Math Logic: Model Theory & Computability
Lecture 25

Tarski's theorem. Let $\underline{N} := (\mathbb{N}, 0, S, +, \cdot)$. Then $Th(\underline{N})$ is not arithmetical, i.e. $\ulcorner Th(\underline{N}) \urcorner := \{ \ulcorner \varphi \urcorner : \underline{N} \models \varphi \}$ is not arithmetical.

Proof. Follows immediately by an application of the fixed point lemma, left as HW. $\square$

Now let $T \subseteq Th(\underline{N})$ be a computable theory. Then we can encode sequences $(\varphi_1, \varphi_2, \dots, \varphi_n)$ of σ_{arthm}-formulas into a number by $\langle \ulcorner \varphi_1 \urcorner, \ulcorner \varphi_2 \urcorner, \dots, \ulcorner \varphi_n \urcorner \rangle$. Then it is easy to see that the binary relation $Proof_T \subseteq \mathbb{N}^2$ defined by

$$Proof_T(a, b) \quad :\Longleftrightarrow \quad a = \ulcorner \varphi \urcorner \text{ for some } \sigma_{arthm}\text{-formula } \varphi \text{ and } b \text{ is a code of a proof of } \varphi \text{ from } T$$

is computable: indeed, we can decode $b = \langle \ulcorner \varphi_1 \urcorner, \ulcorner \varphi_2 \urcorner, \dots, \ulcorner \varphi_n \urcorner \rangle$, check that $\varphi_n = \varphi$, and check whether each φ_i is in $Axiom(\sigma_{arthm})$ or in T, and both sets are computable, or whether there exist $j, k < i$ such that φ_i is obtained from φ_j and φ_k by MP, i.e. that $\varphi_j = (\varphi_k \to \varphi_i)$. Thus we have shown:

Prop. If $T \subseteq Th(\underline{N})$ is computable, then $Proof_T(\cdot, \cdot)$ is computable, in particular, arithmetical.

Thus, let $\underline{Proof_T}(x, y)$ be an extended σ_{arthm}-formula such that for all $a, b \in \mathbb{N}$,

$$Proof_T(a, b) \text{ holds iff } \underline{N} \models \underline{Proof_T}(a, b).$$

Finally, let $\underline{Provable_T}(x) := \exists y \, \underline{Proof_T}(x, y)$. Hence, for each $a \in \mathbb{N}$,

$$\underline{N} \models \underline{Provable_T}(a) \text{ iff } a = \ulcorner \varphi \urcorner \text{ for some } \sigma\text{-arthm formula } \varphi \text{ and } T \vdash \varphi.$$

Gödel's Incompleteness. Let $\sigma := (0, S, +, \cdot)$ and $\underline{N} := (\mathbb{N}, 0, S, +, \cdot)$. Then every computable subtheory $T \subseteq Th(\underline{N})$ is incomplete. In particular, PA is incomplete.

Proof 1 of Gödel Incompleteness (Liar's Paradox). Let $T \subseteq Th(\underline{N})$ be a computable subtheory so there is an extended σ_{arith}-formula $\underline{\text{Provable}_T}(x)$ as above. By the fixed point lemma, there is a σ_{arith}-sentence γ_T, called the Gödel sentence for T, such that (*) $\underline{N} \models (\gamma_T \leftrightarrow \neg \text{Provable}_T(\ulcorner \gamma_T \urcorner / x))$. We show that $\gamma_T \in Th(\underline{N})$, i.e. $\underline{N} \models \gamma_T$, but $T \nvdash \gamma_T$, hence T is incomplete. Note:

$$
\begin{aligned}
T \vdash \gamma_T \quad &\Longleftrightarrow \quad \text{there is } b \in \mathbb{N} \text{ such that } \text{Proof}(\ulcorner \gamma_T \urcorner, b) \text{ holds} \\
&\Longleftrightarrow \quad \underline{N} \models \underline{\text{Provable}_T}(\ulcorner \gamma_T \urcorner / x) \\
(\text{by } (*)) \quad &\Longleftrightarrow \quad \underline{N} \not\models \gamma_T \\
&\Longrightarrow \quad T \nvdash \gamma_T .
\end{aligned}
$$

This contradiction shows that $T \nvdash \gamma_T$, so $\underline{N} \models \neg \underline{\text{Provable}_T}(\ulcorner \gamma_T \urcorner / x)$ hence $\underline{N} \models \gamma_T$ by (*). Thus, T is incomplete. □

Proof 2 of Gödel Incompleteness (from Tarski's theorem). If there were a computable and complete $T \subseteq Th(\underline{N})$, then by completeness, for each σ_{arith}-sentence φ,

$$\underline{N} \models \underline{\text{Provable}_T}(\ulcorner \varphi \urcorner / x) \quad \text{iff} \quad \varphi \in Th(\underline{N}).$$

But this shows that $\ulcorner Th(\underline{N}) \urcorner := \{ \ulcorner \varphi \urcorner : \varphi \in Th(\underline{N}) \}$ is definable in $\underline{N}$, i.e. arithmetical, contradicting Tarski's theorem. □

Quine (a program that prints its own code).

Here is a program that's not a Quine but will give us an idea on how to write a Quine.

```
NotQuiteQuine()
{
  x := "Banach-Tarski x :=";
  for (i:=0, i < length(x); i:=i+1)
  {
    Print(x[i]);
    if (i ≥ 1 ∧ x[i-1] = 'x' ∧ x[i] = ':=')
    {
      Print('"');
      Print(x);
      Print('"');
    }
  }
}
```

This program just prints

Banach-Tarski x := "Banach-Tarski x :="

```
Quine()
  {
      x :=  "Quine()
              {
                        x := ;
                        for (i := 0; i < length(x); i := i + 1)
                        {
                          Print(x[i]);
                          if (i ⩾ 1 ∧ x[i − 1] = 'x' ∧ x[i] = ':=')
                          {
                            Print('"');
                            Print(x);
                            Print('"');
                          }
                        }
              }"
        for (i := 0; i < length(x); i := i + 1)
        {
          Print(x[i]);
          if (i ⩾ 1 ∧ x[i − 1] = 'x' ∧ x[i] = ':=')
          {
            Print('"');
            Print(x);
            Print('"');
          }
        }
    }
```