Uncomputability and Logic

Gödel's Incompleteness Theorems

Zoé Christoff, ILLC

SGSLPS Annual Meeting
April 5, 2012

Plan

Mathematical context

Recent discoveries:

- Cantor's theorem: there are bigger infinities than others. (1891)
- Russell's Paradox (1902)

Russell's Paradox

If any collection of objects having a certain property is to be considered as a set, then, we can define the set:

$$\{x \mid x \notin x\}$$

Does this set belong to itself? If yes, then it doesn't. If not, then it does.
Set theory (without any further assumption about what a set cannot be) is inconsistent.

The Liar Paradox

This sentence is false.

Hilbert's Program

After the crisis due to Russell's paradox, David Hilbert required a PROOF that mathematics is consistent.
He also wanted to keep the possibilities of "wild" infinity as part of mathematics : "No one shall expel us from the paradise that Cantor has created for us."
Ideally, what is wanted is a limited formal system which could generate every mathematical truth and no falsity (even the ones about infinities) and prove them all (and nothing else), using a limited number of rules and axioms.
Using only finite method, a proof of consistency of infinite mathematics !

Kurt Gödel (1906-1978)

- 1929: Gödel's doctoral dissertation, written when he was 23, established the completeness theorem for the first-order logic.
- 1931: "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme" (On formally undecidable propositions of Principia Mathematica and related systems): establishes both incompleteness theorems.

First Incompleteness Theorem

- **There is no complete consistent axiomatizable theory of arithmetic.**

Gödel's incompleteness theorems tell us about the limits of axiomatized formal theories of arithmetic.

Theory

Δ : set of sentences of our language.

- A **theorem** D of Δ is a sentence proved by Δ, a sentence deducible from Δ. We note $\Delta \vdash D$ and if Δ is empty (logical theorem), $\vdash D$.
- A **theory** T is a set of sentences (in a given formal language) which is closed under deduction (contains all the sentences of its language that are provable from it). The theorems of a theory T are just the sentences in T, and $T \vdash D$ and $D \in T$ are two ways of writing the same thing.
- The set of all theorems of a system is a theory.

Axiomatizability

- If there is a recursive set Σ of sentences such that T consists of all and only the sentences provable from Σ, we say T is **axiomatizable**.

Consistency

- A set of sentences Δ is **consistent** iff no contradiction can be proven from it.
- A theory T is **consistent** iff it contains no contradiction.

Not this completeness notion...

Completeness in regard to its semantics, converse of soundness:

- For any formula ϕ in the language and for any set Δ of formulas of the language:

$$\text{if } \Delta \vDash \phi, \text{ then } \Delta \vdash \phi$$

The first proof of completeness for first-order logic was already given by Gödel in 1929 (Gödel's Completeness Theorem).

but this one: negation-completeness

- A set of sentences Δ is (negation-)**complete** iff for every sentence B of its language:

$$\Delta \vdash B \text{ or } \Delta \vdash \neg B$$

- A theory T is (negation-)**complete** iff for every sentence B of its language:

$$B \in T \text{ or } \neg B \in T$$

Strategy of proof

Assume that T is any consistent axiomatizable theory of artithmetic and show that it is not negation-complete, that is, that there is at least one sentence such that the theory cannot prove it nor disprove it (prove its negation).

Strategy of proof

1. Show that our formal language can "talk about" its own syntax (sentences and proofs), through coding. Bring together logic and computability (recursive functions) in this first direction.
2. Show that we can "talk about" recursive functions using formulas IN our language of arithmetic. (definability and representability of recursive functions in the language).
3. Combining both preceding results: show that a formal system can "talk about" itself: we can talk about (sentences and proofs of) a formal system of arithmetic within the formal system of arithmetic itself. We can bring the metalanguage into the language of arithmetic.

Strategy of proof

4. Assuming we have a consistent theory of arithmetic, show that it cannot be defined by any formula of the language. (If it could, then we would have a sentence G of the language of arithmetic such that it is provable that: G is true if and only if G is not a theorem and the theory would be inconsistent.)
5. Using the previously established results (if an axiomatizable theory of arithmetic is complete, then it is recursive and therefore definable and if a theory of arithmetic is consistent, then it is not definable), deduce the first incompleteness theorem.

FOL: Symbols

Logical symbols:

- variables symbols: x, y, z, ...
- connective symbols: $\neg$, &, $\vee$, $\rightarrow$, $\leftrightarrow$
- quantifiers: $\forall$, $\exists$
- identity predicate symbol: $=$

Non-logical symbols (language):

- constants (individual symbols) : a, b, c, ...
- predicates (relation symbols): P, Q, R, ...
- function symbols: f, g, h, ...

