Minimal Type Theory (MTT) shows exactly how all of the constituent parts of an expression relate to each other (in 2D space) when this expression is formalized using a directed acyclic graph (DAG). This provides greater expressiveness than the 1D space of FOPL syntax.

X @ ~True(X) // assign alias operator "@" explained
"@" means the LHS is assigned as an alias for the RHS .
This extension to FOPL syntax provides the means for:
(1) Meaningful names to be assigned to expressions.
(2) Predicates to have other Predicates as terms. // enabling HOL of an unlimited finite order
(3) An Expression to refer directly to itself.

https://en.wikipedia.org/wiki/Logical_consequence#Syntactic_consequence
A formula A is a **syntactic consequence** within some formal system $\mathcal{FS}$ of a set Γ of formulas if there is a formal proof in $\mathcal{FS}$ of A from the set Γ: $\Gamma \vdash_{\mathcal{FS}} A$
Translation to MTT notational conventions: $\Gamma \vdash_{\mathcal{FS}} A \equiv (\exists\Gamma \subset \text{FS} (\Gamma \vdash \text{A}))$

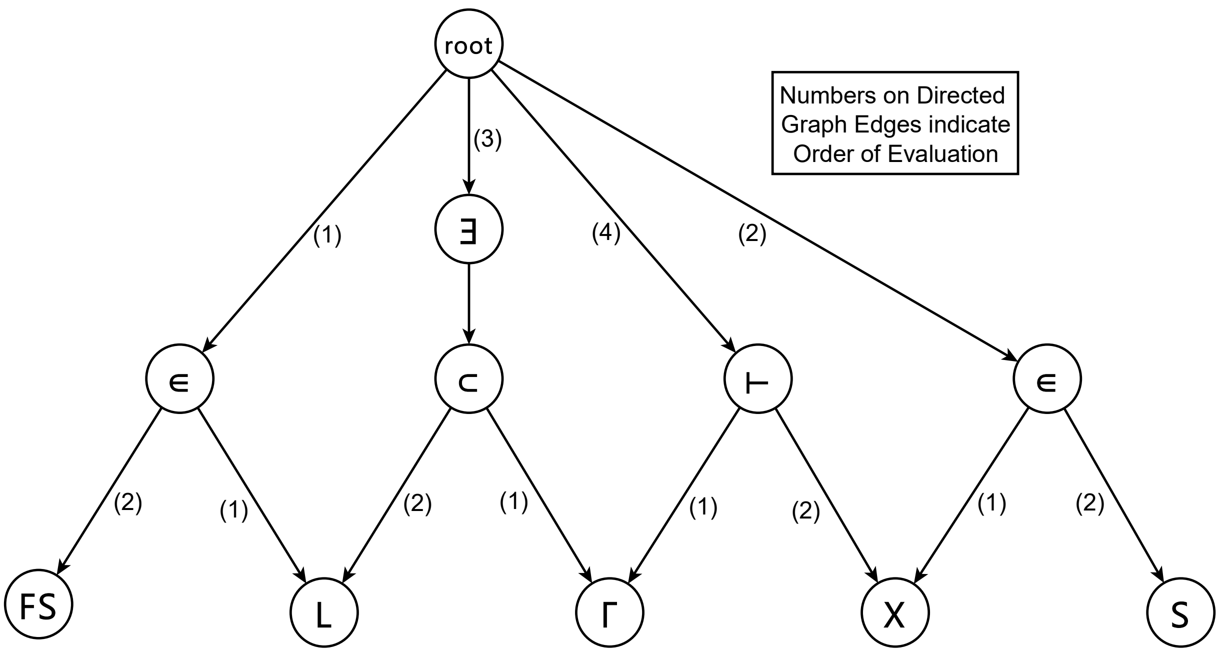
First Order Predicate Logic Syntax used the the basis for the Minimal Type Theory Language:

```
sentence
        : atomic_sentence
        | sentence IMPLIES sentence
        | sentence IFF sentence
        | sentence AND sentence
        | sentence OR sentence
        | sentence PROVES sentence          // enhancement
        | quantifier IDENTIFIER sentence   // MTT syntax is different
        | '~' sentence %prec NOT
        | '(' sentence ')'
        ;

atomic_sentence
        : IDENTIFIER '(' term_list ')'   // ATOMIC PREDICATE
        | IDENTIFIER                     // SENTENTIAL VARIABLE (enhancement)
        ;

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

term_list
        : term_list ',' term
        | term
        ;

quantifier
        : THERE_EXISTS
        | FOR_ALL
        ;
```

Minimal Type Theory augments the above syntax in two key ways:
(a) Adding the Assign Alias Operator: "@"
(b) Requiring every variable to be associated with a specific type.

