The Notion of Truth in Natural and Formal Languages

The purpose of this paper is to complete the RHS of Tarski's famous formula: ∀x True(x) ↔ φ(x)

For any natural (human) or formal (mathematical) language L we know that an expression X of language L is true if and only if there are expressions Γ of language L that connect X to known facts. By extending the notion of a Well Formed Formula to include syntactically formalized rules for rejecting semantically incorrect expressions we recognize and reject expressions that evaluate to neither True nor False.

> Weisstein, Eric W. "**Axiom**." From MathWorld--A Wolfram Web Resource.
> http://mathworld.wolfram.com/Axiom.html
> An axiom is a proposition regarded as self-evidently true without proof.

Axioms are really nothing more than a set of expressions of language that have been assigned the semantic property of True. Axioms form the ultimate foundation of Truth-conditional semantics.

> https://www.britannica.com/topic/tautology
> **Tautology**, in logic, a statement so framed that it cannot be denied without inconsistency. Thus, "All humans are mammals" is held to assert with regard to anything whatsoever that either it is a human or it is not a mammal. But that universal "truth" follows not from any facts noted about real humans but only from the actual use of human and mammal and is thus purely a matter of definition.

The natural language equivalent to an axiom in formal language is a {known fact}. Some expressions of natural language are simply tautologically defined to be True.

Example: "a cat is an animal".
Formalized as: (cat ∈ animals) or (cat ◁ animal)
where ◁ is the [is_a_type_of] operator adapted from UML Inheritance relation.
The only reason that we know that "a cat is an animal" is that it is defined to be True.

> **Meaning Postulates** (1952) by Rudolf Carnap formalized natural language semantics:
> (x) Bachelor(x) → ~Married(x)

All that we know about the otherwise totally meaningless finite string predicate "Bachelor" is that when this predicate is satisfied then the otherwise totally meaningless finite string predicate "Married" is not satisfied. We did not even limit x, so we could say that if {a pile of bricks} is a Bachelor then {a pile of bricks} is not married.

A further refinement would be this:∀x ∈ Human_Being Bachelor(x) → ~Married(x)

Now the above two predicates have been constrained to a type, and all that we know about the otherwise totally meaningless finite string "Human_Being" is that it is the argument type to the otherwise totally meaningless finite string predicates: "Bachelor" and "Married". The above example shows how to formalize natural language axioms.

Generalizing the notion of a (known fact) to formal language we define an axiom as any expression of (formal or natural) language that has been defined to have the semantic property of True.
This concept of an axiom provides the ultimate foundational basis of all conceptual Truth.

> **Validity and Soundness** https://www.iep.utm.edu/val-snd/
> A deductive argument is said to be valid if and only if it takes a form that makes it impossible for the premises to be true and the conclusion nevertheless to be false. Otherwise, a deductive argument is said to be invalid.
>
> A deductive argument is sound if and only if it is both valid, and all of its premises are actually true. Otherwise, a deductive argument is unsound.

A sound argument necessitates the Truth of its conclusion. If we define a symbolic logic predicate to formalize the distinction between a valid deductive argument and a sound deductive argument where represents the premises Γ and C stands for the conclusion then:

Provable(Γ, C) is simply a valid deductive argument where the premises Γ may or may not be true.
Γ ⊢ C is merely infix notation for this predicate Provable(**Γ**, C).

> Introduction to Mathematical logic Sixth edition Elliott Mendelson
> 1.4 An Axiom System for the Propositional Calculus page 28
> A wf C is said to be a consequence in S of a set Γ of wfs if and only if there is a sequence B1, ..., Bk of wfs such that C is Bk and, for each i, either Bi is an axiom or Bi is in Γ, or Bi is a direct consequence by some rule of inference of some of the preceding wfs in the sequence. Such a sequence is called a proof (or deduction) of C from Γ. The members of Γ are called the hypotheses or premisses of the proof.
> **We use Γ ⊢ C as an abbreviation for "C is a consequence of Γ"...**
>
> If Γ is the empty set ∅, then ∅ ⊢ C if and only if C is a theorem. It is customary to omit the sign "∅" and simply write ⊢ C. Thus, ⊢ C is another way of asserting that C is a theorem.

Mendelson's Γ ⊢ C specifies valid deductive inference and ⊢ C specifies sound deductive inference. As we already know sound deduction derives a true conclusion, thus ∀x True(x) ↔ ⊢ x would complete the Tarski (formal correctness) Truth predicate: ∀x True(x) ↔ φ(x).

Since theorems of formal language are nothing more than sound deductive inference and we already have the notion of a natural language axiom, we can specify a natural language theorem as any expression of natural language that can be totally verified as true entirely on the basis of the meaning of its words.

The key understanding that the above analysis provides
(1) Axioms (known facts) are the ultimate foundational basis of conceptual Truth.
(2) Conceptual truth is essentially nothing more than theorems of formal or natural language.
(3) Therefore True(x) is always deductively Provable(x).

