Gödel's Incompleteness Theorem

Leon Weber

27 January 2015

In this talk, Gödel's first and second incompleteness theorems will be formulated and proven. Informally, they assert that every formal system that can express certain arithmetic truths contains true sentences which cannot be proven and that this system cannot prove its own consistency.

(Semi-)random facts about Gödel:

- Was a good friend of Albert Einstein
- Provided a formal proof of God's existence and contributed to modern physics
- Was married to a Adele Nimbursky (a cabaret dancer)

The theory $Th(\mathcal{N}, +, \cdot)$

Definition 0.1 (Syntax of $Th(\mathcal{N}, +, \cdot)$)**.**

- *Recursive definition of 'term':*
  - *Every natural number $n \in \mathbb{N}$ is a term.*
  - *Every variable x_i, $i \in \mathbb{N}$ is a term.*
  - *If t_1 and t_2 are terms, then $(t_1 + t_2)$ and $(t_1 \cdot t_2)$ are terms.*
- *Recursive definition of 'formula':*
  - *If t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula.*
  - *If F, G are formulae, then $\neg F$, $(F \wedge G)$ are formulae.*
  - *If x is a variable and F is a formula, then $\forall x.F$ and $\exists x.F$ are formulae.*

Definition 0.2 (Semantics of $Th(\mathcal{N}, +, \cdot)$(informal))**.** *The symbols signify the structures as given by the standard model of Peano Arithmetic in the usual way.*

Definition 0.3 (Truth of a formula)**.** *Let F, G be formulae, t_1, t_2 be terms, and x be a variable, then*

- *$(t_1 = t_2)$ is true, iff[1] $\varphi(t_1 = t_2)$ holds for any assignment φ.*
- *$\neg F$ is true, iff F is not true.*
- *$(F \wedge G)$ is true, iff F is true and G is true.*
- *$(F \vee G)$ is true, iff F is true or G is true.*
- *$\exists x.F$ is true, iff there is a $n \in \mathbb{N}$ such that $F(x/n)$[2] is true.*
- *$\forall x.F$ is true, iff $F(x/n)$ is true for any $n \in \mathbb{N}$.*

[1] Read: If and only if

[2] Read: F with x being replaced by n

Definition 0.4 (Arithmetic Representability of functions)**.** *A function $f : \mathbb{N}^k \to \mathbb{N}$ is arithmetically representable (a.r.), iff there is a formula of $Th(\mathcal{N}, +, \cdot)$ $F(x_1, x_2, \ldots, x_k, x_{k+1})$, such that for every $n_1, n_2, \ldots, n_k, n_{k+1} \in \mathbb{N}$ holds:*

$f(n_1, n_2, \ldots, n_k) = n_{k+1}$ iff $F(n_1, n_2 \ldots, n_k, n_{k+1})$ *is true.*

The Main Theorem

Lemma 0.1 (Important arithmetically representable functions).

- *Addition ('+') is a.r.*
- *Multiplication ('$\cdot$') is a.r.*
- *Division (of natural numbers) ('div') is a.r.*
- *Modulo ('mod') is a.r.*
- $n_i = a\ mod(1 + (i+1) \cdot b)$ *is a.r.*

Fact 0.1 (Every finite sequence of natural numbers can be identified by only two numbers). *For every sequence of natural numbers $(n_0, n_1, \ldots, n_k)$ there are natural numbers a and b, such that for $i \in 0, 1, \cdots, k$ holds:*

$n_i = a \cdot mod(1 + (i+1) \cdot b)$

Lemma 0.2 (Every WHILE-program is a.r.). *For every WHILE-program* [3] *P with variables $x_0, x_1, \ldots, x_k$ we can construct a $Th(\mathcal{N}, +, \cdot)$-Formula F_p with free variables $x_0, x_1, \ldots, x_k$ and $y_0, y_1, \ldots, y_k$ such that for any $m_i \in \mathbb{N}$ and $n_i \in \mathbb{N}$:*

$F_p(m_0, m_1, \ldots, m_k, n_0, n_1, \ldots, n_k)$ *is true* iff *the computation of P is started with $x_0 = m_0, x_1 = m_1, \ldots, x_k = m_k$ and halts with $x_0 = n_0, n_1 = n_1, \ldots, x_k = n_k$.*

Proof. This proof works by induction on the structure of P:

Basis:

$P =' x_i := x_j + / - c'$: Let $F_p := (y_i = x_j + / - c) \bigwedge_{k \neq i} (y_k = x_k)$

Step:

case $P =' Q; R'$: By induction hypothesis there are F_Q and F_R.

