第六章

6.1

The following program is in LISP:

```
((LAMBDA (X) (LIST X (LIST (QUOTE QUOTE) X)))
(QUOTE (LAMBDA (X) (LIST X (LIST (QUOTE QUOTE) X)))))
```

6.2

$MIN_{TM} = \{< M > | M$ is a minimal TM$\}$. We can prove that any infinite subset of MIN_{TM} is not Turing-recognizable by contradiction. Here we assume that there exists S, an infinite subset of MIN_{TM}, such that S is Turing-recognizable; therefore, we have a TM E which can enumerate S. By using this TM E, we construct another TM N as follows.
$C =$ "On input w:

1. Obtain, via the recursive theorem, own description $< C >$.
2. Run the enumerator E until a machine D appears with a longer description than that of C.
3. Simulate D on input w."

Since MIN_{TM} is infinite, E's list must contain a TM with longer description than C's description. Therefore, step 2 of C eventually terminates with some TM D that is longer than C. Then C simulates D and so is equivalent to it. Because C is shorter than D and is equivalent to D, D cannot be minimal. But D appears on the list that E produces. Thus we have a contradiction.

6.3 Say that M_1^B decides A and M_2^C decides B. Use an oracle TM M_3, where M_3^C decides A. Machine M_3 simulates M_1. Every time M_1 queries its oracle about some string x, machine M_3 tests whether $x \in B$ and provides the answer to M_1. Because machine M_3 doesn't have an oracle for B and cannot perform that test directly, it simulates M_2 on input x to obtain that information. Machine M_3 can obtain the answer to M_2's queries directly because these two machines use the same oracle, C.

6.4

$A_{TM'} = \{< M, w > | M$ is an oracle TM and $M^{A_{TM}}$ accepts $w\}$. Assume that $A_{TM'}$ is decidable relative to A_{TM}; therefore, there exists an oracle TM T with oracle access to A_{TM} deciding $A_{TM'}$. We construct another oracle TM D as follows:
D="On input $< M >$

1. Run $T^{A_{TM}}$ on input $< M, < M >>$.
2. If T accepts, *reject*. If T rejects, *accept*.

$D^{A_{TM}}$ accepts $< M >$ if and only if M rejects $< M >$. When the input of D is $< D >$, we have $D^{A_{TM}}$ accepts $< D >$ if and only if D rejects $< D >$. It makes a contradiction; thus, $A_{TM'}$ is undecidable relative to A_{TM}.

6.5 The statement $\exists x \forall y \left[x+y=y \right]$ is a member of $\text{Th}(\mathcal{N}, +)$ because that statement is true for the standard interpretation of $+$ over the universe $\mathcal{N}$. Recall that we use $\mathcal{N} = \{0, 1, 2, \ldots\}$ in this chapter and so we may use $x = 0$. The statement $\exists x \forall y \left[x+y=x \right]$ is not a member of $\text{Th}(\mathcal{N}, +)$ because that statement isn't true in this model. For any value of x, setting $y = 1$ causes $x+y=x$ to fail.

Hence $M = P_{<N>}$, a TM prints $< N >$ and then halts, prints $< N >$ and N prints $< M >$. These two TMs are not equal because q outputs a string that is different from its input.

6.7

- Let the fixed point machine be $\langle M \rangle$. If M halts on any input w, then clearly it cannot be a fixed point, since the language M recognizes and the language $t(\langle M \rangle)$ recognizes must differ in w.
- However, suppose M loops infinitely on all inputs. Then $t(\langle M \rangle)$ must also loop infinitely, because it is the same machine but with accept and reject states swapped. Therefore M is a fixed-point. So any machine which loops infinitely on all inputs (for example, a machine which can never leave the start state) is a fixed-point.

6.8

Undecidable so one of them unrecognizable cannot mapping reduce.

