(\sim \text{True}(\dots))))$ infinite recursion.

“This sentence is not provable.” would be formalized as:

G := $\sim \text{Provable}(G)$ which expands into: $\sim \text{Provable}(\sim \text{Provable}(\sim \text{Provable}(\dots)))$ infinite recursion

That macro expansion results in infinite recursion is documented in this article:

3.10.5 Self-Referential Macros

A self-referential macro is one whose name appears in its definition. Recall that all macro definitions are rescanned for more macros to replace. If the self-reference were considered a use of the macro, it would produce an infinitely large expansion. ...

Following the ordinary rules, each reference to foo will expand into (4 + foo); then this will be rescanned and will expand into (4 + (4 + foo)); and so on until the computer runs out of memory.

<https://gcc.gnu.org/onlinedocs/cpp/Self-Referential-Macros.html>

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: “ \vdash ” 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.**

¹ Second-order logic, https://en.wikipedia.org/w/index.php?title=Second-order_logic&oldid=979950180

```

%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   // := (definition operator) x := y means x is defined to be another name for y
                    // 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.

definition
  | sentence
  : IDENTIFIER ASSIGN_ALIAS sentence // Enhancement to FOL

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
  ;

```