FOL: Interpretation

An interpretation M for a language L consists of two components:

- **domain** $|M|$: the nonempty set of things M interprets the language to be talking about.
- **denotation** for every non-logical symbol:
  - constant symbol: individual in $|M|$.
  - n-place (nonlogical) predicate: n-place relation (set of n-tuples) on $|M|$.
  - n-place function symbol f: an n-argument function from $|M|$ to $|M|$.

The language $L*$ and its standard interpretation $N*$

- constant symbol: $\mathbf{0}$

  denotes the number 0.
- 2-place predicate symbol: $<$

  denotes the set of pairs of elements of $\mathbb{N}$ such that the first is strictly less than the second.
- function symbols:
  - one-place function symbol: s
    denotes the successor function (takes each number to the next larger one).
  - two-places function symbols: $+$, $\cdot$
    denote the usual addition and multiplication functions.

Abbreviation rule: we abbreviate by $\mathbf{n}$ the composition of n times s on $\mathbf{0}$: $s(s(....\mathbf{0})$. So the numeral $\mathbf{n}$ denotes the number n.

Syntax: inductive definition of terms of $L*$

- atomic:
  - variables: x, y, z, ...
  - constants: $\mathbf{n}$ (for any $n \in \mathbb{N}$)
- complex:
  - any expression $f(t_1, ..., t_n)$ where f is any n-place function applied to n terms

So any term can be built up from atomic terms in a sequence of finitely many steps (formation sequence) by applying function symbols to simpler terms. (And everything that can be built up in this way is a term.)

Syntax: inductive definition of (well-formed) formulas of $L*$

- atomic:

  for t_1 and t_2 any terms of our language:

  - $t_1 = t_2$
  - $t_1 < t_2$
- complex:

  for ϕ and ψ any (well-formed) formulas of our language:

  - $\neg\phi$, $\phi \,\&\, \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$, $\exists x\phi$, $\forall x\phi$.

So any formula can be built up from atomic formulas in a sequence of finitely many steps (formation sequence) by applying connectors and quantifiers to simpler formulas. (And everything that can be built up in this way is a formula.)

Semantics: definition of (arithmetical) truth

- $N* \vDash t_1 = t_2$ iff t_1 and t_2 denote the same number.
- $N* \vDash t_1 < t_2$ iff the number denoted by t_1 is strictly smaller than the one denoted by t_2.
- $N* \vDash \neg\phi$ iff $N* \nvDash \phi$
- $N* \vDash \phi \& \psi$ iff $N* \vDash \phi$ and $N* \vDash \psi$
- $N* \vDash \phi \vee \psi$ iff $N* \vDash \phi$ or $N* \vDash \psi$
- $N* \vDash \phi \rightarrow \psi$ iff $N* \nvDash \phi$ or $N* \vDash \psi$
- $N* \vDash \phi \leftrightarrow \psi$ iff $N* \vDash \phi \rightarrow \psi$ and $N* \vDash \psi \rightarrow \phi$
- $N* \vDash \exists x \phi$ iff for some $n \in \mathbb{N}$, $N* \vDash \phi(n)$
- $N* \vDash \forall x \phi$ iff for all $n \in \mathbb{N}$, $N* \vDash \phi(n)$

Logical *vs* meta-logical notions

- Logical: negation, conjunction, quantification, ...
- Meta-logical: consequence, satisfiability, validity, proof ...

There are symbols for the logical notions in our formal language $L*$ (the object language).
Words like "consequence" or "proof" only appear in the mathematical English in which we talk about our formal language (metalanguage).

Examples of meta-logical (syntactic) notions

For D any sentence of the language and Δ any set of sentences of the language:

- A **proof/derivation** of D from a set Δ is a sequence of wff of the language such that the last formula is D and for each formula in the sequence, either it is a formula $\in \Delta$ or it is obtained from the preceding formulas in the sequence by the application of some of the given deduction rules of the system.
- D is **provable** from Δ iff there is a proof of D from Δ.
- D is **demonstrable** iff there is a proof of D from $\emptyset$.