6.9 Assume for the sake of contradiction that some TM X decides a property P, and P satisfies the conditions of Rice's theorem. One of these conditions says that TMs A and B exist where $\langle A \rangle \in P$ and $\langle B \rangle \notin P$. Use A and B to construct TM R:

```
R = "On input w:
        1. Obtain own description ⟨R⟩ using the recursion theorem.
        2. Run X on ⟨R⟩.
        3. If X accepts ⟨R⟩, simulate B on w.
           If X rejects ⟨R⟩, simulate A on w."
```

If $\langle R \rangle \in P$, then X accepts $\langle R \rangle$ and $L(R) = L(B)$. But $\langle B \rangle \notin P$, contradicting $\langle R \rangle \in P$, because P agrees on TMs that have the same language. We arrive at a similar contradiction if $\langle R \rangle \notin P$. Therefore, our original assumption is false. Every property satisfying the conditions of Rice's theorem is undecidable.

6.10 The statement ϕ_{eq} gives the three conditions of an equivalence relation. A model (A, R_1), where A is any universe and R_1 is any equivalence relation over A, is a model of ϕ_{eq}. For example, let A be the integers $\mathcal{Z}$ and let $R_1 = \{(i, i) | \, i \in \mathcal{Z}\}$.

6.12 Reduce Th($\mathcal{N}$, <) to Th($\mathcal{N}$, +), which we've already shown to be decidable. Show how to convert a sentence ϕ_1 over the language of ($\mathcal{N}$, <) to a sentence ϕ_2 over the language of ($\mathcal{N}$, +) while preserving truth or falsity in the respective models. Replace every occurrence of $i < j$ in ϕ_1 with the formula $\exists k\,[\,(i+k=j) \wedge (k+k \neq k)\,]$ in ϕ_2, where k is a different new variable each time.

Sentence ϕ_2 is equivalent to ϕ_1 because "i is less than j" means that we can add a nonzero value to i and obtain j. Putting ϕ_2 into prenex-normal form, as required by the algorithm for deciding Th($\mathcal{N}$, +), requires a bit of additional work. The new existential quantifiers are brought to the front of the sentence. To do so, these quantifiers must pass through Boolean operations that appear in the sentence. Quantifiers can be brought through the operations of $\wedge$ and $\vee$ without change. Passing through $\neg$ changes $\exists$ to $\forall$ and vice versa. Thus, $\neg\exists k\,\psi$ becomes the equivalent expression $\forall k\,\neg\psi$, and $\neg\forall k\,\psi$ becomes $\exists k\,\neg\psi$.

6.13)

Following is a proof that for each m, the theory $Th(F_m)$, where $F_m = (Z_m, +, \times)$ is a model over the group $Z_m = \{0,1,2,\dots,m-1\}$ and the relations $+,\times$ computed modulo m, is decidable:

Denote a simple addition or multiplication modulo m, consisting of vectors of size 3 where the first and second rows are the arguments and the third row is the result. Without loss of generality, we will look at $+$. For any given m, there are m^2 true additions, i.e. combinations that denote a correct addition modulo m. We can check for a given string over $\Sigma_3 = \left\{\begin{bmatrix}0\\0\\0\end{bmatrix},\dots,\begin{bmatrix}1\\1\\1\end{bmatrix}\right\}$ that is represents a true addition modulo m as follows:

- Make sure all rows represent numbers in $\{0,1,\dots,m-1\}$.
- If yes, check whether the third row fits the result expected for the first two rows (out of the m^2 options).
- If yes, it is a true addition modulo m, otherwise it is not.

Similar method can be applied on multiplication modulo m.

