Lecture 15

Gödel's Incompleteness Theorem

February 26, 1998
Lecturer: Richard Ladner
Notes: Andrew Keller

Let $\mathcal{N} = (N, s, +, \cdot, ^\wedge, <, 0)$ where N are the natural numbers, s, the successor function, $^\wedge$, exponentiation, etc.

Theorem 15.1 (Gödel's Incompleteness Theorem): Th($\mathcal{N}$) is *not* recursively enumerable.

In fact, it turns out that Th($\mathcal{N}$) is beyond the arithmetic hierarchy. The arithmetic hierarchy can be defined by languages reducible to $\emptyset^{(k)}$ where $\emptyset^{(0)} = \emptyset$ and $\emptyset^{(k+1)}$ is the set of all oracle Turing machines encoding $\hat{T}$ such that T accepts $\hat{T}$ with oracle $\emptyset^{(k)}$.

Why is it called the "incompleteness" theorem? Suppose there were recursively enumerable set of arithmetic axioms, $\mathcal{A}$, such that CN($\mathcal{A}$) $\subseteq$ Th($\mathcal{N}$). The Incompleteness Theorem dictates that Th($\mathcal{N}$) is *not* recursively enumerable, so CN($\mathcal{A}$), which is recursively enumerable, cannot be equal to Th($\mathcal{N}$). Hence there must be true statements of arithmetic that cannot be proven from the axioms $\mathcal{A}$. This killed Hilbert's plan for a mechanized system of deriving truths, and is thought to be one of the ten most significant theorems of our century (*USA Today* readers poll, 1996).

The proof described below is simpler than Gödel's original. (For example, he didn't use exponentiation or even Turing machines, but instead invented his own theory of computation and used a diagonalization argument.) It is a reduction from a non-r.e. language to $\{\alpha : \mathcal{N} \models \alpha\}$. Given Turing Machine T which recognizes the *non-recursive* language $L(T)$, this reduction on input x constructs formula α_x such that $x \in \overline{L(T)} \Leftrightarrow \mathcal{N} \models \alpha_x$.

Proof: We start with a Turing Machine $T = (S, \Sigma, \Gamma, \delta, s)$. Let $\Delta = S \cup \Gamma \cup \{\#\}$. We must now make a connection between strings and numbers such that each string $s \in \Delta^*$ corresponds to a unique number. Given that the cardinality of Δ is d, one denotes each letter $a \in \Delta$ by the number $\underline{a}$ where $\{\underline{a} : a \in \Delta\} = \{0, 1, 2, \ldots d-1\}$. The number x corresponding to string s $= \sigma_0, \sigma_1, \ldots, \sigma_{k-1}$ is then given by: $x = \sum_{i=0}^{k-1} \underline{\sigma}_i d^i$. The number x is thus a d-ary numerical representation of s. We assume that $\underline{\#} \neq 0$.

A *partial computation* of T has the form $\#C_0\#C_1 \cdots C_k\#$, where the C_i, $0 \leq i \leq k$ are configurations of T of the same length, each following legally from its predecessor, and C_0 is the intial configuration. This computation then comprises a string $\sigma_0\sigma_1 \cdots \sigma_m$ with d-ary numerical representation $n = \sum_{i=0}^{m-1} \underline{\sigma}_i d^i$.

The partial computation of T upon input $x = x_1 x_2 \cdots x_t$ must have the following three properties:

1. $\sigma_0 = \#, \sigma_1 = s, \sigma_2 = x_1, \ldots, \sigma_{t+1} = x_t, \sigma_{t+2} = \sqcup, \ldots, \sigma_{l-1} = \sqcup, \sigma_l = \#$
2. $\forall p,\ 0 \leq p \leq n - l - 1,\ \sigma_{p+l+1} = f_T(\sigma_p, \sigma_{p+1}, \sigma_{p+2}, \sigma_{p+3})$
3. $\sigma_m = \#$

Wher f_T is the function that determines the $(p+l+1)$-st symbol of a partial computation from the p-th, $(p+1)$-st, $(p+2)$-nd, and $(p+3)$-rd symbols of the partial computation. The first requires that the first configuration is correct, the second requires that each pair of consecutive configurations are legal transitions according to T, and the third requires that the last configuration is complete.

We must now impose the above three properties on a computation string with d-ary numerical representation n using the language of $\mathcal{N}$. Toward this goal, we define the following helper functions:

- "n divided by m = q", $Div(n, m, q) = \exists r < m(m \cdot q + r = n)$
- "n divided by m yields the remainder r", $mod(n, m, r) = \exists q \leq n(m \cdot q + r = n \text{ and } r < m)$
- "the i^{th} dit of n is s", $D(i, n, s) = \exists q \leq n(Div(n, d^\wedge i, q) \wedge Mod(q, d, s))$

Note that the quantifiers in all three helper functions are *bounded* within a range of possible values

Furthermore, each d-ary numerical value i can be represented in the language of $\mathcal{N}$ as $s^i(0)$, denoted $\underline{i}$, where s is the successor function of $\mathcal{N}$.