Effective computability

A function is **effectively computable** iff there is a finite list of definite explicit rules (instructions), following which one could in principle compute its value for any given argument.
Examples: the **basic functions**:

- The zero function z (assigns value 0 to any argument)
- The successor function s (assigns the next bigger number to any argument)
- The identities functions: id_i^n (projection functions): $d_i^n(x_1, ..., x_i, ..., x_n) = x_i$

Effectively computable functions building operations

Composition: For a function f of m arguments and a function g of n arguments: $h = Cn[f, g_1, ..., g_m]$:

$$h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$$

Examples: Constant functions : for each n, the function $const_n$ can be obtained from the basic functions by finitely many (namely n) applications of the successor function s on 0.

Effectively computable functions building operations

Primitive recursion: a function h is said to be definable by (primitive) recursion from the functions f and g, $h = Pr[f, g]$, when both these equations hold:

$$h(x, 0) = f(x)$$
$$h(x, s(y)) = g(x, y, h(x, y))$$

Functions obtainable from the basic functions by composition and primitive recursion are called **primitive recursive**. All such functions are effectively computable.

Arithmetic operations as primitive recursive functions

Addition is repeated succession:

- Informally:

$$x + 0 = x,\ x + s(y) = s(x + y)$$

  We use the second equation to reduce the problem of computing x+y to that of computing x+z for smaller and smaller z, until we arrive at z=0, when the first equation tells us directly how to compute x+0.

- Formally: $sum = Pr[id^1_1, Cn[s, id^3_3]]$:

$$sum(x, 0) = id^1_1(x)$$
$$sum(x, s(y)) = s(sum(x, y)) = (Cn[s, id^3_3](x, y, sum(s, y))$$

Arithmetic operations as primitive recursive functions

Multiplication is repeated addition:

- Informally:

$$x \cdot 0 = 0$$
$$x \cdot s(y) = s(x \cdot y)$$

- Formally: $prod = Pr[z, Cn[sum, id_1^3 id_3^3]]$:

$$prod(x, 0) = z(x)$$
$$prod(x, s(y)) = sum(x, prod(x, y)) =$$
$$Cn[sum, id_1^3, id_3^3](x, y, prod(x, y))$$

This enables us to reduce the computation of a product to the computations of sums, which we already know how to compute.

Arithmetic operations as primitive recursive functions

Exponentiation is repeated multiplication

- Informally:

$$x \cdot 0 = 0$$
$$x \cdot s(y) = s(x \cdot y)$$

- Formally: $exp = Pr[Cn[s, z], Cn[prod, id_1^3 id_3^3]]$

$$exp(x, 0) = const_1(x) = s(z(x)) = Cn[s, z](x)$$
$$exp(x, s(y)) = prod(x, exp(x, y)) =$$
$$Cn[prod, id_1^3 id_3^3](x, y, exp(x, y))$$

This enables us to reduce the computation of an exponentiation to the computation of products, which we know how to compute.
And so on (super exponentiation is defined in terms of exponentiation, super-duper-exponentiation in terms of super exponentiation,...).

One more effectively computable functions-building operation

Minimization: Given a computable $n+1$-ary function f the minimization of f is the n-ary function h (we note Mn $[f]$) such that: $h(x_1, x_2, ..., x_n) =$

- the smallest y s.t. $f(x_1, x_2, ..., x_n, y) = 0$, if such a y exists
- undefined if no such y exists.

Recursive function: any function that can be constructed from the basic functions (of the recursive function theory) by the function-building operations of composition, primitive recursion and minimization.

Recursive functions are effectively computable

- All recursive functions are effectively computable. What about the converse?
- Church's Thesis: all effectively computable functions are recursive.
- "The interest of Church's Thesis derives largely from the fact that some particular functions of great interest in logic and mathematics are nonrecursive. If the thesis is correct, we can infer the practical advice that logicians and mathematicians would be wasting their time looking for a set of instructions to compute the function." (Boolos)

Recursive sets/relations

Recursive sets provide many more examples of (primitive) recursive functions.

- A set of (tuples of) natural numbers is **effectively decidable** iff there is an effective procedure that, applied to any (tuple of) natural number(s), gives in a finite amount of time the correct answer to the question whether it belongs to the set.
- **characteristic function** of a set: the function that assigns value 1 to numbers in the set and the value 0 to numbers not in the set.
- A (primitive) **recursive set** is a set whose characteristic function is (primitive) recursive.
- A set is effectively decidable iff its characteristic function is effectively computable.

Recursive sets/relations are decidable

- Since every recursive function is effectively computable, every recursive set is decidable.
- Church's (hypo)thesis (all effectively computable functions are recursive) implies that all effectively decidable sets are recursive.

