10 Reductions between Problems

A *reduction* is a way of solving one problem, assuming that you already know how to solve another problem.

10.1 Turing reductions

Suppose that A and B are two computational problems. They can be decision problems or functional problems. We need to formalize the idea that, if we already know how to solve B, we can solve A.

Definition 10.1. A *Turing reduction* from A to B is a program that computes A and that is able to ask questions about B at no cost.

Definition 10.2. Say that $A \le_t B$ provided there exists a Turing reduction from A to B.

10.1.1 Examples of Turing reductions

Example 10.1. Suppose that

$$\begin{aligned} A &= \{n \in \mathcal{N} \mid n \text{ is even}\} \\ B &= \{n \in \mathcal{N} \mid n \text{ is odd}\}. \end{aligned}$$

The following program is a Turing reduction from A to B.

```
"{even(x):
  if x ∈ B then
    return 0
  else
    return 1
}"
```

Notice that it asks if $x \in B$. That is where it uses B at no cost. Since there exists a Turing reduction from A to B, we know that $A \leq_t B$.

Example 10.2. Suppose that

$$\begin{aligned} A &= \{p \mid p \text{ is a quadratic polynomial in } x \text{ and } p \text{ has a real zero}\} \\ B &= \{p \mid p \text{ is a polynomial in } x \text{ and } p \text{ has a real zero}\} \end{aligned}$$

Notice that B allows p to have any degree. If you are allowed to ask about zeros of polynomials of any degree, it is easy to ask questions about quadratic polynomials. The following program is a Turing reduction from A to B, establishing that $A \leq_t B$.

```
"{has-zero(p):
   if p ∈ B then
      return 1
   else
      return 0
}"
```

Example 10.3. You can do a Turing reduction between two functions. Define $f : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ to $g : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ as follows.

$$\begin{aligned} f(m, n) &= m + n \\ g(m, n) &= m \cdot n \end{aligned}$$

Here is a reduction from g to f. It multiplies by doing repeated additions.

```
"{g(m, n):
   y = 0
   for i = 1, ... , m
      y = f(y, n)
   return y
}"
```

Example 10.4. The preceding three examples showed how to define a Turing reduction between two computable problems. You could just replace the test $x \in B$ or the use of $f(p,n)$ by a program that tells you whether $x \in B$ or that computes $f(p,n)$. But this example shows a Turing reduction between two uncomputable problems. Define

$$\begin{aligned}\text{NOTHLT} &= \{(p,x) \mid \text{Run}(p,x)\uparrow\}\\ \text{HLT} &= \{(p,x) \mid \text{Run}(p,x)\downarrow\}\end{aligned}$$

The following is a Turing reduction from NOTHLT to HLT.

```
"{loops(p, x):
   if (p, x) ∈ HLT then
      return 0
   else
      return 1
}"
```

It is important to recognize that the test $(p,x) \in \text{HLT}$ is done without the need for a program that carries out that test. It is done for free. That is good because there is no program that solves the halting problem.

10.1.2 Properties of Turing reductions

Suppose that A and B are computational problems. The following theorem should be obvious. Just use the Turing reduction program.

Theorem 10.1. If $A \le_t B$ and B is computable, then A is computable.

We can turn that around using a tautology that is related to the law of the contrapositive:

$$(P \wedge Q) \rightarrow R \;\equiv\; (P \wedge \neg R) \rightarrow \neg Q.$$

Corollary 10.2. If $A \le_t B$ and A is not computable, then B is not computable.

(A corollary is just a theorem whose proof is obvious or trivial, given a previous theorem.) That suggests a way to prove that a problem B is not computable: choose a problem A that you already know is not computable and show that $A \le_t B$.

10.1.3 An intuitive understanding of relation $\le_t$

Definition 10.2 defines what $A \le_t B$ means. But it can be helpful to have an intuitive understanding to go along with the definition. Let's look at problems from a viewpoint where languages come in only two levels of difficulty: a computable problem is considered easy and an uncomputable problem is considered difficult. Then, according to Theorem 10.1 and Corollary 10.2, $A \le_t B$ indicates that

(a) A is no harder than B, and

(b) B is at least as hard as A.

