# The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

## *Solution Manual*
## *Chapter 6, Problem 6.3*

If $A$ many-one reduces to $B$, then it also Turing reduces to $B$: we can view the many-one reduction as a very specialized Turing reduction in which we make a single call to the decision procedure for $B$ and do no postprocessing whatsoever, returning the answer given by the decision procedure for $B$.

That many-one reductions are transitive is easy to see: if $A$ many-one reduce to $B$ through mapping $f$ and $B$ many-one reduces to $C$ through mapping $g$, then the mapping $g \cdot f$ will many-one reduce $A$ to $C$. Formally, given $x \in A$, applying the mapping yields $(g \cdot f)(x) = g(f(x)) = g(y)$. We have $y \in B$ by hypothesis, since $f$ is a many-one reduction from $A$ to $B$; now, given $y \in B$ and since $g$ is a many-one reduction from $B$ to $C$, we must have $g(y) \in C$ as desired. Conversely, given $x' \notin A$, then applying the mapping yields $(g \cdot f)(x') = g(f(x')) = g(y')$. We have $y' \notin B$ by hypothesis, since $f$ is a many-one reduction from $A$ to $B$; now, given $y' \notin B$ and since $g$ is a many-one reduction from $B$ to $C$, we must have $g(y') \notin C$ as desired.

That Turing reductions are transitive is equally easy to see. If we have a procedure $\mathcal{P}_1$ to reduce $A$ to $B$ and another procedure $\mathcal{P}_2$ to reduce $B$ to $C$, then we can reduce $A$ directly to $C$ by replacing within $\mathcal{P}_1$ each call to the decision procedure for $B$ by a call to $\mathcal{P}_2$. The resulting algorithm only makes calls to the decision procedure for $C$ and, by hypothesis, correctly decides membership in $A$.