More recursive relations

Processes for defining new (primitive) recursive relations when applied to (primitive) recursive relations:

- Substitution: $S(x_1, ..., x_n) \leftrightarrow R(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$
- Negation: $S(x_1, ..., x_n) \leftrightarrow \neg R(x_1, ..., x_n)$
- Conjunction: $S(x_1, ..., x_n) \leftrightarrow R1(x_1, ..., x_n) \& R2(x_1, ..., x_n)$
- Disjunction: $S(x_1, ..., x_n) \leftrightarrow R1(x_1, ..., x_n) \vee R2(x_1, ..., x_n)$

The negation is simply the complement, the conjunction the intersection, and the disjunction the union.

- Bounded universal quantification:
  $S(x_1, ..., x_n, u) \leftrightarrow \forall v < u, R(x_1, ..., x_n, v)$
- Bounded existential quantification:
  $S(x_1, ..., x_n, u) \leftrightarrow \exists v < u, R(x_1, ..., x_n, v)$

Arithmetization

We now connect the notions pertaining to computability with the ones pertaining to FOL in a first direction.

- We are going to show that we can express things about the syntax of our formal system of arithmetic in the formal system of arithmetic.
- Necessary preliminary: code expressions of our formal language by natural numbers.

Gödel numbering

An encoding function such that it assigns to each symbol of our formal language (as well as the parenthesis and comma) a unique number of $\mathbb{N}$,

- Once every symbol of the language is encoded by its Gödel number, any string of symbols can be represented as a sequence of natural numbers, which can itself be represented by a unique number.
- Result: any string of symbols of our language can be represented/translated by a unique number: terms, formulas, sentences, proofs.

Gödel numbering

One way to do it is to use prime factorization, so that for a string of symbols (such that g_n is the Gödel number of the nth*symbol*).
Example:

- A string of 4 symbols with code numbers g_1, g_2, g_3, g_4 will be encoded by :
- $2^{g_1} \cdot 3^{g_2} \cdot 5^{g_3} \cdot 7^{g_4}$

Since there is a unique decomposition of any such product, we can always compute back (decode) to get the unique string of symbols. Gödel used this trick at two levels: to encode sequences of symbols and to encode sequences of formulas.

Recursive sets of (code numbers of) expressions

We can now define, in a derivative sense of "recursive":

- A **set of symbols or expressions** is **recursive** iff the set of code numbers of elements of the set in question is recursive (remember the previous definition: if a set is recursive, its characteristic function is recursive, and so the belongingness to the set is decidable).
- A **language** is **recursive** iff the set of code numbers of symbols in the language is recursive.

Arithmetization of syntax

Important consequences of the numbering:

- The **logical operations** of negation, disjunction, existential quantification, substitution of a term (for free occurrences of a variable), and so on, are recursive. (Example: the function taking as argument the code number of a sentence and giving as value the code number of the negation of this sentence is recursive.)
- The sets of **formulas** and of **sentences** of our language are recursive.
- If Δ is a recursive set of sentences, then this relation is recursive: n **is the code number of a derivation of the sentence with code number** d **from** Δ.

Any axiomatized theory is **semirecursive**

The set of sentences deducible from a given recursive set of sentences is semirecursive.

- Let Σ be a recursive set of sentences
- Rsd: s is the code number of a proof of the sentence with code number d from Σ.
- The set S of code numbers of sentences deducible from Σ is defined by:

$$Sd \leftrightarrow \exists s\ Rsd$$

Positively effectively decidable, because of the (unbounded) existential quantification: if such a witness s is found, we know $d \in S$ but not negatively, looking forever for a corresponding s.

What about a **complete** axiomatized theory?

$$Sd \leftrightarrow \exists s\ Rsd$$

- If T is (negation-) complete, then for any number d such that it is the code number of a sentence D, either $d \in S$, or the code number $d*$ of the sentence $\neg D \in S$.
- If T is inconsistent, d and $d*$ are both in S (since any sentence can be deduced from a contradiction).
- If T is consistent, d and $d*$ cannot be both in S. So exactly one of them is in S.

Any **complete** axiomatized theory is **recursive**

- We define the **complement** of the set of code numbers of sentences in T by the union of the set of numbers which are not codes of any sentence at all and the set of code numbers of sentences not in T. So the complement is semirecursive as well.
- By **Kleene's complementation principle**: if both a set and its complement are semirecursive, then they both are in fact recursive. (intuitively: we can check in both sets, and one of them will give us a positive answer at some point).
- For any given number d, we can decide if it is the code number of a theorem of T: T is **decidable**.