Next, for any given formula of the form $\varphi = Q_1x_1 \dots Q_lx_l[\psi(x_1,\dots,x_l)]$, where Q_i are quantifiers, x_i are variables and ψ is a quantifier-free formula with the $+$ and $\times$ modulo m relations (and the standard operators, e.g. $\neg$), we check all possible assignments of all x_i for the corresponding quantifier Q_i as follows:

Denote $I_l(x_1,\dots,x_l) = \psi(x_1,\dots,x_l)$. For any $i > 0$:

- If $Q_i = \exists$: $I_{i-1}(x_1,\dots,x_{i-1}) = \bigvee_{k=0}^{m-1} I_i(x_1,\dots,x_{i-1},k)$

6.14)

Following is a proof that for any two languages A, B a language J exists such that $A \leq_T J$ and $B \leq_T J$:

Let $J = \{\bar{0}a \mid a \in A\} \cup \{\bar{1}b \mid b \in B\}$, where $\bar{0}, \bar{1}$ are symbols that do not appear in any $w \in A \cup B$. We can then define a TM M that is decidable relative to J as follows:

"For input w:

- Check with the oracle of J whether $\bar{0}w \in J$.
- If it accepts, *accept*. Otherwise, *reject*."

Clearly $w \in A \Leftrightarrow \bar{0}w \in J$. In a similar manner we can construct a TM that decides B by mapping w to $\bar{1}w$ and querying the oracle of J. Therefore both $A \leq_T J$ and $B \leq_T J$, as required.

6.15

We want to show that for any language A, there exists a language B such that $A \leq_T B$ and $B \not\leq_T A$. For a language B, let Red_B denote the set of all languages that are Turing reducible to B.

(a) Show that $Red_\emptyset$ is the set of Turing decidable languages.

(b) Show that $A^B_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM with oracle } B \text{ and } M \text{ accepts } w\}$ is undecidable relative to B.

(c) Conclude that that for any language A there exists a language B such that $A \leq_T B$ and $B \not\leq_T A$.

6.16

Theorem 1.1. *There exist computably enumerable languages A and B that are Turing incomparable.*

Proof. We present an enumeration procedure that enumerates A and B simultaneously, thereby showing that both are computably enumerable. The enumeration proceeds in stages: stage 0, stage 1, stage 2, etc. Each stage is subdivided into two phases, an A phase and a B phase. In an A phase, an element of A (and in a B phase, an element of B) can potentially be enumerated; that is, in each phase one element may be put into the appropriate language. In addition, we maintain during the construction two lists–L_A and L_B– the entries of which are all of the form (i, x). In these pairs i is the index of a Turing machine M_i, and x is an input for which we will try to guarantee that $x \in A \Leftrightarrow x \in W_i^B$ ($x \in B \Leftrightarrow x \in W_i^A$, respectively). Each entry in these lists is either 'active' or 'inactive'. Furthermore, x_A (x_B) will always be an input that is so large that it does not affect any of the preceding decisions.

Step 0 is used for initialization.

```
    *Step 0.* (Phase $A$ and $B$)
    Let $A = B = L_A = L_B = \emptyset$ and $x_A = x_B = 0$
```

The actual construction then proceeds as follows:

```
• *Step* $n+1$. (Phase $A$)

    $L_A := L_A \cup \{(n, x_A)\}$;  ($(n, x_A)$ is 'active').
    FOR all active $(i, x) \in L_A$ in increasing order according to $i$ DO
        IF $M_i$ on input $x$ and Oracle $B$ (as constructed so far)
            accepts in $n$ or fewer steps
            THEN $A := A \cup \{x\}$;  Declare $(i, x)$ to be 'inactive';
                    ⎧ Let $y$ be the largest oracle query in
                    ⎪                         the computation above;
                    ⎪ $x_B := \max(x_B, y+1)$;
                    ⎪ $j := 0$;
                (*) ⎨ FOR $(k,y) \in L_B$, $k \geq i$ DO
                    ⎪     $L_B := L_B - \{(k,y)\} \cup \{(k, x_B + j)\}$;  (*active*)
                    ⎪     $j := j+1$;
                    ⎪ END;
                    ⎩ $x_B := x_B + j$;
                   GOTO Phase $B$;
        END;
    END;
    GOTO Phase $B$;

• *Step* $n+1$. (Phase $B$)

    $L_B := L_B \cup \{(n, x_B)\}$;  (*active*)
    FOR all active $(i, x) \in L_B$ in increasing order according to $i$ DO
        IF $M_i$ on input $x$ and Oracle $A$ (as constructed so far)
            accepts in at most $n$ steps
            THEN $B := B \cup \{x\}$;  Declare $(i, x)$ to be 'inactive';
```

```
                    ⎧ Let $y$ be the largest oracle query in
                    ⎪                       the computation above;
                    ⎪ $x_A := \max(x_A, y+1)$;
                    ⎪ $j := 0$;
                (*) ⎨ FOR $(k,y) \in L_A,\ k > i$ DO
                    ⎪    $L_A := L_A - \{(k,y)\} \cup \{(k, x_A + j)\}$;  (*active*)
                    ⎪    $j := j + 1$;
                    ⎪ END;
                    ⎩ $x_A := x_A + j$;
                   GOTO Step $n+2$;
      END;
   END;
   GOTO Step $n+2$;
```

