7 Equivalence Theorems and the Church-Turing Thesis

We are now going to show that the models of computability we discussed so far, that is, URM-computability, Turing-computability, partial recursiveness and lambda-definability are equivalent. These equivalences will support the thesis of Church and Turing that each of the four models captures exactly the intuitive notion of effective computability.

7.1 Equivalence Theorem

For a partial function \( f: \mathbb{N}^k \to \mathbb{N} \) the following statements are equivalent:

- \( f \) is URM-computable.
- \( f \) is Turing-computable.
- \( f \) is partial recursive.
- \( f \) is lambda-definable.

Proof. We will only give a sketch of the proof. in the previous chapter we already proved the equivalence of partial recursiveness and lambda-definability. For the remaining problems our strategy will be to prove first the equivalence of URM-computability and partial recursiveness, and then, in a similar way, the equivalence of Turing-computability and partial recursiveness.

1. URM-computability implies partial recursiveness.

Let \( f: \mathbb{N}^k \to \mathbb{N} \) be computed by the URM program \( P \), i.e. \( f = P^{(k)} \). Consider the following functions:

\[
c(x_1, \ldots, x_k, t) := \begin{cases} 
\text{contents of } R_1 \text{ after } t \text{ steps in the computation of } P^{(k)}(x_1, \ldots, x_k), & \text{if this computation has not already stopped;} \\
\text{the final contents of } R_1, & \text{if the computation of } P^{(k)}(x_1, \ldots, x_k) \\
\text{has stopped after fewer than } t \text{ steps.} & 
\end{cases}
\]

\[
j(x_1, \ldots, x_k, t) := \begin{cases} 
\text{number of the next instruction, when } t \text{ steps in the computation of } P^{(k)}(x_1, \ldots, x_k) \text{ have been performed,} & \\
\text{if } P^{(k)}(x_1, \ldots, x_k) \text{ has not stopped after } t \text{ steps or fewer;} & \\
0 & \text{if } P^{(k)}(x_1, \ldots, x_k) \text{ has stopped after } t \text{ steps or fewer;} 
\end{cases}
\]

The functions \( c \) and \( j \) can be used to compute \( f \) as follows:

Assume that \( f(x_1, \ldots, x_k) \) is defined. Then the computation of \( P^{(k)}(x_1, \ldots, x_k, t) \) stops after, say, \( t_0 \) steps. By definition of \( j \) and \( c \) this means that

\[
t_0 = \mu t. (j(x_1, \ldots, x_k, t) = 0)
\]

and
\[ f(x_1, \ldots, x_k) = c(x_1, \ldots, x_k, t_0) \]

If, on the other hand, \( f(x_1, \ldots, x_k) \) is undefined, then \( P^{(k)}(x_1, \ldots, x_k, t) \) never stops, and so \( j(x_1, \ldots, x_k, t) \) will be never zero. Thus \( \mu t.(j(x_1, \ldots, x_k, t) = 0) \) is undefined. Hence, in either case, we have

\[ f(x_1, \ldots, x_k) \simeq c(x_1, \ldots, x_k, \mu t.(j(x_1, \ldots, x_k, t) = 0)) \]

So, to show that \( f \) is partial recursive, it is sufficient to show that \( c \) and \( j \) are recursive functions. It is clear that \( c \) and \( j \) are computable in the informal sense. With some effort one can even show that \( c \) and \( j \) are primitive recursive.

2. Partial recursiveness implies URM-computability.

In order to show this implication one has to prove that the set of URM-computable partial functions contains the basic functions, zero, succ, \( \text{proj}^n_i \), and is closed under the operations of composition, primitive recursion, minimization.

For the basic functions this is easy.

In the case of composition one has to glue programs together, in a similar way as it was done in the proof of the undecidability of the Halting problem.

In order to compute \( f(x_1, \ldots, x_k, y) \), where \( f \) is defined from \( g \) and \( h \) by primitive recursion, one successively computes

\[
\begin{align*}
z_0 &= g(x_1, \ldots, x_k), \\
z_1 &= h(x_1, \ldots, x_k, 0, z_0), \\
z_2 &= h(x_1, \ldots, x_k, 1, z_1), \\
&\vdots \\
&\vdots \\
z_y &= h(x_1, \ldots, x_k, y - 1, z_{y-1})
\end{align*}
\]

Then \( f(x_1, \ldots, x_k, y) = z_y \). This process can be easily carried out on a URM.

Minimization can be dealt with similarly.

3. Turing-computability implies partial recursiveness.

This can be proven similarly to 1.

4. Partial recursiveness implies Turing-computability.

Similar to 2., but slightly harder, since with the more primitive Turing machines it is harder to establish the required closure properties.
7.2 The Church-Turing Thesis

The Church-Turing Thesis states that the three notions of computability, proven to be equivalent in the previous theorem, coincide with the informal notion of effective computability.

Although this thesis is not a mathematical theorem and hence cannot be proven, it is supported by strong empirical evidence. This evidence is provided, among others, by the following facts:

(a) The equivalence theorem above can be extended to a number of other models of computability, among them

- While programs,
- Symbol manipulation systems of Post and Markov,
- Equational calculi of Kleene and Gödel.
- Any of the current programming languages, such as Pascal, C, Prolog, e.t.c.

These equivalences hold effectively, that is, there are effective procedures that translate a program of one language computing a numerical function into a program of the other language computing the same function.

(b) So far, no function has been found that is intuitively computable, but not Turing-computable.

In the following we will adopt the Church-Turing Thesis. This will allow us to use informal arguments for the computability of a given function as a “proof” of its computability in a technical sense. Of course, in each case one could, with some effort, avoid the reference to the Church-Turing Thesis, by proving, say, partial recursiveness directly.

Note that, due to the extended effective equivalence theorem, (a), the halting problem for any of the above mentioned programming languages is undecidable. Because, if we had a decision procedure that decides whether or not a, say, Pascal program halts at a given input, we could, using a translation of Pascal programs into URM programs, easily decide the halting problem for URM programs, which is impossible.