"Talking about" recursive functions IN $L*$

Now we want to show that we can talk about recursive functions directly in our formal language of arithmetic. For this, we are going to use two notions: definability and representability.

Definability of recursive functions and relations in $L*$

- A formula $F(x)$ of the language or arithmetic **arithmetically defines** a set S of natural numbers iff:

$$\text{for all } n \in \mathbb{N},\ n \in S \text{ iff } F(\mathbf{n}) \text{ is } \textbf{true} \text{ in } N*$$

Similarly for functions and relations:

- A function is arithmetical ($=$ arithmetically definable) iff its graph relation is arithmetical. For a one-place function: f is arithmetical iff there is a formula $F(x, y)$ of the language of arithmetic such that for all n and m we have $f(n) = m$ iff $F(\mathbf{n}, \mathbf{m})$ is true in $N*$.
- A formula $F(x, y)$ arithmetically defines a two place relation R on natural numbers iff for all $n \in \mathbb{N}$, n and m we have $(n, m) \in R$ iff $F(\mathbf{n},\mathbf{m})$ is **true** in $N*$.

(similarly for many place relations and functions).

Definability of recursive functions in $L*$

We can show (through a few complications) that all recursive functions and relations are definable by formulas of $L*$, and by formulas which do not contain unbounded universal quantification ($\exists$-rudimentary formulas).

Representability of recursive functions

To strengthen this result, we use a theory Q of minimal arithmetic. We can show that for any recursive function f, we can find a formula F such that such that F **represents** f in Q.

- A formula $F(x, y)$ arithmetically **represents** a binary relation R on natural numbers iff for all $n \in \mathbb{N}$, n and m we have $(n, m) \in R$ iff $F(\mathbf{n}, \mathbf{m})$ is **provable** from the axioms of Q.

Minimal arithmetic Q

A finite set of axioms, which are correct and strong enough to prove all correct existential-rudimentary sentences (all correct sentences which don't contain unbounded universal quantification). They are not sufficient for number theory, but any set of adequate axioms of number theory should include them.

Axioms of Q

$\forall x, \forall y$:

1. $\mathbf{0} \neq x'$
2. $x' = y' \rightarrow x = y$
3. $x + \mathbf{0} = \mathbf{0}$
4. $x + y' = (x + y)'$
5. $x \cdot \mathbf{0} = \mathbf{0}$
6. $x \cdot y' = (x \cdot y) + x$
7. $x \not< \mathbf{0}$
8. $x < y' \leftrightarrow ((x < y) \vee x = y)$
9. $\mathbf{0} < y \leftrightarrow (y \neq \mathbf{0})$
10. $x < y' \leftrightarrow ((x < y) \,\&\, x \neq y)$

A weak theory for a strong result

- Since every recursive function is definable by a correct (not unbounded universal) formula of $L*$, and since Q proves every such formula, every recursive function is representable in Q.
- Therefore, every recursive function is representable in any extension of Q.

Combining both directions

- We have seen how we can encode any expression of $L*$ by numbers and use recursive functions to "talk about" these expressions in $L*$.
- We have seen how we can "talk about" recursive functions in $L*$.
- Combining both, we can now "talk about" expressions of $L*$ IN $L*$.

Diagonalization

- Let $\ulcorner A \urcorner$ be the code number of a formula A.
- The **diagonalization** of A is the expression:

$$\exists x(x = \ulcorner A \urcorner \& A)$$

Diagonalization of a formula with one free variable

When A is of the form $F(x)$:

- For $F(t)$ any instance of $F(x)$: $\exists x(x = t \ \& \ F(x)) \leftrightarrow F(t)$.
- Diagonalization of A: $\exists x(x = \ulcorner \mathrm{A} \urcorner \& \mathrm{A}(\ulcorner \mathrm{A} \urcorner))$
- $\exists x(x = \ulcorner \mathrm{A} \urcorner \& \mathrm{A}(\ulcorner \mathrm{A} \urcorner)) \leftrightarrow \mathrm{A}(\ulcorner \mathrm{A} \urcorner)$
- In this particular case, the diagonalization will be true in the standard interpretation iff A is satisfied by its own Gödel number in the standard interpretation.

The Diagonal Lemma

Let T be a theory containing Q.

