Minimal Type Theory (MTT)

Minimal type theory (MTT) is based on type theory in that it is agnostic about predicate logic order and expressly disallows the evaluation of incompatible types. It is called minimal because it has the fewest possible number of fundamental types, and has all of its syntax is expressed entirely as directed paths between nodes of a directed acyclic graph (DAG).

Rudolf Carnap formalized the semantic meaning of otherwise meaningless propositional variables in his (1952) Meaning Postulates. The axiom: **Bachelor(x)** ↔ **~Married(x)** formalizes the semantic meaning of one aspect of these two English words using FOPL.

Minimal type theory applies meaning postulates to formalize (in a knowledge ontology) the semantic meaning of variables representing formal language abstractions. Sentential variables are simply the DAG node number of the root node of an MTT sentence, allowing a sentence to easily refer to itself.

MTT is a form of type theory that "borrows" its syntax from first order predicate logic (FOPL). Relations of MTT (corresponding to FOPL predicates) are connected to their terms by outward directed paths from the Relation to its terms. Every element of MTT corresponding to an element of FOPL { Predicate, Term, Logical-Symbol } has its own a directed acyclic graph node.

MTT has only two basic types (units of sub-atomic semantic compositionality):
(1) Relations // Predicates of some finite order Predicate Logic
(2) Non-Relations // Nodes having no outward directed paths

Syntactic-Logical-Consequence(Γ, x) Is a binary predicate within FOPL syntax defined as:
A formula ***x*** is a **syntactic consequence** within some formal system ***L*** of a set **Γ** of formulas if there is a formal proof in ***L*** of ***x*** from the set **Γ**. // **short-hand: (Γ ⊢x)**

Formalism (philosophy of mathematics)
Every formal system referenced in this paper is analyzed entirely within the foundational premise that all formal systems are comprised entirely of finite strings representing WFF that specify finite string rewrite rules. **Example: "P ∧ (P → Q)" can be rewritten as "Q".** We use finite strings as our basis only to avoid the presupposition that a finite string represents a WFF.

Types must be expressly stated in Minimal Type Theory
L ∈ Formal Systems, ∃x ∈ L, ∃Γ ⊂ L | (Γ ⊢ x)
x ∈ L --- x is one element of the set of WFF of L.
Γ ⊂ L --- Γ is a subset of the WFF of L.
Γ ⊢ x --- x is the syntactic logical consequence of Γ.

When-so-ever an expression requires the insertion of a cycle in the MTT (otherwise acyclic) directed graph this cycle indicates pathological self-reference (PSR). Pathological self-reference causes the evaluation of an expression to form an infinite loop.

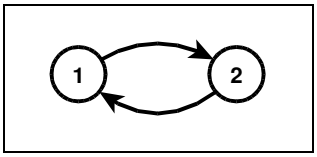
Seven is greater than five. "Greater-Than(Seven, Five)"
(1) Greater-Than --->(2)(3) // binary tree, thus ***not an evaluation infinite loop***
(2) Seven
(3) Five

Liar Paradox formalized as Predicate Logic syntax and semantics formalized as MTT.

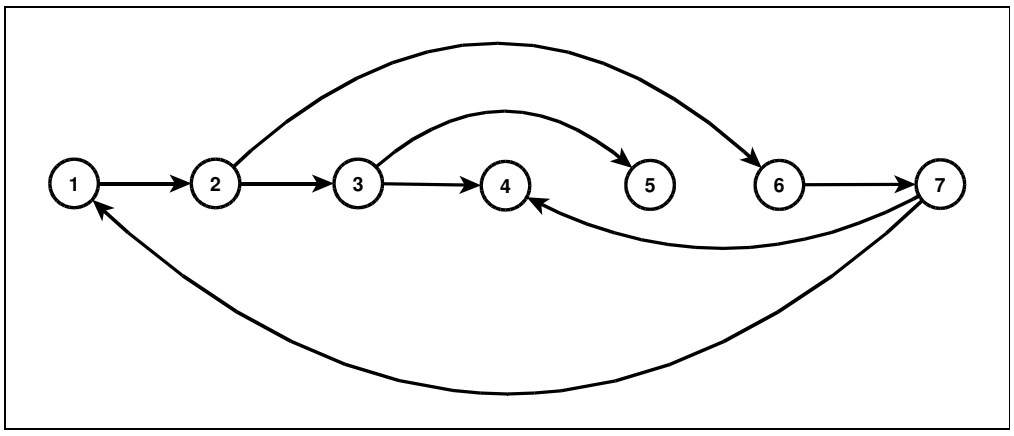
// Defining Tarski's (1933) Formal correctness of True: ∀x True(x) ↔ φ(x)
∀L ∈ Formal Systems ∀x ∈ finite strings, ∃Γ ⊂ L ((Γ ⊢ x) ↔ True(x))

The above formula template defines the semantic meaning of Boolean-True within the metalanguage of MTT. This formula template is applied by establishing a bijective mathematical mapping between the MTT formula template and the corresponding syntax of each individual formal system's object language, thus defining True(x) within each formal system.

