Basic Computability

1 m-reducibility

Definition. Let A and B be languages over the same alphabet Σ. A is a many-to-one reducible to B (write $A \leqslant B$) if there is a Turing machine F that terminates on every input $u \in \Sigma^*$, and such that

$$A\{u \in \Sigma^* | F(u) \in B\}$$

Informally: checking $u \in A$ is no harder than checking $w \in B$

1.1 Properties of m-reducibility

Proposition. Suppose $A \leqslant B$

1. If B is Turing-decidable, so is A
2. If B is Turing-recognisable, so is A
3. If $A \leqslant B$ and $B \leqslant C$, then $A \leqslant C$

Definition. Denote $A \equiv B$ to mean that $A \leqslant B$ and $B \leqslant A$

Informally: A and B are equally difficult

2 m-completeness

Definition. A language A is m-complete if

1. A is Turing-recognisable
2. For every Turing-recognisable language B, $B \leqslant A$

Informally: If A is m-complete then A is as hard as any other Turing-recognisable language

Corollary If A is m-complete and $A \leqslant B$, then B is m-complete

Definition - The Halting language H consists of the words $\langle M \rangle \circ w$ (over some fixed alphabet) such that the Turing machine M terminates on w

Theorem H is M complete

Proof: Generic reduction. Pick any Turing-recognisable language A. It is recognised by some machine M_A. Reduce it to H by mapping any word w onto the word $\langle M_A \rangle \circ w$. It is obvious that the reduction is computable and $w \in A$ iff $\langle M_A \rangle \circ w \in H$

Definition: H_0 is the "diagonal" of H, i.e. the language $\langle M \rangle \circ \langle M \rangle$ such that M terminates on $\langle M \rangle$

Theorem: H_0 is m-complete

Proof: Reduction from H. Given a word $\langle M \rangle \circ w$, create a Turing machine $N_{M,w}$ that simulates M on w (and note that it ignores the input) - this can be done using a universal Turing machine. Now, $N_{M,w}$ terminates on any input iff M terminates on w. In particular $N_{M,w}$ terminates on $\langle N_{M,w}$ iff M terminates on w

3 Oracle Turing Machine and t-reducibility

Definition

1. An oracle for a language A is a black-box that takes a word w as an input and instantly (and correctly) replies if $w \in A$

2. An oracle Turing machine M, denotes by M^A is a Turing machine that has an additional capability of making calls to an oracle for the language A

Definition: A language A is t-reducible to a language B is A is decidable by some oracle Turing machine M^B

Theorem: If $A \leqslant_t B$ and B is Turing-decidable, then A is Turing-decidable

4 Computable and Partially Computable Functions

Definition. A total function $f : \Sigma^* \to \Sigma^*$ is computable if there is a TM $\mathscr{F}$ such that on any input $x \in \Sigma^*$, $\mathscr{F}$ produces $f(x)$ as the output

Definition. A partial function $g : \Sigma^* \to \Sigma^*$ is partially computable if there is a TM $\mathscr{G}$ such that on any input $x \in \mathrm{dom}(g)$, $\mathscr{G}$ produces $g(x)$ as the output and if $x \notin \mathrm{dom}(g)$, $\mathscr{G}$ doesn't terminate

Proposition. A language (set) $S \subseteq \Sigma^*$ is Turing-recognisable iff it is:

- The domain of a partially computable function
- The range of a computable function
- The range of a partially computable function

5 Parameter Theorem

Theorem. Let $\mathscr{M}(x, y)$ be a TM that expects a two-part input $x \sqcup y$. There is a TM $\mathscr{S}\mathscr{M}\mathscr{N}(t, x)$ that on inputs $\langle \mathscr{M} \rangle$ and x, produces a (description of a) TM $\langle \mathscr{M}_x \rangle$ such that for every y, $\mathscr{M}_x(y) = \mathscr{M}(x, y)$

6 Recursion theorem

Theorem. Let $\mathscr{M}(x, y)$ be a TM that expects a two-part input $x \sqcup y$. There is a TM $\mathscr{R}(y)$ such that for every y, $\mathscr{R}(y) = \mathscr{M}(\langle \mathscr{R} \rangle, y)$

7 Partially Computable Functions w/o Machines

We consider functions on the set of natural numbers $\mathbb{N}$

Definition. The initial functions are

1. The successor: $s(x) = x + 1$ (returns one more than what you give it)
2. The zero: $n(x) = 0$ (returns 0)
3. The projections $u_i^n(x_1, x_2, ..., x_n) = x_i$ for every $n \in \mathbb{N}$, $1 \leqslant i \leqslant n$ (takes n numbers, returns ith one)

8 Primitive Recursive functions

Definition. A function is called **primitive recursive** if it can be obtained from the initial functions by a finite number of applications of composition and primitive recursion (defined below)

Definition Let f be a function of k variables and let $g_1, g_2, ..., g_k$ be functions of n variables. The function h of n variables if obtained from f and $g_1, g_2, ..., g_k$ by composition if

$$h(x_1, x_2, ..., x_n) =^{def} f(g_1(x_1, x_2, ..., x_n), g_2(x_1, x_2, ...x_n), ...g_k(x_1, x_2, ..., x_n))$$

Definition. Let f and g be total functions of n and $n + 1$ variables, respectively. The function h of $n + 1$ variables is obtained from f and g by primitive recursion if

$$h(x_1, x_2, .., x_n, 0) =^{def} f(x_1, x_2, ..., x_n)$$

$$h(x_1, x_2, ..., x_n, t + 1) =^{def} g(t, h(x_1, x_2, ..., x_n, t), x_1, x_2, ..., x_n)$$

Addition can be defined as follows:

$$a(x, y) = x + y$$

$$a(x, t + 1) = s(a(x, t))$$

Multiplication can be defined as follows:

$$m(x, t + 1) = a(m(x, t), x)$$

9 Gödel Numbers

Given a sequence of numbers $x_1, x_2, ..., x_n$ encode it by a single number

Pick the first n prime numbers and raise each to the respective value of x, so the first prime raised to x_1 etc, apart from the last one, which is raised to $x_n + 1$ and multiply them all together. This will generate the Gödel number of this sequence

You can recover the sequence through factorisation of the Gödel number.

1 is added to the last exponent as it allows you to know where to stop