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 $FS$ of a set $\Gamma$ of formulas if there is a formal proof in $FS$ of $A$ from the set $\Gamma$: $\Gamma \vdash_{FS} A$
Translation to MTT notational conventions: $\Gamma \vdash_{FS} A \equiv (\exists \Gamma \in FS (\Gamma \vdash A))$

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) \iff L \in Formal\_Systems, X \in Finite\_Strings, \exists \Gamma \subseteq L (\Gamma \vdash X)

Numbers on Directed Graph Edges indicate Order of Evaluation
Refutable(L, X) @ L ∈ Formal_Systems, X ∈ Finite_Strings, \( \exists \Gamma \subset L (\Gamma \vdash \neg X) \)

00 root (1)(4)(7)(10)
01 \( \in \) (2)(3)
02 L
03 Formal_Systems
04 \( \in \) (5)(6)
05 X
06 Finite_Strings
07 \( \exists \) (9)
08 \( \subset \) (9)(2)
09 \( \Gamma \)
10 \( \vdash \) (9)(11)
11 \( \neg \) (5)
\neg \text{Provable}(L, X) \iff L \in \text{Formal\_Systems}, X \in \text{Finite\_Strings}, \neg \exists \Gamma \subset L (\Gamma \vdash X)
G @ ∀L ∈ Formal\_Systems, \sim ∃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.
completing the substitution
(G) ¬ (∃x) Dem (x, Sub(n, 17, n) )

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")