thy.1 Properties of Reducibility

cmp:thy:ppr:sec

The intuition behind writing $A \leq_m B$ is that A is "no harder than" B. The following two propositions support this intuition.

cmp:thy:ppr:prop:trans-red

Proposition thy.1. *If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$.*

Proof. Composing a reduction of A to B with a reduction of B to C yields a reduction of A to C. (You should check the details!) □

cmp:thy:ppr:prop:reduce

Proposition thy.2. *Let A and B be any sets, and suppose A is many-one reducible to B.*

1. *If B is computably enumerable, so is A.*
2. *If B is computable, so is A.*

Proof. Let f be a many-one reduction from A to B. For the first claim, just check that if B is the domain of a partial function g, then A is the domain of $g \circ f$:

$$\begin{aligned} x \in A &\text{iff } f(x) \in B \\ &\text{iff } g(f(x)) \downarrow . \end{aligned}$$

For the second claim, remember that if B is computable then B and $\overline{B}$ are computably enumerable. It is not hard to check that f is also a many-one reduction of $\overline{A}$ to $\overline{B}$, so, by the first part of this proof, A and $\overline{A}$ are computably enumerable. So A is computable as well. (Alternatively, you can check that $\chi_A = \chi_B \circ f$; so if χ_B is computable, then so is χ_A.) □

digression

A more general notion of reducibility called *Turing reducibility* is useful in other contexts, especially for proving undecidability results. Note that by **??**, the complement of K_0 is not reducible to K_0, since it is not computably enumerable. But, intuitively, if you knew the answers to questions about K_0, you would know the answer to questions about its complement as well. A set A is said to be Turing reducible to B if one can determine answers to questions in A using a computable procedure that can ask questions about B. This is more liberal than many-one reducibility, in which (1) you are only allowed to ask one question about B, and (2) a "yes" answer has to translate to a "yes" answer to the question about A, and similarly for "no." It is still the case that if A is Turing reducible to B and B is computable then A is computable as well (though, as we have seen, the analogous statement does not hold for computable enumerability).

You should think about the various notions of reducibility we have discussed, and understand the distinctions between them. We will, however, only deal with many-one reducibility in this chapter. Incidentally, both types of reducibility discussed in the last paragraph have analogues in computational

complexity, with the added requirement that the Turing machines run in polynomial time: the complexity version of many-one reducibility is known as *Karp reducibility*, while the complexity version of Turing reducibility is known as *Cook reducibility*.

Photo Credits

Bibliography