If you keep that intuition in mind, you will make fewer mistakes. For example, we have seen that, if A is uncomputable and $A \le_t B$, then B is uncomputable too (since it is at least as difficult as uncomputable problem A). What if A is uncomputable and $B \le_t A$? That only tells you that B is no more difficult than an uncomputable problem. So?

10.2 Mapping reductions

Mapping reductions are a restricted form of reductions that only work for decision problems, but that have some advantages over Turing reductions for decision problems. Suppose that A and B are languages.

Definition 10.3. A *mapping reduction* from A to B is a computable function f such that, for every x, $x \in A \leftrightarrow f(x) \in B$.

Definition 10.4. Say that $A \le_m B$ provided there exists a mapping reduction from A to B.

10.2.1 Examples of mapping reductions

Example 10.5. Suppose that

$$\begin{array}{rcl} A & = & \{n \in \mathcal{N} \mid n \text{ is even}\} \\ B & = & \{n \in \mathcal{N} \mid n \text{ is odd}\} \end{array}$$

$f(x) = x + 1$ is a mapping reduction from A to B. There is no need to write a program (except to observe that $f(x)$ is computable).

Example 10.6. Suppose that

$$\begin{aligned} A &= \{p \mid p \text{ is a quadratic polynomial in } x \text{ and } p \text{ has a real zero}\} \\ B &= \{p \mid p \text{ is a polynomial in } x \text{ and } p \text{ has a real zero}\} \end{aligned}$$

Then $f(x) = x$ is a mapping reduction from A to B.

Example 10.7. Define

$$\begin{aligned} K &= \{p \mid \text{Run}(p, p)\downarrow\} \\ \text{HLT} &= \{(p, x) \mid \text{Run}(p, x)\downarrow\} \end{aligned}$$

Then $f(p) = (p, p)$ is a mapping reduction from K to HLT. Notice that

$$\begin{aligned} p \in K &\leftrightarrow \text{Run}(p, p)\downarrow \\ &\leftrightarrow (p, p) \in \text{HLT}. \end{aligned}$$

showing that the requirement $p \in K \leftrightarrow f(p) \in \text{HLT}$ of a mapping reduction from K to HLT is met.

10.2.2 Properties of mapping reductions

Mapping reductions share some properties with Turing reductions.

Theorem 10.3. If $A \leq_m B$ and B is computable then A is computable.

Proof. Suppose that $A \leq_m B$. That is, there exists a mapping reduction from A to B. Ask someone else to provide a mapping reduction f from A to B. The following program is a Turing reduction from A to B.

```
"{a(x):
   y = f(x)
   if y ∈ B
      return 1
   else
      return 0
}"
```

Since $x \in A \leftrightarrow f(x) \in B$, it should be clear that $a(x)$ computes A. So $A \leq_t B$. Now simply use Theorem 10.1.

$\diamondsuit$———————————————————————————$\diamondsuit$

Corollary 10.4. If $A \leq_m B$ and A is not computable then B is not computable.

10.3 Using Turing reductions to find mapping reductions

Students often have find it easier to discover Turing reductions than mapping reductions. One way to discover a mapping reduction is to find a Turing reduction first and to convert that to a mapping reduction. You just need to obey two requirements in the Turing reduction from A to B.

(a) The Turing reduction must only ask one question about whether a string y is in B.

(b) The answer that the Turing reduction returns must be the same as the answer (0 or 1) returned by the test $y \in B$.

If you obey those requirements, then you find that your Turing reduction must have the form

```
"{a(x):
   y = f(x)
   if y ∈ B
      return 1
   else
      return 0
}"
```

The mapping reduction is f.

We showed earlier that, if A and B are defined by

$$\begin{aligned} A &= \{(p,x) \mid \text{Run}(p,x)\uparrow\} \\ B &= \{(p,x) \mid \text{Run}(p,x)\downarrow\} \end{aligned}$$

then $A \leq_t B$. It is worth noting that $A \not\leq_m B$. The reason is that any Turing reduction from A to B must negate the answer that it gets to the question about membership in B, and that is not allowed in a mapping reduction.

prev **next**