APPENDIX A

A.1. Efficient Computation of Powers Modulo m

We illustrate an efficient method of computing powers modulo m with an example. Assume that we want to compute $3^{547} \bmod 10$. First write 547 in base 2: 1000100011, hence $547 = 2^9 + 2^5 + 2 + 1 = ((2^4+1)\,2^4+1)\,2+1$, so: $3^{547} = ((3^{2^4} \cdot 3)^{2^4} \cdot 3)^2 \cdot 3$. Next we compute the expression beginning with the inner parenthesis, and reducing modulo 10 at each step: $3^2 = 9 \pmod{10}$, $3^{2^2} = 9^2 = 81 = 1 \pmod{10}$, $3^{2^3} = 1^2 = 1 \pmod{10}$, $3^{2^4} = 1^2 = 1 \pmod{10}$, $3^{2^4} \cdot 3 = 1 \cdot 3 = 3 \pmod{10}$, etc. At the end we find $3^{547} = 7 \pmod{10}$.

The algorithm in pseudocode would be like this:

```
procedure pow_mod(a,x,m) {computes a^x mod m}
  p := 1
  bx := binary_array(x) {x as a binary array}
  t := a mod m
  for k := 1 to length(bx)
    begin
      p := (p * p) mod m
      if bx[k] = 1 then
        {if k-th binary digit of x is 1}
        p := (p * t) mod m
    end
  return p
end pow_mod
```

The following is a program in C implementing the algorithm:

```
int pow(int a, int x, int m) {
  int p = 1;
  int y = (1 << (8 * size of(int) - 2));

  a %= m;

  while (!(y & x)) y >>= 1;

  while (y) {
    p *= p;
    p %= m;
    if (x & y) {
      p *= a;
      p %= m;
    }
    y >>= 1;
  }
  return p;
}
```

The following is an alternative algorithm equivalent to running through the binary representation of the exponent from right to left instead of left to right:

```
procedure pow_mod(a,x,m) {computes a^x mod m}
  p := 1
  t := a mod m
  while x > 0
    begin
      if x is odd then
        p := (p * t) mod m
      t := (t * t) mod m
      x := floor(x/2)
    end
  return p
end pow_mod
```

A.2. Machines and Languages

A.2.1. Turing Machines. A *Turing machine* is a theoretical device intended to define rigorously the concept of *algorithm.* It consists of

1. An *infinite tape* made of a sequence of cells. Each cell may be empty or may contain a symbol from a given alphabet.
2. A *control unit* containing a finite set of instructions.
3. A *tape head* able to read and write (or delete) symbols from the tape.

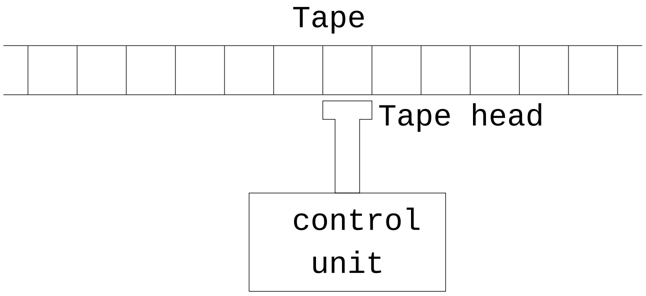

FIGURE A.1. Turing Machine.

Each machine instruction contains the following five parts:

1. The current machine state.
2. A tape symbol read from the current tape cell.
3. A tape symbol to write into the current tape cell.
4. A direction for the tape head to move: L = 'move one cell to the left', R = 'move one cell to the right', S = 'stay in the current cell'.
5. The next machine state.

Turing machines are generalizations of finite-state automata. A finite-state automaton is just a Turing machine whose tape head moves always from left to right and never writes to the tape. The input of the finite-state automaton is presented as symbols written in the tape.

In general we make the following assumptions:

1. An input is represented on the tape by placing the letters of the strings in contiguous tape cells. All other cells contain the blank symbol, which we may denote λ.

2. The tape is initially positioned at the leftmost cell of the input string unless specified otherwise.
3. There is one *starting state*.
4. There is one *halt state*, which we denote by "Halt".

The execution of a Turing machine stops when it enters the Halt state or when it enters a state for which there is no valid move. The output of the Turing machine is the contents of the tape when the machine stops.

We say that an input string is *accepted* by a Turing machine if the machine enters the Halt state. Otherwise the string is *rejected*. This can happen in two ways: by entering a state other than the Halt state from which there is no move, or by running forever (for instance executing an infinite loop).

If a Turing machine has at least two instructions with the same state and input letter, then the machine is *nondeterministic*. Otherwise it is *deterministic*.

Finite-State Automata. A finite-state automata can be interpreted as a Turing machine whose tape head moves only from left to right and never writes to the tape.

Pushdown Automata. A *pushdown automaton* is finite-state automaton with a stack, i.e., a storage structure in which symbols can be put and extracted from it by two operations: *push* (place on the top of the stack) and *pop* (take from the top of the stack)—consequently the last symbol put into the stack is the first symbol taken out. Additionally there is a third operation, *nop*, that leaves the stack intact. The next state function takes into account not only the current state and the symbol read from the input, but also the symbol at the top of the stack. After reading the next input symbol and the symbol at the top of the stack, the automaton executes a stack operation and goes to the next state. Initially there is a single symbol in the stack.

Linearly Bounded Automata. A *linearly bounded automaton* is a Turing machine whose tape is limited to the size of its input string plus two boundary cells that may not be changed.

Computable Functions. Consider a Turing machine T working on symbols from an alphabet of only one symbol $A = \{|\}$ ("stroke"). Let $f : \mathbb{N} \to \mathbb{N}$ the function defined so that $f(n) = m$ means that if the initial input of T consists of a string of $n+1$ strokes, the output of T is a string of $m+1$ strokes. We say that f is *computed* by the Turing machine T. A *computable function* is a function computed by some Turing machine. A computable function $f(n)$ *halts* for a given value of its argument n if T with input $n+1$ strokes halts. A computable function f is *total* if $f(n)$ halts for every n.

An *effective* enumeration of a set is a listing of its elements by an algorithm.

A.2.2. Hierarchy of Languages. Here we mention a hierarchy of languages that includes (and extends) Chomsky's classification, in increasing order of inclusion.

1. *Regular languages.* They are recognized by finite-state automata. *Example*: $\{a^m b^n \mid m, n = 1, 2, 3 \dots\}$.
2. *Deterministic context-free languages*, recognized by deterministic pushdown automata. *Example*: $\{a^n b^n \mid n = 1, 2, 3 \dots\}$.
3. *Context-free languages*, recognized by nondeterministic pushdown automata. *Example*: palindromes over $\{a, b\}$.
4. *Context-sensitive languages*, languages without λ recognized by linearly bounded automata. *Example*: $\{a^n b^n c^n \mid n = 1, 2, 3 \dots\}$
5. *Unrestricted or phrase-structure grammars*, recognized by Turing machines.
6. *Recursively enumerable languages.* A language is recursively enumerable if there is a Turing machine that outputs all the strings of the language. *Example*: $\{a^n \mid f_n(n) \text{ halts}\}$, where $f_0, f_1, f_2, \dots$ is an effective enumeration of all computable functions.
7. *Nongramatical languages*, languages that are not definable by any grammar and cannot be recognized by Turing machines. *Example*: $\{a^n \mid f_n \text{ is total}\}$.