Let $F_P := \exists . z_0, \exists . z_1, \ldots, \exists . z_k . (F_Q(x_0, x_1, \ldots, z_k, z_0, z_1, \ldots, z_k) \wedge F_R(z_0, z_1, \ldots, z_k, y_0, y_1, \ldots, y_k))$

case $P =' WHILE\ x_i\ DO\ Q\ END'$: By induction hypothesis there is F_Q. Let

$F_P := \exists a_0 . \exists b_0 . \exists a_1 . \exists b_1 . \cdots \exists a_k . \exists b_k . \exists t.$ [4]
$[G(a_0, b_0, 0, x_0) \wedge G(a_1, b_1, 0, x_1) \wedge \cdots \wedge G(a_k, b_k, 0, x_k)] \wedge$
$[G(a_0, b_0, t, y_0) \wedge G(a_1, b_1, t, y_1) \wedge \cdots \wedge G(a_k, b_k, t, y_k)] \wedge$ [5]
$\forall j < t . \exists w . (G(a_i, b_i, j, w) \wedge (w > 0)) \wedge G(a_i, b_i, t, 0) \wedge$ [6]
$\forall j < t . \exists w_0 . \exists w_1 . \cdots \exists w_k . \exists w'_0 . \exists w'_1 . \cdots \exists w'_k .$
$[F_Q(w_0, w_1, \ldots, w_k, w'_0, w'_1, \ldots, w'_k) \wedge$
$G(a_0, b_0, j, w_0) \wedge G(a_1, b_1, j, w_1) \wedge \cdots \wedge G(a_k, b_k, j, w_k) \wedge$
$G(a_0, b_0, j+1, w'_0) \wedge G(a_1, b_1, j+1, w'_1) \wedge \cdots \wedge G(a_k, b_k, j+1, w'_k)]$
[7] □

Fact 0.2 (Every Turing Machine is a.r. by a formula with only one free variable). *From the lemma above and the fact that you can simulate a*

[3] WHILE-programs (which are as expressive as turing machines) are composed of the following elements:

- variables: x_i
- constants: $n \in \mathbb{N}$
- delimiters: ; :=
- operators: + −
- keywords: WHILE DO END

Syntax and semantics of WHILE-programs are defined in the canonical way. The only noteworthy difference from the semantics of some standard programming languages is: 'WHILE x DO stuff END' executes *stuff* until x=0.

[4] Each a_i, b_i identifies the sequence of values of the variable x_i. t is the number of executions after which are counter variable reaches zero.

[5] This formula asserts that the initial/final value of x_i is in fact the first/last number in the sequence described by a_i, b_i

[6] This formula asserts that the value of the counter variable is zero exactly after t steps and never becomes zero at any step before.

[7] This formula asserts that the transition of the value of x_i in step $j \to j+1$ is in fact governed by the formula F.

Turing Machine (TM) with a WHILE-program, follows that every TM is a.r. In fact, given a TM M and a string w it is possible to construct a formula $\varphi_{M,w}$ *with a single free variable x such that*
$\exists x.\varphi_{M,w}$ iff M *accepts* w

Theorem 0.1. $Th(\mathcal{N}, +, \cdot)$*is undecidable*

Proof. We prove this by mapping the *Halting Problem*[8] to the problem if a formula of $Th(\mathcal{N}, +, \cdot)$is true: Given a TM M and a string w construct $\varphi_{M,w}$ as described in the fact above.

Let D be a TM that decides if a formula of $Th(\mathcal{N}, +, \cdot)$is true. It is obvious that D accepts $\varphi_{M,w}$ *iff* M accepts w. □

[8] Here for convenience, the form of the *Halting Problem* is taken to be $\{(M : Code\ of\ TM,\ w : String) | M\ accepts\ w\}$

Definition 0.5 (Partial characterization of proof systems)**.** *Let* π *be a sequence of FOL-Formulae and* ψ *be a sentence*[9] *of FOL. If (*note the missing 'f'*)* π *is a proof, then*

[9] A formula with no free variables

1. $\{(\psi, \pi) | \pi\ is\ a\ proof\ of\ \psi\}$ *is decidable.*
2. *if* π *proves* ψ *then* ψ *is true.*
3. *if* π *proves* $\varphi \to \psi \wedge \varphi$ *then* π *proves* φ.[10]

[10] This rule is called 'modus ponens'.

