Scribe Notes for *Symbolic Computation*
Class 11—June 4, 1996

Scribes: Craig Struble and Julie Mihalcik-Eno

Abstract

Today's class covered *Turing reduction*. Using Turing reduction, languages can be related. By relating a language L whose type is unknown to a language L_2 whose type is known, the type of L can be bounded or determined.

1 Review

Recall the languages,

$$\begin{aligned} \Pi_0 &= \{\#\alpha(G)\#\alpha(x) \mid x \in L(G)\} \subseteq \Delta^* \\ \Pi_1 &= \{\#\alpha(G)\# \mid \alpha(G) \in L(G)\} \subseteq \Delta^* \end{aligned}$$

where $\Delta = \{0, 1, \#\}$. Based on our observations from last session, $\Pi_0 = L(U)$ and Π_1 are recursively enumerable. However, $\overline{\Pi_1}$ is not recursively enumerable which we said implied $\overline{\Pi_0}$ is also not recursively enumerable. Hence, Π_0 and Π_1 are not recursive languages.

We will establish this more formally using Turing reduction.

2 Turing Reduction

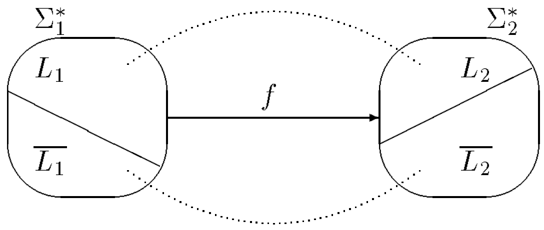
Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$. L_1 is *Turing reducible* to L_2 if there exists a computable total function

$$f : \Sigma_1^* \to \Sigma_2^*$$

such that

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

So, if L_1 is Turing reducible to L_2, there is a computable total function that maps strings between Σ_1^* and Σ_2^* so that the membership in the languages are preserved. The following figure demonstrates this relationship.



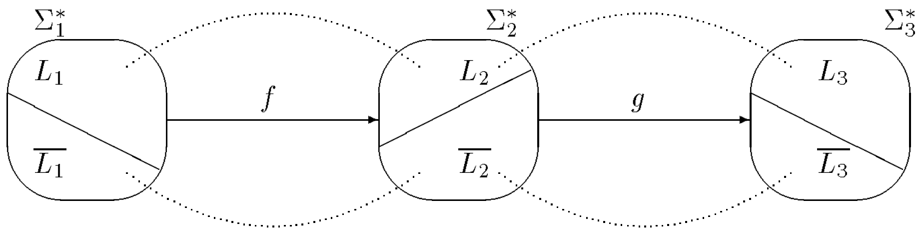
When L_1 is Turing reducible to L_2, we write

$$L_1 \leq_{\mathrm{T}} L_2.$$

This relationship is transitive. That is,

$$L_1 \leq_{\mathrm{T}} L_2 \leq_{\mathrm{T}} L_3 \Rightarrow L_1 \leq_{\mathrm{T}} L_3.$$

Pictorially, this looks like



where f, g are computable total functions that map between the sets of strings as described above. It is easy to see that by composing f and g, we get a function $f \circ g : \Sigma_1^* \rightarrow \Sigma_3^*$ that is also a computable total function and maintains the property that $x \in L_1 \Leftrightarrow f(g(x)) \in L_3$.

Theorem 1 *Suppose $L_1 \leq_{\mathrm{T}} L_2$. Then the following statements are true:*

1. *If L_2 is recursively enumerable, then L_1 is recursively enumerable.*
2. *If L_2 is recursive, then L_1 is recursive.*
3. *If L_1 is not recursively enumerable, then L_2 is not recursively enumerable.*
4. *If L_1 is not recursive, then L_2 is not recursive.*

3 Example 1

Using Π_0 and Π_1, let us show that $\Pi_1 \leq_{\mathrm{T}} \Pi_0$. Let $f : \Delta^* \rightarrow \Delta^*$ be defined such that

$$f(\#\alpha(G)\#) = \#\alpha(G)\#\alpha(\alpha(G))\#.$$

That is, we let $\alpha(G)$ be our x in the definition of Π_0. Strings in Π_1 will be in Π_0 if and only if the grammar G generates an encoded copy of itself. So, f maintains membership between the languages. Thus, $\Pi_1 \leq_{\mathrm{T}} \Pi_0$. *Note, if the input to f is in an incorrect format, we can just generate a string that is not possibly in Π_0.*

We demonstrated that Π_1 is not recursive in the previous class. However, we based our belief that Π_0 is not recursive on an intuitive relationship between Π_1 and Π_0. Now, applying our theorem, we know

$$\Pi_1 \text{ is not recursive} \Rightarrow \Pi_0 \text{ is not recursive},$$

which is more formal than our previous argument.