Now define the formula $pcomp_{x,T}(n)$ in the language of $\mathcal{N}$ which forces the 3 properties listed above onto the partial computation string of T with d-ary numerical representation n:

$pcomp_{x,T}(n) = (\exists m \leq n)(\exists l \leq n)(\psi_1 \wedge \psi_2 \wedge \psi 3)$, where terms ψ_1, ψ_2, ψ_3 impose the above properties 1,2,and 3, respectively:

- $\psi_1 = Div(n, d^\wedge m, \underline{\#}) \wedge D(0, n, \underline{\#}) \wedge D(\underline{1}, n, \underline{s}) \wedge D(\underline{2}, n, \underline{x}_1) \wedge \cdots \wedge D(\underline{t}+\underline{1}, n, \underline{x}_t) \wedge (\forall i \leq l - \underline{1})(\underline{t} + \underline{1} < i \Rightarrow D(i, n, \underline{\sqcup})) \wedge D(l, n, \underline{\#})$
- $\psi_2 = (\forall p \leq m - l - \underline{1})(\bigwedge_{\sigma_0\sigma_1\sigma_2\sigma_3}(D(p, n, \underline{\sigma}_0) \wedge D(p+\underline{1}, n, \underline{\sigma}_1) \wedge D(p+\underline{2}, n, \underline{\sigma}_2) \wedge D(p+\underline{3}, n, \underline{\sigma}_3) \Rightarrow D(p + l + \underline{1}, n, \underline{f_T(\sigma_0, \sigma_1, \sigma_2, \sigma_3)}))$
- $\psi_3 = Div(n, d^\wedge m, \underline{\#})$

Finally, define the related formula $comp_{x,T}$ which additionally requires that the computation string for T with d-ary numerical representation n have an accepting state, A:

$comp_{x,T}(n) = pcomp_{x,T}(n) \wedge (\exists p \leq n) D(p, n, \underline{A})$

Claim: $x \in \overline{L(T)} \Leftrightarrow \mathcal{N} \models \forall n\ \neg comp_{x,T}(n)$

The above is a reduction from a non-recursive Co-r.e. language to $\mathcal{N}$ model satisfiability:

$\overline{L(T)} \leq_m \{\alpha : \mathcal{N} \models \alpha\}$

Since $\overline{L(T)}$ is not r.e., neither is $\{\alpha : \mathcal{N} \models \alpha\}$.

$\square$

15.1. Natural Axioms for NumberTheory

The set NT include the following 14 axioms:

1. $\forall x(s(x) \neq 0)$
2. $\forall x \forall y(s(x) = s(y) \Rightarrow x = y)$
3. $\forall x(x = 0 \vee \exists y s(y) = x)$
4. $\forall x(x + 0 = x)$
5. $\forall x \forall y(x + s(y) = s(x + y))$
6. $\forall x(x \cdot 0 = 0)$
7. $\forall x \forall y(x \cdot s(y) = (x \cdot y) + x)$
8. $\forall x(x^\wedge 0 = s(0))$
9. $\forall x \forall y(x^\wedge s(y) = (x^\wedge y) \cdot x)$
10. $\forall x(x < s(x))$
11. $\forall x \forall y(x < y \Rightarrow (s(x) \leq y))$
12. $\forall x \forall y(\neg(x < y) \Leftrightarrow y \leq x)$
13. $\forall x \forall y \forall z(((x < y) \wedge (y < z)) \Rightarrow x < z)$
14. $\forall x \forall y \forall z \forall z'(mod(x, y, z) \wedge mod(x, y, z') \Rightarrow z = z')$

Define a *Bounded Formula* as either of the following:

- a quantifier-free formula
- formulas $(\exists x_n < t)\alpha(x_1 \cdots x_n)$ and $(\forall x_n < t)\alpha(x_1 \cdots x_n)$, given that $\alpha(x_1 \cdots x_n)$ is known to be bounded and t is a term over the variables $x_1 \cdots x_{n-1}$.

Theorem 15.2: If α is a bounded sentence (all its variables bounded) and $\mathcal{N} \models \alpha$, then $NT \models \alpha$.

Since alpha is bounded, all its variables have a finite domain, and thus all bounded quantifiers can be expressed by a finite terms. Thus, the sentence is equivalent to a finite length quantifier free formula. Any atomic formula can be verified for its truth or falsity from the axioms NT. Thus, any finite length quantifier free formula can be determined from NT.

Theorem 15.3: If $NT \subseteq \Gamma$ and Γ is consistent, then $CN(\Gamma)$ is *not* computable

On simple consequence is that $CN(NT)$ is not computable

Proof:

Definition: If S and T are 2 sets such that $S \cap T = \emptyset$, then S and T are *recursively inseparable* if there is no computable set R such that $S \subseteq R$ and $T \subseteq \overline{R}$.

For example, the following 2 languages are recursively inseparable:

- $L_1 = \{\hat{T} : T \text{ halts and accepts } \lambda\}$
- $L_2 = \{\hat{T} : T \text{ halts and rejects } \lambda\}$

where $\hat{T}$ is a standard encoding of Turing machines and λ is the empty string. See Papadimitriou, page 63, for a proof that L_1 and L_2 are recursively inseparable.

Claim: $CN(NT)$ and $USAT$ are recursively inseparable, where $USAT = \{\alpha : \models \neg\alpha\}$

Proof: By reduction from L_1 and L_2 above. Given Turing Machine T, construct the formula ϕ_T such that:

1. if λ is accepted by T, then $NT \models \phi_T$
2. if λ is rejected by T, then $\models \neg\phi_T$

To be continued..... $\square$