We claim that the languages A and B that are "enumerated" by the preceding construction have the desired properties, namely that $A \not\leq_T B$ and $B \not\leq_T A$, or more precisely that for all $i \in I\!N$ there exist x and x' with $x \in A \Leftrightarrow x \in W_i^B$ and $x' \in B \Leftrightarrow x' \in W_i^A$.

Notice that the entries (i,x) in the lists L_A and L_B can change and that such changes respect the following priority ordering: The entry $(0,\ldots)$ in List L_A has highest priority, then $(0,\ldots)$ in list L_B, then $(1,\ldots)$ in list L_A, then $(1,\ldots)$ in list L_B, etc. This means that if at stage n of the construction $(i,x) \in L_A$ and M_i accepts x with oracle B in at most n steps, then x is enumerated into A. To prevent subsequent changes to the oracle B that might alter this computation, all of the entries $(j,\ldots)$ in the list L_B that have lower priority than (i,x) are removed from the list and replaced by new entries which are "large enough" that they do not affect the computation. On the other hand, it can happen that a requirement "$x \in A \Leftrightarrow x \in W_i^B$" that at some point in the construction appears satisfied is later "injured." This happens if some x' is enumerated into B for the sake of some entry (i',x') of higher priority (i.e., for which $i' < i$). It is precisely the determination of these priorities that is carried out in section $(*)$ of the construction.

What is important is that for each i the entry $(i,\ldots)$ in list L_A or L_B is changed only finitely many times ("finite injury"). This can be proven by induction on i.

6.17

5. Let A and B be two disjoint languages over a common alphabet Σ. Say that language C ***separates*** A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Show that if A and B are any two disjoint co-Turing-recognizable languages, then there exists a decidable language C that separates A and B. (A language L is co-Turing-recognizable if its complement $\overline{L}$ is Turing-recognizable.)

   Answer: Suppose that A and B are disjoint co-Turing-recognizable languages. We now prove that there exists a decidable language C that separates A and B. Since A is co-Turing-recognizable, its complement $\overline{A}$ must have an enumerator $E_{\overline{A}}$. Similarly, the fact that B is co-Turing-recognizable implies $\overline{B}$ has an enumerator $E_{\overline{B}}$. Since A and B are disjoint, i.e., $A \cap B = \emptyset$, we have that $\overline{A} \cup \overline{B} = \Sigma^*$ by DeMorgan's law. Thus, every string in Σ^* is in the union of $\overline{A}$ and $\overline{B}$. Furthermore, since A and B are disjoint, every string in B is in $\overline{A}$, and every string in A is in $\overline{B}$.