4 Example 2

Let $L_2 = \{\#\alpha(G)\# \mid \lambda \in L(G)\}$. L_2 is the set of all grammars that generate the empty string. Clearly L_2 is recursively enumerable because we can enumerate the strings a grammar G generates until a λ is achieved.

To show L_2 is not recursive, consider Π_0. The Turing reduction $L_2 \leq_{\mathrm{T}} \Pi_0$ is easy to show. But this is a useless reduction because it only tells us that L_2 is recursively enumerable, which we already know.

We need to try $\Pi_0 \leq_{\mathrm{T}} L_2$ instead. Let us define $f : \Delta^* \rightarrow \Delta^*$ as

$$f(\#\alpha(G)\#\alpha(x)\#) = \#\alpha(G')\#.$$

So we want $x \in L(G) \Leftrightarrow \lambda \in L(G')$. Let G' enumerate $L(G)$ and check for x each time a new element ω is enumerated. If the check finds that $x = \omega$, then G' generates a λ. Otherwise $L(G') = \emptyset$. *Note, there is a G' for each (G, x) pair in Π_0 and $L(G')$ is either $\{\lambda\}$ or $\emptyset$.*

Now, applying our theorem, Π_0 not recursive $\Rightarrow L_2$ not recursive.

5 Exercise 1

Consider,

$$\begin{aligned} L_3 &= \{\#\alpha(G)\# \mid L(G) = \emptyset\} \\ \overline{L_3} &= \{\#\alpha(G)\# \mid L(G) \neq \emptyset\}. \end{aligned}$$

Answer the questions:

1. Is L_3 recursive?
2. Is L_3 recursively enumerable?
3. Is $\overline{L_3}$ recursively enumerable?

Clearly the answer to question 3 is yes. Simply enumerate all strings in Π_0 and for each $\#\alpha(G)\#\alpha(x)\# \in \Pi_0$, $\#\alpha(G)\# \in \overline{L_3}$.

To show that L_3 is not recursively enumerable, we considered two reductions. The first reduction considered was

$$\overline{\Pi_0} \leq_{\mathrm{T}} L_3.$$

If we define our f as in example 2, then we have shown that $\overline{\Pi_0} \leq_{\mathrm{T}} L_3$.

The second reduction considered was

$$\overline{L_2} \leq_{\mathrm{T}} L_3.$$

Define $g : \Delta^* \rightarrow \Delta^*$ to be

$$g(\#\alpha(G)\#) = \#\alpha(G')\#.$$

Let G' be a grammar that enumerates $L(G)$. If a λ is found, then G' generates a λ, else $L(G') = \emptyset$. So, $\lambda \notin L(G) \Leftrightarrow L(G') = \emptyset$.

Since $\overline{\Pi_0}$ and $\overline{L_2}$ are not recursively enumerable, L_3 is not recursively enumerable and L_3 is not recursive.

6 Exercise 2

Consider,

$$\begin{aligned} L_4 &= \{\#\alpha(G_1)\#\alpha(G_2)\# \mid L(G_1) = L(G_2)\} \\ \overline{L_4} &= \{\#\alpha(G_1)\#\alpha(G_2)\# \mid L(G_1) \neq L(G_2)\} \end{aligned}$$

Answer the questions:

1. Is L_4 recursive?
2. Is L_4 recursively enumerable?
3. Is $\overline{L_4}$ recursively enumerable?

We can answer questions 1 and 2 using a reduction,

$$\overline{L_2} \leq_{\mathrm{T}} L_4.$$

Let $f : \Delta^* \to \Delta^*$ be defined by

$$f(\#\alpha(G)\#) = \#\alpha(G_1)\#\alpha(G_2)\#.$$

Now if

$$\begin{aligned} L(G_1) &= \begin{cases} \{\lambda\} & \lambda \in L(G) \\ \emptyset & \lambda \notin L(G) \end{cases}, \\ L(G_2) &= \emptyset, \end{aligned}$$

then $\#\alpha(G)\# \in \overline{L_2} \Leftrightarrow L(G_1) = L(G_2)$. This implies that L_4 is not recursively enumerable, answering question 2, and since L_4 is not recursively enumerable, it is also not recursive, answering question 1.

Question 3 will be answered in the next session.