Theorem 0.2. *The language* $\{w \in FOL - Formulae \mid w\ is\ provable\ in$ $Th(\mathcal{N}, +, \cdot)\}$ *is Turing-recognizable.*

Proof. Proofs (as characterized here) are finite sequences of formulae. Those formulae too are finite and comprised of elements from a finite set of symbols. Thus, there is a way of systematically generating all possible formulae without missing one.

Consequently, we can give an algorithm P for recognizing if a sequence of formulae π is a proof for a given statement ψ: For each sequence run the proof checker asserted in our characterization of proofs on (ψ, π). If it accepts accept, otherwise try the next sequence. □

Theorem 0.3 (First Incompleteness Theorem)**.** *There is a true statement of* $Th(\mathcal{N}, +, \cdot)$*which is not provable.*

Proof. This can be proved by contradiction. Assume that every statement of $Th(\mathcal{N}, +, \cdot)$is provable. Then, consider the following algorithm: Given a statement ψ of $Th(\mathcal{N}, +, \cdot)$. Run the recognizer P on ψ and $\neg\psi$ in parallel. By the preceding theorem and the law of the excluded middle[11], one of the two threads must accept. This algorithm decides $Th(\mathcal{N}, +, \cdot)$and thus, contradicts a theorem above. $\lightning$ □

[11] For every formula ψ holds: either ψ or $\neg\psi$ is true.

Theorem 0.4. *It is possible to construct a formula* ψ *which is unprovable in* $Th(\mathcal{N}, +, \cdot)$.

Proof.

- Idea: Define a sentence which says 'This sentence is not provable'.
- Construction: Let S be a Turing Machine implementing the following algorithm:
  1. Obtain self-description $\langle S \rangle$. [12]
  2. Construct $\psi = \neg \exists x . \varphi_{S,0}$
  3. Run algorithm P from above on ψ.
  4. If P accepts, accept. If P rejects, reject.

[12] That this is always possible is guaranteed by the *Recursion Theorem*: For every TM T that computes the function $t : \Sigma^* x \Sigma^* \to \Sigma^*$ there is a TM R that computes the function $r : \Sigma^* \to \Sigma^*$ such that for every $w \in \Sigma^*$, $r(w) = t(\langle R \rangle, w)$.

- Assume ψ is provable in $Th(\mathcal{N}, +, \cdot)$. Thus, for any input S will halt and accept after a finite amount of time. But, from the construction of ψ follows that S will accept ψ *iff* ψ is *not* true in $Th(\mathcal{N}, +, \cdot)$. This contradicts our assumption that only true statements can be proven. ↯

□

Theorem 0.5 (Second Incompleteness Theorem)**.** *It is impossible to prove the consistency of $Th(\mathcal{N}, +, \cdot)$by means of $Th(\mathcal{N}, +, \cdot)$.*

Proof. Idea: Assume it is possible to prove the consistency of $Th(\mathcal{N}, +, \cdot)$from within $Th(\mathcal{N}, +, \cdot)$. Then because '$Th(\mathcal{N}, +, \cdot)$is consistent' is a.r., the implication 'if $Th(\mathcal{N}, +, \cdot)$is consistent then ψ' is provable *in* $Th(\mathcal{N}, +, \cdot)$. But then by modus ponens and our assumption, ψ is provable, which contradicts the theorem above. Thus, $Th(\mathcal{N}, +, \cdot)$cannot prove its own consistency. □

References

John Dawson. *Logical Dilemmas: The Life and Work of Kurt Gödel*. A K Peters/CRC Press, 1996.

Panu Raatikainen. Gödel's incompleteness theorems. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2015 edition, 2015.

Uwe Schöning. *Theoretische Informatik - kurz gefasst (German Edition)*. Spektrum Akademischer Verlag, 2001.

Michael Sipser. *Introduction to the Theory of Computation*. PWS Pub. Co., 1996.