Chapter 5

Computation-Theoretic Issues

This chapter concerns two subjects from the theory of computation, namely the halting problem and non-uniform computational complexity. Some issues concerning these subjects are investigated thinking in terms of instruction sequences.

Positioning Turing’s result regarding the undecidability of the halting problem as a result about programs rather than machines, and taking single-pass instruction sequences as considered in SPISA as programs, we analyse the autosolvability requirement that a program of a certain kind must solve the halting problem for all programs of that kind. We present positive and negative results concerning the autosolvability of the halting problem for programs.

Thinking in terms of a single-pass instruction sequence as considered in SPISA, we define counterparts of the classical non-uniform complexity classes \( P/poly \) and \( NP/poly \), introduce a notion of completeness for the counterpart of \( NP/poly \) using a non-uniform reducibility relation, formulate several complexity hypotheses, including a counterpart of the well-known complexity theoretic conjecture that \( NP \not\subseteq P/poly \), and show that a problem closely related to \( 3SAT \) is \( NP \)-complete as well as complete for the counterpart of \( NP/poly \).

5.1 Autosolvability of Halting Problem Instances

Turing’s result regarding the undecidability of the halting problem is a result about Turing machines. It says that there does not exist a single Turing machine that, given the description of an arbitrary Turing machine and input, will determine whether the computation of that Turing machine applied to that input eventually halts (see e.g. [Turing (1937)]). Implicit in this result is the autosolvability requirement that a machine of a certain kind must solve the halting problem for all machines of that kind. The halting problem is frequently
paraphrased as a result about programs as follows: the halting problem is the problem to
determine, given a program and an input to the program, whether execution of the pro-
gram on that input will eventually terminate. If we position Turing’s result regarding the
undecidability of the halting problem as a result about programs rather than machines, we
get the autosolvability requirement that a program of a certain kind must solve the halting
problem for all programs of that kind. In this section, we investigate this autosolvability
requirement in a setting in which programs take the form of instruction sequences.

5.1.1 Functional units relating to Turing machine tapes

First, we define some notions that have a bearing on the halting problem in the setting of
ISNR and functional units. The notions in question are defined in terms of functional units
for the following state space:

\[ T = \{ v^* w \mid v, w \in \{0, 1, :\}^* \} . \]

The elements of \( T \) can be understood as the possible contents of the tape of a Turing
machine whose tape alphabet is \( \{0, 1, :\} \), including the position of the tape head. Consider
an element \( v^* w \in T \). Then \( v \) corresponds to the content of the tape to the left of the
position of the tape head and \( w \) corresponds to the content of the tape from the position of
the tape head to the right — the indefinite numbers of padding blanks at both ends are left
out. The colon serves as a separator of bit sequences. This is for instance useful if the input
of a program consists of another program and an input to the latter program, both encoded
as a bit sequence. We could have taken any other tape alphabet whose cardinality is greater
than one, but \( \{0, 1, :\} \) is extremely handy when dealing with issues relating to the halting
problem.

Below, we will use a computable injective function \( \alpha : T \to \mathbb{N} \) to encode the members of
\( T \) as natural numbers. Because \( T \) is a countably infinite set, we assume that it is understood
what is a computable function from \( T \) to \( \mathbb{N} \). An obvious instance of a computable injective
function \( \alpha : T \to \mathbb{N} \) is the one where \( \alpha(a_1 \ldots a_n) \) is the natural number represented in
the quinary number-system by \( a_1 \ldots a_n \) if the symbols 0, 1, : and \( \hat{\cdot} \) are taken as digits
representing the numbers 1, 2, 3 and 4, respectively.

**Definition 5.1.** A method operation \( M \in \mathcal{MO}(T) \) is **computable** if there exist computable
functions \( F, G : \mathbb{N} \to \mathbb{N} \) such that \( M(v) = (\beta(F(\alpha(v))), \alpha^{-1}(G(\alpha(v)))) \) for all \( v \in T, \)
where \( \alpha : T \to \mathbb{N} \) is a computable injective function and \( \beta : \mathbb{N} \to \mathbb{B} \) is inductively defined
by \( \beta(0) = t \) and \( \beta(n + 1) = f. \) A functional unit \( U \in \mathcal{FU}(T) \) is **computable** if, for each