1 m-reduciblity

Definition. Let A and B be languages over alphabet Σ. A is many-to-one reducible to B, written $A \leq B$, 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, this means that checking $u \in A$ is no harder than checking $w \in B$.

1.1 Properties

Proposition. Suppose $A \leq B$.

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

Denote $A \equiv B$ to mean that $A \leq B$ and $B \leq A$. Informally, this means that A and B are equally difficult.

2 m-completeness

Language A is **m-complete** if:

1. A is Turing-recognisable, and
2. for every Turing-recognisable language B, $B \leq A$.

Informally, if A is m-complete, then A is as hard as any other Turing-recognisable language

Corollary. If A is m-comnplete and $A \leq B$, then B is m-complete.

Definition. The Halting language H consists of the words $\langle M \rangle \sqcup w$, 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 to a word $\langle M_A \rangle \sqcup w$.
The reduction is computable and $w \in A$ if and only if $\langle M_A \rangle \sqcup w \in H$.

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

Theorem. H_0 is m-complete.
Proof. Reduction from H.
Given a word $\langle M \rangle \sqcup w$, create a Turing machine $N_{M,w}$ that simulates M on w.

Note that this machine ignores the input. This can be done using a universal Turing machine.
$N_{M,w}$ terminates on any input if and only if M terminates on w.
Thus, $N_{M,w}$ terminates on $\langle N_{M,w} \rangle$ if and only if 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 word w as input and instantly, and correctly, replies if $w \in A$
2. An oracle Turing machine M, denoted 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 if A is decidable by some oracle Turing machine M^B.

Theorem. If $A \leq_t B$, 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 Turing machine F such that on any input $x \in \Sigma^*$, F produces $f(x)$ as output.

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

Proposition. A language (set) $S \subseteq \Sigma^*$ is Turing-recognisable if and only if 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

Let $M(x, y)$ be a Turing machine that expects the two-part input $x \sqcup y$. There is a Turing machine $SMN(t, x)$ that, on inputs $\langle M \rangle$ and x, produces a description of a Turing machine $\langle M_x \rangle$ such that for every y, $M_x(y) = M(x, y)$.
Proof. Follows from the recursion theorem.

6 Recursion theorem

Let $M(x, y)$ be a Turing machine that expects the two-part input $x \sqcup y$. There is a Turing machine $R(y)$ such that for every y, $R(y) = M(\langle R \rangle, y)$

7 Partially computable functions without machines

Definition. The **initial functions** are:

1. The successor, $s(x) = x + 1$
2. The zero, $n(x) = 0$
3. The projections, $u_i^n(x_1, x_2, ..., x_n) = x_i$ for every $n \in \mathbb{N}$, $1 \leq i \leq n$.

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.

Let f be a function on k variables and $g_1, g_2, ..., g_k$ be functions on n variables. The function h on n variables is obtained from f and $g_1, g_2, ..., g_n$ by **composition** if:

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

Let f and g be toptal functions on n variables and $n + 2$ variables respectively. The function h, on $n + 1$ variables, is obtained from f and g by primitive recursion if:

$$h(x_1, x_2, ..., x_n, 0) =^{def} f(x_1, x_2, ..., x_n)$$
$$h(x_1, x_2, ..., x_n, t + 1) =^{def} g(t, h(x_1, x_2, ..., x_n, t), x_1, x_2, ..., x_n)$$

Primitive recursive functions are computable (find Turing machines to do this).

9 Step-counter function is primitive recursive

Proposition. The following functions are primitive recursive:

- Addition
- Subtraction
- Multiplication
- Integral division (quotient and remainder)

- Exponentiation
- Integral logarithm
- n'th prime number
- i'th digit in base b expansion

Definition. The **Gödel number** of a sequence $x_1, ..., x_n$, defined as $p_1^{x_1}...p_{n-1}^{x_{n-1}}p_n^{x_{n+1}}$ where p_i is the i'th prime number, is primitive recursive.

A string w over a finite alphabet can be encoded by a single number $[w]$.
A Turing machine M can be encoded by a single number $[\langle M\rangle]$, shortened to $[M]$.

A configuration of a Turing machine M – consisting of a state, a head position, and tape content – can be encoded as a single number $[q, i, w]$ via a primitive recursive function, $[q, i, w] = C(q, i, w)$.
If a configuration q, i, w yields a configuration q', i', w', the function $Step([q, i, w]) = [q', i', w']$ is primitive recursive.

The step-counter function can be defined as:

$$SC([M], [w], 0) = [q_{start}, 0, w]$$

$$SC([M], [w], t + 1) = Step(SC([M], [w], t))$$

10 Gödel incompleteness

In a formal system that reason about natural numbers, a formula ϕ is a finite sequence of symbols, so can be suitably encoded by a single number $[\phi]$.

A formula Π in the system can be encoded by a single number $[\Pi]$. The predicate $\text{Proof}([\Pi], [\phi])$, stating that Π is a proof of ϕ, is primitive recursive.

Assume that the system is consistent (cannot prove both ϕ and $\neg\phi$ for any formula ϕ. Assume that every *true* formula system is provable, and assume that the system can reason about a Turing machine's computations.
For every instance of the Halting problem, the predicate:

$$\exists p \ \text{Proof}(p, [M \text{ does not terminate on } w])$$

is semi-decidable (recognisable).

For a positive instance, the predicate is true. By our assumption, there is a proof encoded by some number p. Find p by brute force, trying 0, 1, ..., etc.

But then, the co-Halting problem is semi-decidable. However, as the Halting problem is semi-decidable, this would imply that the Halting problem is decidable – a contradiction.

11 Gödel's self-referential sentence

Let $\phi(x)$ be a formula with one free variable x. Define the predicate $P(n, [\phi])$ to mean "n doesn't encode a proof of $\phi([\phi])$."

Define the formula $\psi(y)$ to be $\forall x P(x, y)$. It has a single free variable y, so consider $\psi([\psi])$.
$\psi([\psi])$ says that every x doesn't encode a proof of $\psi([\psi])$, i.e., $\psi([\psi])$ is not provable, and admits it!

Is $\psi([\psi])$ false? No!
If it were, what it says is false, thus it is provable. However, everything that is provable must be true – a contradiction.
Thus, $\psi([\psi])$ is true, so what it says is true, and $\psi([\psi])$ is not provable.

12 Robinson arithmetic Q

Q is the weakest sub-system of arithmetic in which Gödel's incompleteness holds.
It has a constant zero 0 and function symbols S for successor, $+$ for addition, and $\times$ for multiplication. It has no induction.

12.1 Axioms

1. $S(x) \neq 0$
2. $(S(x) = S(y)) \implies x = y$
3. $y = 0 \vee \exists x(S(x) = y)$
4. $x + 0 = x$
5. $x + S(y) = S(x + y)$
6. $x \times 0 = 0$
7. $x \times S(y) = x \times y + x$