"This sentence is not true" FOPL: **x = "~True(x)"**
Minimal Type Theory (MTT) Directed Acyclic Graph (DAG) of Liar Paradox
(1) Negation —> (2) // x is an alias for this node
(2) True —> (1) // cycle indicates error: evaluation infinite loop



"This sentence is not true" FOPL: **x = "~∃Γ ⊂ L | (Γ ⊢ x)"**
Minimal Type Theory (MTT) Directed Acyclic Graph (DAG) of Liar Paradox
(1) Negation —> (2) // x is an alias for this node
(2) There-Exists —> (3)(6)
(3) Subset of —> (4)(5)
(4) Γ
(5) L
(6) Such-That —> (7)
(7) Syntactic-Logical-Consequence —> (4)(1) // cycle indicates error: evaluation infinite loop



The above formalism proves that the Liar Paradox is not a truth bearer because it gets stuck in an infinite evaluation loop. Because it is not a truth bearer, the Liar Paradox is not a semantically correct logical sentence. This same analysis equally applies to the (Truth Teller Paradox) Negation of the Liar Paradox: **x = "∃Γ ⊂ L | (Γ ⊢ x)".**

Formalizing (the semantics of) Gödel's (1931) incompleteness theorem using MTT

// **Terminology comes from Nagel, Newman, and Hofstadter pages 96-97**
// **G = Gödel_Number("~(∃x) | (x ⊢ G)")** // **Where G, x are Gödel numbers of PM**
G = "~∃x ⊂ PM | (x ⊢ G)"

Minimal Type Theory (MTT) Directed Acyclic Graph (DAG) of Incompleteness Theorem
(1) Negation —> (2) // G is an alias for this node
(2) There-Exists —> (3)(6)
(3) Subset of —> (4)(5)
(4) x
(5) PM
(6) Such-That —> (7)
(7) Syntactic-Logical-Consequence —> (4)(1) // cycle indicates error: evaluation infinite loop

When we formalize Kurt Gödel's (1931) first incompleteness theorem using MTT we see that exactly like the Liar Paradox the expression named G above is not a logical sentence of any formal system because its evaluation gets stuck in an loop.

https://plato.stanford.edu/entries/goedel-incompleteness/
The first incompleteness theorem states that in any consistent formal system F within which a certain amount of arithmetic can be carried out, there are statements of the language of F which can neither be proved nor disproved in F.

As the above Minimal Type Theory formalization of Kurt Gödel's (1931) first incompleteness theorem unequivocally shows the conclusion of this theorem is incorrect. Kurt Gödel really only proved that some logical expressions that are ***not*** statements of the language of F cannot be proved in F.

Kurt Gödel made this mistake because (at the time) there was no way to formalize the semantic meaning of logical expressions, thus the correctness of this expression was only validated as a syntactically WFF.

Concrete example for the specific Formal System of the Peano Arithmetic

Less-Than < operator defined using Peano Arithmetic
$a < b \leftrightarrow \exists c \in \mathbb{N} | ((a \neq b) \wedge (b = a + c))$

x = "7 > 3" ∈ finite strings, ∃Γ ⊂ Peano Arithmetic ((Γ ⊢ x) ↔ True(x))
x = "7 < 3" ∈ finite strings, ~∃Γ ⊂ Peano Arithmetic ((Γ ⊢ x) ↔ ~True(x))
x = "7 < 3" ∈ finite strings, ∃Γ ⊂ Peano Arithmetic ((Γ ⊢ ~x) ↔ False(x))
x = "jx2dr" ∈ finite strings, ~∃Γ ⊂ Peano Arithmetic ((Γ ⊢ x) ↔ ~True(x))
x = "jx2dr" ∈ finite strings, ~∃Γ ⊂ Peano Arithmetic ((Γ ⊢ ~x) ↔ ~False(x))

Since we were able to complete Alfred Tarski's (1933) formal correctness of True(x) formula, (correctly refuting his (1936) undefinability theorem) when-so-ever the predicate Truth-Bearer(x) evaluates to true, this simultaneously validates WFF(x) and Semantically-Correct(x) for any finite string x.