1 Definition of a Markov algorithm

A Markov Algorithm on an alphabet \( A \) is a finite sequence \( \langle I_1, \ldots, I_k \rangle \) of pairs of words in \( A^* \), \( I_i = (x_i, y_i) \) for \( 1 \leq i \leq k \). A Markov algorithm proceeds as follows: Given an initial string \( w \in A^* \), we find the first \( i \) such that \( x_i \) occurs in \( w \) and replace the right-most occurrence by \( y_i \). Formally, if \( w = ux_iv \) is such that whenever \( x_i v = u'x_i v' \) we must have \( u' = \emptyset \), then we replace \( w \) by \( uy_i v \). We then replace \( w \) by this new word and repeat this process as long as some \( x_i \) occurs in \( w \), halting if and only if we reach a word with no such occurrences.

**Definition 1** Let \( A \supseteq \{\ast, B, 1\} \). A function \( f : \mathbb{N}^n \rightarrow \mathbb{N} \) is computed by a Markov algorithm if, for all \( x_1, \ldots, x_n \in \mathbb{N}^n \):

1. \( f(x_1, \ldots, x_n) \downarrow \) if and only if the algorithm terminates when started with the string
   \[ \ast 1^{x_1+1}B1^{x_2+1}B \ldots B1^{x_n+1} \]
2. if \( f(x_1, \ldots, x_n) \downarrow \), then the algorithm halts with the output string \( 1^{f(x_1, \ldots, x_n)} \).

Functions on larger alphabets can be defined similarly.

A function is Markov Algorithm-Computable (MA-computable) if there is a Markov algorithm which computes it.

Example: The function \( f(x) = x - 1 \) is computed by the algorithm:

\[
I_1 = (\ast 11, \emptyset) \\
I_2 = (\ast 1, \emptyset)
\]

2 MA-computability

**Proposition 2** Every TM-computable function is MA-computable.

**Proof:** We use a Markov algorithm to simulate the running of the Turing machine \( M = (A, Q, I) \). We use the alphabet \( B = \{\ast, \bullet\} \cup A \cup Q \) (assuming these are disjoint). The idea is to have our string represent the tape description, the state, and the head position at a given time, with instructions simulating one step of the machine. We will assume that \( M \) follows the same input-output conventions as above, but that the Markov algorithm uses \( \bullet \) instead of \( \ast \), so that the initial tape descriptions will be \( \ast w \) for some \( w \in A^* \) and the initial word for the algorithm will be \( \bullet w \). Our word will have the form \( \ast w_1 q w_2 \) where \( w_1, w_2 \in A^* \) and \( q \in Q \) to indicate that the machine is in state \( q \), the tape contains \( \ast w_1 w_2 \) and the head is reading the rightmost character of \( \ast w_1 \). The instructions are as follows:
1. \((\bullet, *q_0)\)

2. For each \((q_i, s_i, q_j, s_j, L) \in M\) we include \((s_i q_i, q_j s_j)\).

3. For each \((q_i, s_i, q_j, s_j, R) \in M\) and \(a \in A\) we include \((s_i q_i a, s_j a q_j)\).

4. For each \((q_i, s_i, q_j, s_j, 0) \in M\) we include \((s_i q_i, s_j q_j)\).

5. \((*, \emptyset)\).

Note that except for the first and last instruction, the order is irrelevant; precisely one of the given \(x_i\)'s occurs, unless the machine has reached the halt state in which case none of them do. The intermediate instructions then modify one TM state description to the successive state description.

**Proposition 3** Every MA-computable function is While-computable.

The idea is to use the fact that the occurrence and replacement functions are all primitive recursive, hence While-computable. Then so too is the function which finds the least \(i\) such that \(x_i\) occurs in \(w\) and replaces it by \(y_i\), as this can be built from cases. Then we need a single while loop which repeats as long as some \(x_i\) occurs in \(w\).

**Corollary 4** The classes of While-computable, TM-computable and MA-computable partial functions are all the same.