Formalizing Declarative Sentences: ∀x True(x) ↔ φ(x)

This paper derives mathematical mapping theory of truth on the basis of the Alfred Tarski semantic conception of truth. It does this by reverse engineering previously unknown details of the Tarski theory via a specific concrete example. This example fully elaborates the semantic meaning of φ(x) in the following Tarski formula: For all x, True(x) if and only if φ(x).

We begin with the Tarski's foundation: If the language under discussion (the object language) is L, then the definition should be given in another language known as the metalanguage, call it M.

[This sentence has five words]
// Leaf node lexemes are shown in lowercase.
(S (NP this sentence)
  (VP has
    (NP five words)))

To simplify the notational conventions (both functions return integer)
Count-Words(s) ↔ Natural-Language-Parse(s).Count-Leaf-Nodes()

Three mandatory steps for correctly interpreting the semantic meaning of declarative sentences
// Translate natural language s into metalanguage x such that:
(1) (s ∈ L) ↦ (x ∈ M)
  (s ∈ L) [This sentence has five words]
  (x ∈ M) Integer-Equality(Count-Words(s), 5)

// Verify that s is a valid declarative sentence type using two predicates
(2) WFF(x) ↔
  x ↦ Binary-Relation & Compatible-Types(Binary-Relation, arg1, arg2)
  x ↦ Binary-Relation(Integer-Equality) & Compatible-Types(Integer-Equality, Integer, Integer)

// Determine whether or not declarative sentence s is True
(3) Binary-Relation-Definitely-Exists(x) ↔
  x ↦ Binary-Relation(arg1, arg2)
  x ↦ Integer-Equality(Count-Words(s), 5)

It turns out that truth is simply the fact that a (binary predicate) relationship definitely exists between two things. The above sentence makes the assertion that it has five words. When we count these words we find that this count has an integer equality relationship to five.

On the basis of the above we can precisely specify the atomic semantic meaning of the right hand side of this Tarski formula: ∀x True(x) ↔ φ(x):
WFF(x) & x ↦ Binary-Relation-Definitely-Exists(arg1, arg2)

In other words: φ(x) ranges over relations between pairs of compatible types where the relation definitely exists, thus syntactically defining the otherwise meaningless expression True(x).
Example: (s ∈ L) [Pete owns a car] ↔ (x ∈ M) Ownership(Pete, car) ↔ True(x)

Sentences that have no Correct Declarative Interpretation: x ↔ ~WFF(x)

[what time is it]	no binary relation (not declarative sentence)
[Colorless green ideas sleep furiously]	binary relation with incompatible types
[this sentence is false]	x ⊨ ~True(x)
[this sentence cannot be proven true]	~(x ⊨ True(x))
[This sentence is not true]	x ⊨ ~True(x)

(1) (s ∈ L) ↦ (x ∈ M)
(s ∈ L) [this sentence is true] // s ↔ "This sentence is true."
(x ∈ M) x ⊨ True(x) // x ↔ x ⊨ True(x)

To determine True(x) we plug the defined x into the right-hand side of: True(x) ↔ φ(x).
The first step of φ(x) is WFF(x) and is defined having two sub-steps:
(a) Is a Binary-Relation specified? // YES
x ↔ Logical-Entailment(x, True(x)) ↦ Binary-Relation

(b) Does the specified Binary-Relation have compatible types? // NO
Logical-Entailment(x, True(x))

Since the second argument to Logical-Entailment specifies an infinitely recursive structure Logical-Entailment(x, True(x)) never completes, thus Logical-Entailment has incompatible types.