   Using these facts, we construct a Turing machine M as follows:

```
M = "On input $w$, where $w \in \Sigma^*$:
      1. Run $E_{\overline{B}}$ and $E_{\overline{A}}$ in parallel.
      2. Alternating between the enumerators, and starting with $E_{\overline{B}}$,
         compare the outputs of each of the enumerators, one string
         at a time, to the input $w$.
      3. If some output of $E_{\overline{B}}$ matches $w$, accept.
         If some output of $E_{\overline{A}}$ matches $w$, reject."
```

Let C be the language recognized by TM M. Since $\overline{A} \cup \overline{B} = \Sigma^*$, every string is enumerated by $E_{\overline{A}}$ or $E_{\overline{B}}$ (or both). Hence, M will halt on all inputs, so M is a decider for language C.

We now need to show that C separates A and B. Since every string in A is in $\overline{B}$, the output of $E_{\overline{B}}$ contains all strings of A. Thus, M accepts all strings that are output by only $E_{\overline{B}}$, so M accepts all strings of A since $E_{\overline{A}}$ never outputs any strings in A. Likewise, since every string in B is in $\overline{A}$, the output of $E_{\overline{A}}$ contains all strings of B. But M rejects all strings that are output by only $E_{\overline{A}}$, so M rejects all strings in B since $E_{\overline{B}}$ never outputs strings from B. Thus, M accepts all strings in A and rejects all strings in B, so its language C separates A and B.

Note that we did not prove which set C of strings M accepted. The particular language of C depends on the order of the outputs of the enumerators. However, the only strings in question are the strings that are in $\overline{A} \cap \overline{B}$. Whether these strings are in C or in $\overline{C}$ is not relevant to the question of separating A and B.

6.19

Following is a proof that there exist languages that are not recognizable by an oracle Turing machine with oracle for A_{TM}: The proof follows corollary 4.18 (page 178). The set of legal encodings of Turing machines with oracle access to A_{TM} is countable (the same argument as used for proving the set of all standard Turing machines is countable). However the set of all languages is uncountable: any language A can be mapped to an infinite binary vector (sequence of 0's and 1's) corresponding to its characteristic vector χ_A, as described in the book. Therefore the set of all languages cannot be put into a correspondence with the set of all Turing machines with oracle access to A_{TM}, therefore some languages exist that are not recognizable by any Turing machine with oracle access to A_{TM}.

6.20

We consider the following TM M:
M ="On input P, an instance of the PCP:

1. For each possible finite sequence of dominos of P:
2. *Accept* if the sequence forms a match. If not, continue."

To determine whether $P \in PCP$, we test whether $< M, P > \in A_{TM}$. If so, a match exists, so $P \in PCP$. Otherwise no match exists, so $P \notin PCP$.

6.21

Solution 1:

SOLUTION:Here's an algorithm for computing $K(x)$. On input x,

1. Go through all binary strings s in lexicographic order, and for each such s, parse s as $\langle M, w\rangle$ for some TM M and input w. If s fails to parse, move to the next such s.
2. Modify the machine M so that all transitions to the reject state go to the accept state. Call the modified machine M'. Now, M halts on w if and only if M' accepts w.
3. Next, query A_{TM} on input $\langle M', w\rangle$. If A_{TM} accepts $\langle M', w\rangle$, simulate M on w (this will halt), and check whether M on input w halts with x on its tape. Output $|s|$.

Since we are going through the strings in lexicographic order, we will output the length of the shortest description (and the lexicographically first one if there is a tie).

Solution 2:

Given a string x, we can simply start testing all strings s up to length |x| + c (where c is the size of a TM that halts immediately upon starting) as potential descriptions of x. If s is well-formed as <M>w, then we simulate M with input w and see if it halts with x on the tape. The problem, of course, is that we don't know if M will halt on input w; but with an oracle for A_TM we can determine this. If M doesn't halt we move on to the next string s, and so on.

6.22

Create an oracle TM that does the following: Given a number n, begin enumerating all strings x of length n. For each x compute K(x) using the algorithm from 6.13. As soon as we find an x with K(x) >= |x|, output x and halt.