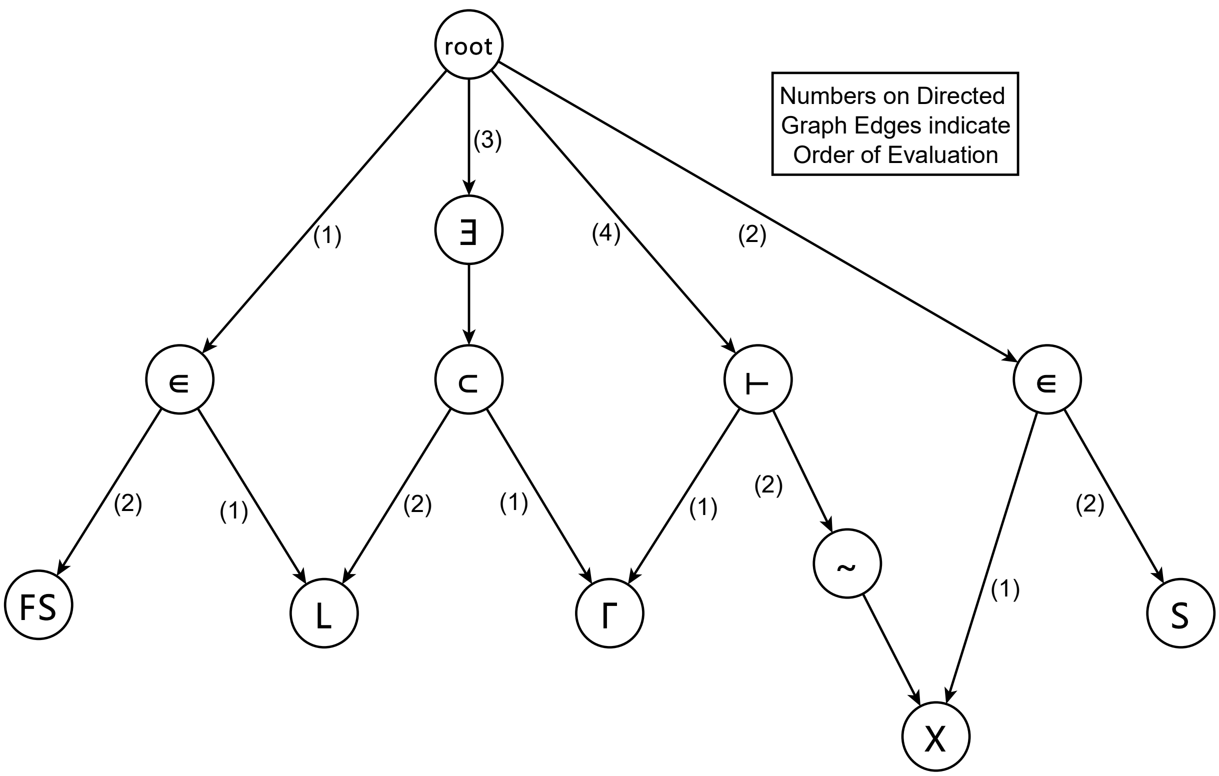
Provable(L, X) @ L ∈ Formal_Systems, X ∈ Finite_Strings, ∃Γ ⊂ L (Γ ⊢ X)

00 root (1)(4)(7)(10)
01 ∈ (2)(3)
02 L
03 Formal_Systems
04 ∈ (5)(6)
05 X
06 Finite_Strings
07 ∃ (8)
08 ⊂ (9)(2)
09 Γ
10 ⊢ (9)(5)