- Then for any formula B(y) there is a sentence G such that:

$$\vdash G \leftrightarrow B(\ulcorner \mathrm{G} \urcorner)$$

Any consistent theory of arithmetic is undefinable

Let T be a consistent theory extending Q. Then the set Θ of code numbers of theorems of T is not arithmetically definable in T:

- Assume Θ is definable.
- Let $\theta(y)$ be the formula defining it. ($y \in \Theta$ iff $\theta(y)$ is true).
- By the diagonal lemma, there is a sentence G such that:

$$\vdash G \leftrightarrow \neg\theta(\ulcorner G \urcorner)$$

Any consistent theory of arithmetic is undecidable

- Let T be a consistent extension of Q.
- We've just shown that T is not arithmetically definable.
- We already know that every recursive set is arithmetically definable.
- T is not recursive.
- By Church's thesis (every decidable function is recursive), T is not decidable.

The final step

From these two results:

- If T is an axiomatizable complete theory, T is decidable.
- No consistent extension of Q is decidable.

There is no consistent complete axiomatizable extension of Q.

The end of Hilbert's program?

- maybe, but there is worse!

Second Incompleteness Theorem

- In addition to being incomplete (there are blindspots of the system), a consistent extension of Q cannot prove its own consistency!

The end of Hilbert's program!

"If a nice arithmetical theory T can't even prove itself to be consistent, it certainly can't prove that a richer theory T+ is consistent (since if the richer theory is consistent, then any cut-down part of it is consistent).
Hence we can't use "safe" reasoning of the kind we can encode in ordinary arithmetic to prove other more "risky" mathematical theories are in good shape. For example, we can't use unproblematic arithmetical reasoning to convince ourselves of the consistency of set theory (with its postulation of a universe of wildly infinite sets).
And that is a very interesting result, for it seems to sabotage what is called Hilbert's Programme, which is precisely the project of defending the wilder reaches of infinitistic mathematics by giving consistency proofs which use only safe methods."
(Smith)

Back to set theory

- We said that set theory was inconsistent (Russell's paradox). Then what is the solution?
- To avoid problems, avoid self-reference!

The cumulative hierarchy of sets

- We can build sets only from what has been "previously" formed: start with the empty set $\emptyset$. Form the set $\{\emptyset\}$ containing $\emptyset$ as its only member. Now form the set containing the empty set plus the set we've just built. Keep on going, at each stage forming sets having as elements only previously constructed sets.
- One level at a time!
- Result: the so-called Russell's set is not a set. (Similarly for other collections which were leading to paradoxes without this restriction: there is no such thing as the "set of all sets" anymore, for instance.)

Back to the Liar

- Tarski's theorem of undefinability of truth: the set of code numbers of correct sentences of arithmetic is not definable.
- Let l be any sentence of a formal language and "l" be something like its name. Is it possible to design a formal system in such a way that we have a predicate T of truth in the formal language, such that:

  $$M \vDash T(''l'') \text{ iff } M \vDash l \text{ ?}$$

  If we want a total truth function and if we want to admit liar sentences, then no.
- Solution? Avoid self-reference!
- One level at a time!
- But are Liar sentences ill-formed?

Example: Kripke's theory of truth

- Three-valued semantics.
- A sentence containing the truth predicate T applied to a sentence I will be undefined (neither true nor false) as long as the truth-value of the sentence I is not previously defined. But it will become true or false at the next level! Exactly as a set which would not be defined yet would be defined only as soon as all his elements are. We can go to higher and higher levels, but one step at a time.

A difference between minds and machines?

- These solutions seem to simply avoid/forbid self-reference.
- But natural language allows us to do it and we are perfectly able to "see" that the sentence "this sentence is unprovable" is not false (and therefore, as long as we admit that it must be either true or false, must be true).

Conclusion

"Either mathematics is too big for the human mind or the human mind is more than a machine." (Gödel)

Bibliography

- Boolos Georges S., Burgess John P., Jeffrey Richard C., 2007, *Computability and Logic*, Fifth Edition, Cambridge, CUP.
- Cook Roy T., 2009, *A Dictionary of Philosophical Logic*, Edinburgh, Edinburgh University Press.
- Kripke Saul, 1975, "Outline of a Theory of Truth", *The Journal of Philosophy*, Vol. 72, No. 19, 690-716.
- Gödel Kurt, "Uber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme", in Solomon Feferman, ed, 1986, *Kurt Gödel: Collected Works,* volume 1, 144-195, Oxford, OUP. (Original German text with parallel English translation).
- Smith Peter, 2007, *Introduction to Gödel's theorems*, Cambridge, CUP.