6.23

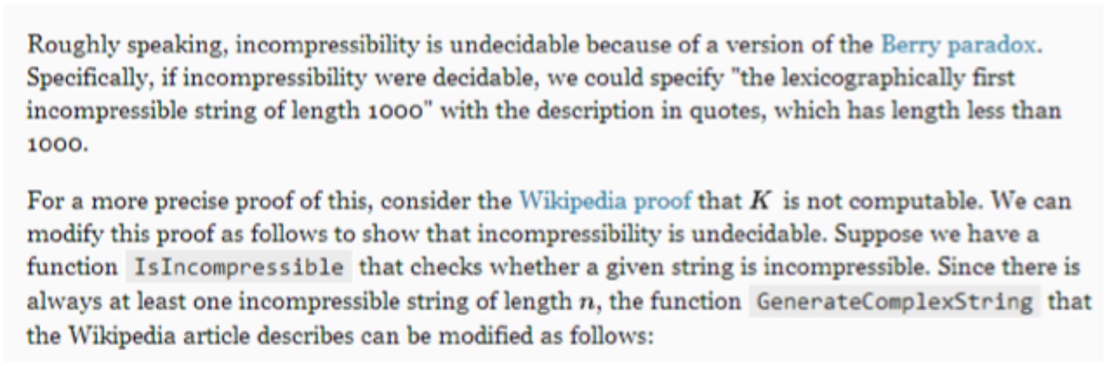
Suppose on the contrary that Turing machine M computes K(x). Define a Turing machine N which computes the 1st incompressible string of length n:
Let N = "On input n (an integer in binary notation),
1. Enumerate strings s of length n in lexicographical order.
2. For each s, call M to compute K(s).
3. If K(s) >= |s|, output s and halt."
Now the length of <N>n is |<N>| + log(n) = log(n) + c. [Note that in <N>n, n is a binary string, wherease the n in "log(n)" refers to the integer defined by the string n.] By choosing n sufficiently large we have |s| = n > log(n) + c = |<N>n|, which contradicts the incompressibility of s. Therefore machine M cannot exist.

6.24

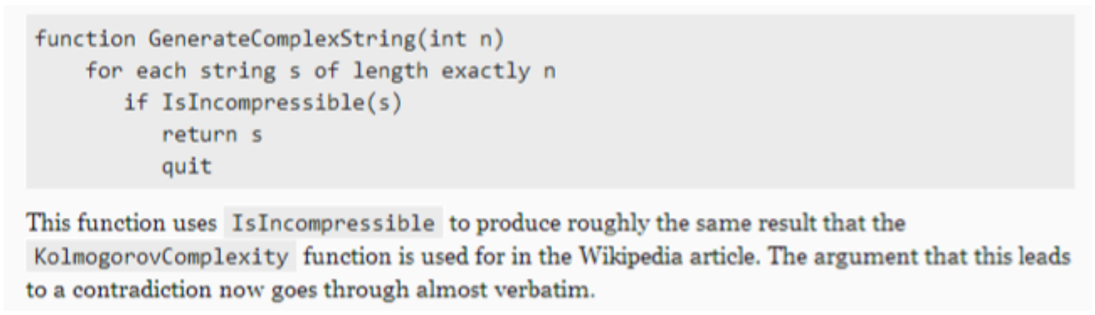
Solution 1:

In the proof of 6.15, we used the calculation of K(s) only to determine whether a string was incompressible. So if we assume a machine M exists which decides if a string is incompressible, we can use it in a similar construction to that used for 6.15.
Let N = "On input n (an integer in binary notation),
1. Enumerate strings s of length n in lexicographical order.
2. For each s, compute <M,s>
3. If M accepts s, output s and halt."

Solution 2:


Roughly speaking, incompressibility is undecidable because of a version of the Berry paradox. Specifically, if incompressibility were decidable, we could specify "the lexicographically first incompressible string of length 1000" with the description in quotes, which has length less than 1000.