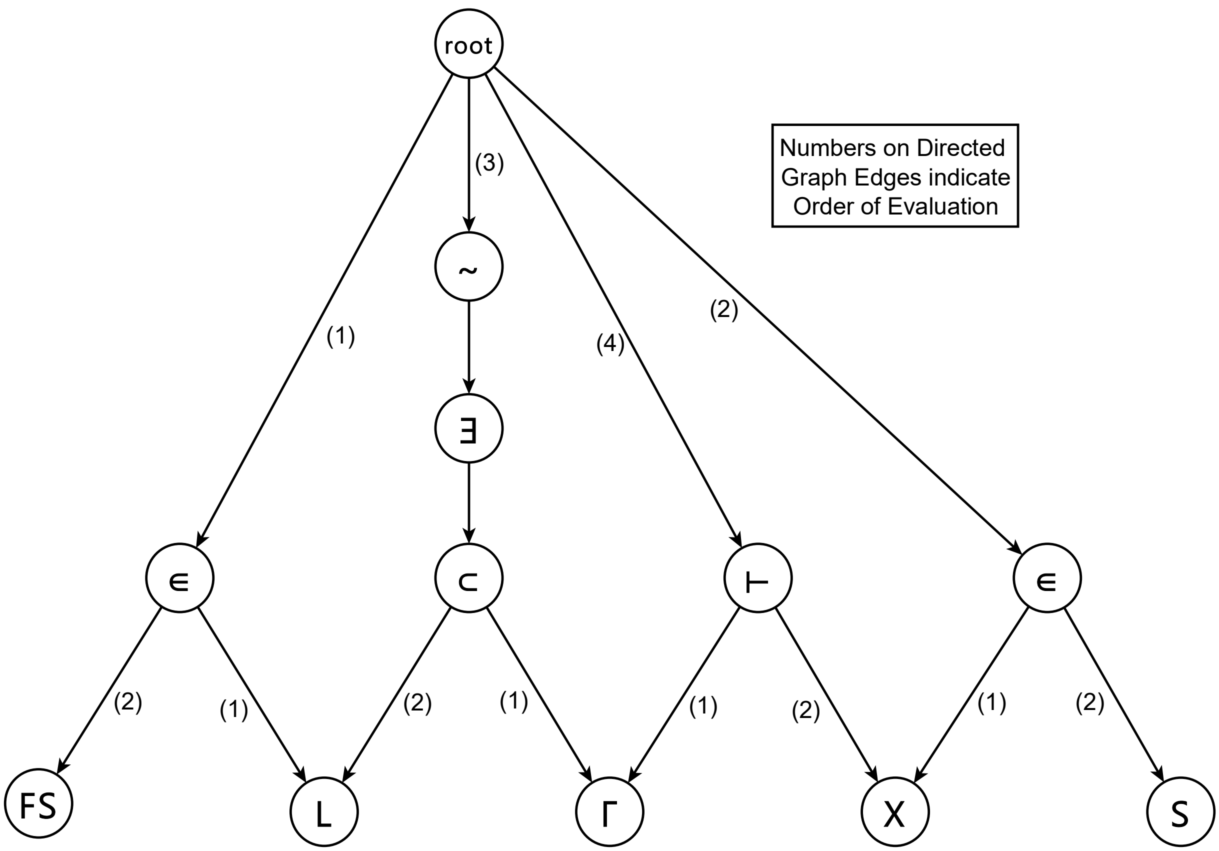
Refutable(L, X) @ L ∈ Formal_Systems, X ∈ Finite_Strings, ∃Γ ⊂ L (Γ ⊢ ~X)

00 root (1)(4)(7)(10)
01 ∈ (2)(3)
02 L
03 Formal_Systems
04 ∈ (5)(6)
05 X
06 Finite_Strings
07 ∃ (9)
08 ⊂ (9)(2)
09 Γ
10 ⊢ (9)(11)
11 ~ (5)



~Provable(L, X) @ L ∈ Formal_Systems, X ∈ Finite_Strings, ~∃Γ ⊂ L (Γ ⊢ X)