Sentence (mathematical logic) https://en.wikipedia.org/wiki/Sentence_(mathematical_logic)
In mathematical logic, a sentence of a predicate logic is a Boolean-valued well-formed formula with no free variables. A sentence can be viewed as expressing a proposition, something that must be true or false. The restriction of having no free variables is needed to make sure that sentences can have concrete, fixed truth values: As the free variables of a (general) formula can range over several values, the truth value of such a formula may vary.

Apparently in all of the years prior to my original (2016) paper no one ever separated the analysis of propositions into their atomic units of semantic compositionality:
(a) Assertion (Boolean) // What the expression of language claims to be True
(b) Satisfied (Boolean) // Whether or not the expression of language is True

Previously these two properties were conflated together as the Satisfiability property making the analytical system insufficiently granular to detect the semantic error.

What has previously been understood to be True outside the system and not Provable within the system is actually a contradiction between two different semantic properties of the same Proposition. So True, but not Provable is really nothing more than a contradiction between an expression's Assertion property and its Satisfiability property.

In the case of semantic error of Pathological Self-Reference(Olcott 2004) the Truth of the expression's Assertion property does not make the entire expression True because the Truth values of the Assertion property and the Satisfiability property of the expression are mutually exclusive.

Because two different Boolean properties of an expression of language have mutually exclusive values this makes it impossible to evaluate this expression as either True or False. In other words every self-contradictory statement of language is logically equivalent to deductive inference on the basis of contradictory premises, thus unsound.

When we formalize expressions of language such as the Liar Paradox using this universal truth predicate: $\forall x$ True(x) $\leftrightarrow \vdash x$ we can finally understand its semantic error.

"This sentence is not true." --- Formalized as: LP $\leftrightarrow \sim\vdash$LP
English: A sentence is materially equivalent to a statement of its own unprovability.

If $\vdash$LP this contradicts its assertion: ($\sim\vdash$LP) $\therefore$ ~True(LP)
If we could prove that LP is provable, then its assertion that it is not provable is contradicted.

Next we see if LP is refutable by trying to prove its negation.
If $\vdash\sim$LP this contradicts its assertion: ~($\sim\vdash$LP) $\therefore$ ~False(LP)
If we could prove that ~LP is provable, then its assertion that it is not not provable (AKA provable) is contradicted.

$\forall x$ ((~True(x) $\wedge$ ~False(x)) $\leftrightarrow$ ~Logic_Sentence(x))
$\therefore$ ~Logic_Sentence(LP)

Just like the essence of the 1931 Incompleteness Theorem:

> 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.
> (Raatikainen, Panu: Fall 2018)

The Liar Paradox is neither provable nor refutable.

Because of Pathological self-reference (Olcott 2004) LP's Assertion and its Satisfied properties have mutually exclusive Boolean values making LP neither True nor False thus semantically incorrect.

If there was a sequence of WFF that proves LP it would be self-contradictory because it proved that its own proof must fail therefore making its proof succeed. Therefore LP's assertion is True.

Even though LP's assertion is True, because the LP expression cannot possibly be satisfied the LP expression cannot possibly be True. It is also impossible for the LP expression to be False because its assertion is True. Therefore LP's Satisfiable property is False.

So we have the paradoxical case where the assertion of a Proposition is True, yet this does not make the Proposition itself True. Since LP cannot possibly be either True or False it is therefore a semantically incorrect Proposition because all Propositions must be True or False.

> ON FORMALLY UNDECIDABLE PROPOSITIONS
> OF PRINCIPIA MATHEMATICA AND RELATED SYSTEMS I
> by Kurt Godel Vienna
> The analogy between this result and Richard's antinomy leaps to the eye; there is also a close relationship with the "liar" antinomy,14
>
> 14 Every epistemological antinomy can likewise be used for a similar undecidability proof.

Since Kurt Gödel said that the Liar Paradox "can ... be used for a similar undecidability proof." The semantic error of the Liar Paradox equally applies to the 1931 Incompleteness Theorem.

A formula precisely analogous to the Liar Paradox specifying Provability instead of Truth
"This sentence is not provable".
$\exists F \in$ Formal_Systems $(\exists G$ Language$(G, F) \wedge (G \leftrightarrow \sim(F \vdash G)))$

Because it is logically equivalent to the formalized Liar Paradox the above simplification of the formalized essence of Gödel's 1931 Incompleteness Theorem can be understood to be unsatisfiable thus refuting the conclusion of the theorem itself.

When we simply look at Incompleteness a little bit more clearly the famous theorem's famous conclusion falls apart.

Minimal Type Theory (formal specification)

MTT was created by augmenting the syntax of First Order Logic (FOL) to specify Higher Order Logic (HOL) expressions with types. FOL quantifiers can specify an optional type. FOL is a subset of MTT. The ASSIGN_ALIAS operator := enables MTT expressions to be chained together to form HOL expressions. It also provides the means for an expression of language to refer directly to itself (not merely to it name).

"This sentence is not true." is formalized as LP := ~True(LP)

Because := specifies macro substitution we can see that the formalized Liar Paradox is not a WFF of any formal system because it specifies an infinite string.

```
%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)
%%

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
     ;
```