For a more precise proof of this, consider the Wikipedia proof that K is not computable. We can modify this proof as follows to show that incompressibility is undecidable. Suppose we have a function `IsIncompressible` that checks whether a given string is incompressible. Since there is always at least one incompressible string of length n, the function `GenerateComplexString` that the Wikipedia article describes can be modified as follows:



```
function GenerateComplexString(int n)
    for each string s of length exactly n
       if IsIncompressible(s)
          return s
          quit
```

This function uses `IsIncompressible` to produce roughly the same result that the `KolmogorovComplexity` function is used for in the Wikipedia article. The argument that this leads to a contradiction now goes through almost verbatim.

6.25

With just a little bit of modification we can use the same argument as in 6.15 and 6.16. Suppose M enumerates an infinite set of incompressible strings. Define a Turing machine N which computes an incompressible string of length at least n:
Let N = "On input n (an integer in binary notation),
1. Call M to enumerate incompressible strings s.
2. If |s| >= n, output s and halt."

6.27

Solution Outline: For this problem, we assume that a TM can recognize its own code[1]. We then show that $A_{TM} \leq_m S$ and $A_{TM} \leq_m \overline{S}$, which also imply $\overline{A_{TM}} \leq_m \overline{S}$ and $\overline{A_{TM}} \leq_m S$ respectively.

We first give the reduction from A_{TM} to S. Given an instance $\langle M, w\rangle$ of A_{TM}, we construct a machine M' which given an input x, rejects if $x \neq \langle M'\rangle$ and simulates M on w if $x = \langle M'\rangle$. Thus, $L(M') = \{\langle M'\rangle\}$ if M accepts w and $\emptyset$ otherwise. Similarly, for the reduction from A_{TM} to $\overline{S}$, we make M' accept if $x = \langle M'\rangle$ and simulate M on x otherwise. In this case, it gives $L(M') = \Sigma^*$ if M accepts w and $\{\langle M'\rangle\}$ otherwise.

6.26

Not exactly an answer:

> As a hint, what's wrong with this argument?
>
> Given a TM for x and y, we can construct a TM for xy in size
> K(x)+K(y)+O(1) where the constant depends on the details of the
> encoding and the chosen type of TM.
>
> Given TM X with input p which produces x and TM Y with input q which
> produces y, produce a TM which schedules X and then Y and give it
the
> input pq.
>
> So for c bigger than the size of the scheduling TM, we get K(xy) <=
> K(x) + K(y) + c.

I'm supposing that what you have in mind is that we would use the scheduling TM M to describe the string xy as <M><X>p<Y>q (where <X>p is a minimal description of x and <Y>q is a minimal description of y. The difficulty is that we can't in general distinguish where "<X>p" ends and "<Y>q" begins. So the problem would reduce to showing that there is no "constant sized" way of delimiting two arbitary strings. But even if we show this, it only means that this particular technique can't be used to describe the string xy. There could be some other description of xy that has no relationship to the descriptions of x and y separately, but still satisfies K(xy) <= K(x) + K(y) + c. To prove that this can't be done, I think we need to use a more general approach.

6．28

Show that in the structure $(\mathbf{N}; +)$ the following relations are definable:

(a) Ordering, $\{\langle m, n\rangle \,|m < n\}$

Answer: $\exists x\,(\neg(x + x = x) \wedge m + x = n)$

(b) Zero, $\{0\}$

Answer: $x + x = x$

(c) Successor, $\{\langle m, n\rangle \,|n = S(m)\}$

Answer:

We can encode successor using ordering and zero as follows:

$\exists u\,(\forall x\,(x < u \rightarrow x = 0) \wedge m + u = n)$.

Expanding this to use only $+$, we get:

$\exists u\,\exists z\,(z + z = z \wedge \forall x\,((\exists y\,(\neg(y = z) \wedge x + y = u)) \rightarrow x = z) \wedge m + u = n)$