Computability I

Recommended readings: Sipser, Chapter 3.

In this part of the course, we will prove that some problems (languages) cannot be solved by any algorithm. In the NP-completeness part of the course we saw that there are problems (the NP-complete problems) which are (apparently) intractable, in the sense that the best algorithms take too long to compute solutions. However, in principle at least, all the NP-complete problems can be solved; just check all (exponentially many, though) potential certificates.

On the other hand there are problems which do not even have a "bad" algorithm. An example of a problem that cannot be solved by any algorithm is the Halting problem: given a program P and an input x to P, decide whether P halts on input x.

Turing (1936): Introduced TMs. Proved no TM can solve the Halting problem. Argued that any algorithm can be simulated by a TM. This statement in often called the Church-Turing Thesis.

Note that TMs in Sipser have one-way input tapes. If a TM M scans the left end of its tape, and it is told to shift left, it does not move.

Claim 1 Sipser's TMs and our TMs can simulate each other (in fact, they can simulate each other *polynomially*).

How can a TM with one-way tape simulate a TM with a two-way tape?

Idea 1: Use a marker for the right end of the used tape space. If scanning the left end and want to shift left, move the entire tape content right one square.

Idea 2: "Fold" the tape. Store the top part of the tape on even squares, bottom part on odd squares.

General Claim: A TM can simulate every reasonable variation of a TM (e.g., a TM with several tapes).

Let $L(M)$ be the language accepted by a TM M. Then $x \in L(M)$ iff M, when started on input x, eventually enters the state q_{accept}. We assume that the input x is in Σ^*, so $L(M) \subseteq \Sigma^*$.

Note (and this is important to keep in mind for the rest of the course) that x can fail to be in $L(M)$ for two reasons: either M fails to halt on input x, or M halts in the reject state, q_{reject}.

Definition 1 Let $L \subseteq \Sigma^*$. Then L is *semi-decidable* if $L = L(M)$ for some TM M. (Note that Sipser says "recognizable" instead of "semi-decidable".) On the other hand, a TM M

decides L if $L = L(M)$ **and M halts for every input** $x \in \Sigma^*$. L is *decidable* if some TM decides L. Note that we are not concerned with the length of the computation for these definitions.

For example, the Halting problem is semi-decidable, but not decidable; we will see later why.

Definition 2 An *enumerator* E is a TM with a separate write-only output tape. (Sipser says printer instead of output tape.) E writes strings on its output. $L(E)$ is the set of strings x such that when E starts with a blank tape, x is eventually written on the output. Thus $L(E) \subseteq \Gamma^*$, where Γ is the output alphabet. L is *enumerable* iff $L = L(E)$ for some enumerator E.

Theorem 1 L is enumerable iff L is semi-decidable.

PROOF: The proof basically follows from the definitions of semi-decidability and enumerability, and from the notion of "simulation":

If L is enumerable, then $L = L(E)$ for some enumerator E, and then $L = L(M)$ where M is a TM defined as follows:

> On input x, M simulates E (on a blank tape), and each time E outputs a y, M checks if $x = y$, and halts in $q_{\mathtt{accept}}$ if ''yes''.

Thus L is semi-decidable.

Conversely, let M be a TM such that $L(M) = L$. For each string $s \in \Sigma^*$, E must output s only if M accepts s. *Problem*: M may never halt on s. This is why our enumerator E will simulate M on many strings s "in parallel".

Let $s_1, s_2, \ldots$ be a list of all strings in Σ^*. Then E does the following:

```
for i : 1..∞
    for j : 1..i
        if M accepts s_j within i steps then
           output s_j
        end if
    end for
end for
```

If M accepts $s = s_k$ within l steps, then E will output s when $i = \max\{k, l\}$. Hence, L is enumerable. □