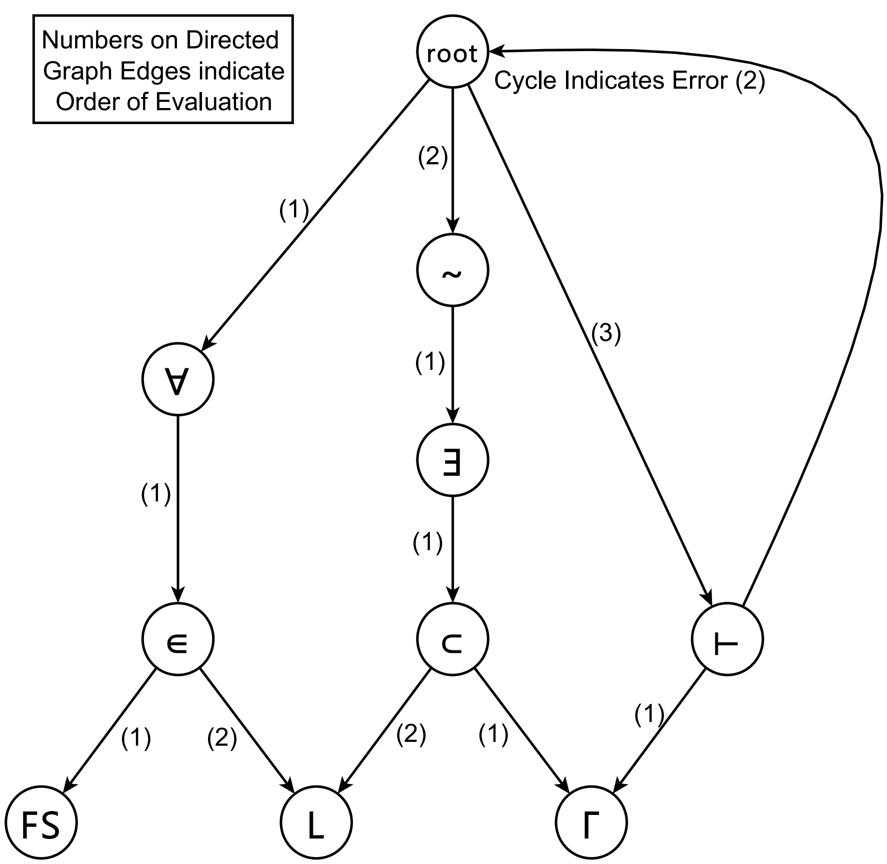
00 root (1)(4)(7)(11)
01 ∈ (2)(3)
02 L
03 Formal_Systems
04 ∈ (5)(6)
05 X
06 Finite_Strings
07 ~ (8)
08 ∃ (9)
09 ⊂ (10)(2)
10 Γ
11 ⊢ (10)(5)



G @ ∀L ∈ Formal_Systems, ~∃Γ ⊂ L (Γ ⊢ G)

"@" means the LHS is assigned as an alias for the RHS .
There is no referencing / dereferencing needed, G is one and the same thing as the expression that refers to G. (Unlike Tarksi naming) G is not referring to its name, G is referring to itself.

00 root (1)(5)(9) // G is an alias for this node
01 ∀ (2)
02 ∈ (3)(4)
03 L
04 Formal Systems
05 ~ (6)
06 ∃ (7)
07 ⊂ (8)(3)
08 Γ
09 ⊢ (8)(0) // cycle indicates infinite evaluation loop error



In the case of Pathological Self-Reference (PSR) the second argument to the ⊢ predicate forms and infinite loop instead of ever reaching its expected sentential variable. This prevents the evaluation of the expression from ever completing.

Gödel's Proof (Revised Edition) 2001
Nagel, Newman, and Hofstadter page 97
(G) ~(∃x) Dem (x, Sub(n, 17, n))

completing the substitution
(G) ~(∃x) Dem (x, G)

converting to common notation
(G) ~(∃x)(x ⊢ G)

Example of Provable(L, R)

WFF of L
(1) P // premise
(2) P → Q // axiom
(3) Q → R // axiom

Proof (using finite string rewrite rules)
Logical_Inference("P", "P → Q") ∴ "Q"
Logical_Inference("Q", "Q → R") ∴ "R"
∴ Provable("R")