FORMAL DEFINITION OF REDUCTIONS: MAPPING REDUCIBILITY

This material is covered in Section 5.3 of the textbook.

A central task in computability (and in complexity theory to be discussed in the second half of the course) is to *compare the difficulty* of computational problems. Here we formalize the idea that problem A is "easier than" or "at most as difficult as" problem B by defining the notion of mapping reductions.

Mapping reductions can also be viewed as a formalization of the notion reducibility discussed up to this point. Below a mapping reduction can be defined by any computable function. The significant feature, when later discussing algorithmic complexity, is that a mapping reduction allows us to measure how hard it is to compute the reduction.

Definition. A function $f : \Sigma^* \to \Gamma^*$ is a computable function if there exists a Turing machine M such that when M is started on input w, it halts in a configuration that has $f(w)$ as the non-blank contents of the tape, for any $w \in \Sigma^*$.

Note:

- According to the Church-Turing thesis any function on strings that can be computed using an algorithm, is a computable function as defined above.
- The above definition requires that M halts on all inputs. If the definition allows some computations to be infinite, the function could be called "computable partial function".

Definition. Let A and B be languages over Σ. The language A is mapping reducible to language B, denoted $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$ that has the property:

$$(\forall w \in \Sigma^*)\ w \in A \text{ iff } f(w) \in B.$$

We say that A reduces to B via the function f.

Intuitively, the above definition means that testing the property "$w \in A$?" can be algorithmically reduced to the question "$f(w) \in B$?".

The mapping reductions defined above are used similarly as our earlier "informal notion of reduction" for establishing undecidability.

In particular, the relation $\leq_m$ has the following properties:

1. $\leq_m$ is reflexive and transitive (exercise).
2. If $A \leq_m B$ and B is decidable, then A is decidable (Theorem 5.22).
3. If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable (Theorem 5.28).

The above property 2. is typically used in the below form to establish that some language is undecidable:

- If $A \leq_m B$ and A is undecidable, then B is undecidable.

We say that a language A is complete for the class of Turing-recognizable languages if

1. A is Turing-recognizable, and
2. $B \leq_m A$ for all Turing-recognizable languages B.

The languages that are complete for the class of Turing-recognizable languages are in a certain sense "maximally difficult" languages in this class. This is the same idea that we will encounter later when discussing complete languages for complexity classes (the best known example being NP-completeness).

Recall the language A_{TM}:

$$A_{TM} = \{< M, w > |\ M \text{ is a TM and } M \text{ accepts } w\}.$$

Proposition. A_{TM} is complete for the class of Turing-recognizable languages.

Proof. (i) We know that A_{TM} is Turing-recognizable (section 4.2).

(ii) Let B be an arbitrary Turing-recognizable language, $B = L(M_B)$ for some Turing machine M_B. We define the function f by setting

$$f(w) =< M_B, w >, \quad w \in \Sigma^*.$$

The function f is computable (why?), and we have

$$w \in B = L(M_B) \text{ iff } f(w) =< M_B, w > \in A_{TM}.$$

Thus, $B \leq_m A_{TM}$.

Now (i) and (ii) imply that A_{TM} is complete for the class of Turing-recognizable languages. □

The following lemma allows us to obtain many more languages that are complete for the class of Turing-recognizable languages.

Lemma. Assume that A is complete for Turing-recognizable languages, and B is Turing-recognizable. If $A \leq_m B$, then also B is complete for Turing-recognizable languages.

Proof. Use transitivity of $\leq_m$. □

Note that mapping reductions need to be computed by Turing machines that halt on all inputs. On the other hand, a TM recognizing some language need not halt on "incorrect" inputs. Thus, the machine model used to define Turing-recognizable languages is (in some sense) more general than the machine model that computes mapping reductions. This is

what we "want" when defining complete problems for some class of langauges. We illustrate the observation by the following.

Analogously with the complete problems for the class of Turing-recognizable languages we could define that a language A is "complete for the class of decidable languages" if: (i) A is decidable, and (ii) $B \leq_m A$ for all decidable languages B.

- If we use this definition, which languages would be complete for the class of decidable languages? (Once you figure this out, you will see that the definition would not be very useful.)

We will return to this issue when discussing time-bounded or space-bounded mapping reductions.