CSE491/596 Lecture Fri. 10/9: Mapping Reductions and Undecidability

Definition: A language A **mapping reduces** (also called, **many-one reduces**) to a language B if there is a computable function $f: \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,

$$x \in A \iff f(x) \in B .$$

We then write $A \leq_m B$. Sometimes we add "via f" to emphasize that f does the reduction. Here is an elementary example that involves our pairing function $\langle \cdot, \cdot \rangle$. Recall the definitions of K_{TM} and A_{TM} via Gödel numbers as $K_{TM} = \{i : i \in L(M_i)\}$ and $A_{TM} = \{\langle i, x \rangle : x \in L(M_i)\}$.

Example: $K_{TM} \leq_m A_{TM}$ via $f(i) = \langle i, i \rangle$ for all i.

Recall that when i and x are coded in binary, we can regard "$\langle i, x \rangle$" as literally sandwiching them between the < and , and > characters, then converting from ASCII to binary. In any event, the function $f(i) = \langle i, i \rangle$ is not only computable, it is computable in linear time---hence in polynomial time. We write $A \leq_m^p B$ when the reduction function is computable in polynomial time. This does not matter when we are studying REC and RE, but will be vital when we jump to NP and P. In point of fact, essentially all reductions we see will be polynomial-time computable.

If you don't use the Gödel numbers but identify programs M with string codes written as $\langle M \rangle$, then you would write $f(\langle M \rangle) = \langle M, M \rangle$. There is no need to write "$\langle M, \langle M \rangle \rangle$"---just $\langle M, M \rangle$ signifies that M is being packaged up as both program and data. The problem then becomes what to do with $f(x)$ for strings x that are not valid codes? There are two main styles of handling this:

1. Consider any "non-compiling code" to be a code for the "null machine" M_0. So you would get $f(x) = \langle M_0, M_0 \rangle$.
2. If you know the target language B and a fixed string y_0 thst is not in B, then you can define $f(x) = y_0$ for all "invalid" x.

In this case you can consider these styles to be the same by taking $y_0 = \langle M_0, M_0 \rangle$. When $B = \Sigma^*$, however, both ideas are not applicable.

3. A permissible third way is to *ignore* the issue of invalid codes and regard f as a computable function not from Σ^* *to* Σ^* but (in this case) from the type "One Turing machine" to the type "A Turing machine and a string".

Option 3 is AOK in practice but beware a curious fact: Officially since 1998, the set of valid C++ programs is no longer decidable. For every ANSI standard compiler there are C++ programs that employ "template metaprogramming" in ways that can proliferate like in "The Sorcerer's Apprentice" and make the compiler never halt---until the stack blows. But we may treat "TM" and "Java program" etc. as basic types presumed decidable---which implies they can be put into a nondecreasing 1-to-1 correspondence with all strings (or all numbers), anyway.

Of course the definition of f being computable is that there is a total Turing machine computing it. Many sources reference that Turing machine. There are already the Turing machines being *analyzed* in problems like K_{TM} and A_{TM}. Worse, IMHO, reductions proofs in these sources also refer to hypothetical Turing machines that don't exist. I try to cut down the multiplicity by avoiding the last and portraying the reduction functions as *tranformations of program code*. Here is an example. Define $HP_{TM} = \{\langle i, x\rangle : M_i(x)\downarrow\}$. This is the language of the **Halting Problem**.

Example: $A_{TM} \leq_m HP_{TM}$ via $f(\langle M, x\rangle) = \langle M', x\rangle$ for all i, where M' is transformed from M as follows:

- We may presume M is in "good housekeeping" form with q_{acc} and q_{rej} its only halting states.
- Make M' by adding a loop $(q_{rej}, c/c, S, q_{rej})$ for every $c \in \Gamma$.
- [M' is not in "good housekeeping form" but we can bolt on a new rejecting state q'_{rej} that is never reached to restore that form for cosmetic purposes.]

That gives the **c**onstruction. Next, we observe that the (function f defined by the construction) is **c**omputable. As a code transformation, we just have to find q_{rej} in the code of M and add loops to it. So, in fact, *linear-time* computability is clear. It remains to show **c**orrectness. This means we ned to show that for all machines M and inputs x to M (which are elements of the domain "a machine and a string"):

$$\langle M, x\rangle \in A_{TM} \iff \langle M', x\rangle \in HP_{TM}$$

since $\langle M', x\rangle = f(\langle M, x\rangle)$. Unpacking what membership in the languages of these problems signifies, this means we need to show---again, for all M and x:

M accepts $x \iff M'$ on input x halts.

Sometimes one can show an equivalence "directly" in one go, but often, and as a fallback, one can show the implications in each direction separately:

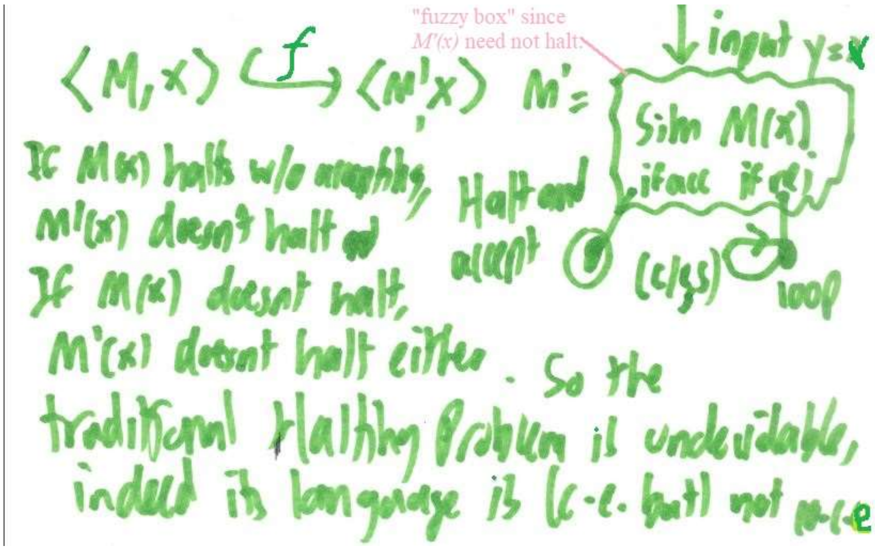
M accepts $x \implies M(x)$ goes to $q_{acc} \implies M'(x)$ goes to q_{acc} as well $\implies M'(x)\downarrow$.
M does not accept $x \implies$ either $M(x)\uparrow$ or $M(x)$ goes to $q_{rej} \implies M'(x)\uparrow$ either way.

For the second part, we could prefer doing the converse:

$M'(x)\downarrow \implies M'$ accepts x (because q_{acc} is the only place M' can halt $\implies M$ accepts x too.

Thus $x \in L(M) \iff M'(x)\downarrow$ so the reduction is correct. ☒

I like to diagram the constructions using little pictures:



Q: Does the same f also reduce HP_{TM} back to A_{TM}?

This finally shows Turing's famous theorem the way he stated it:

Corollary: The (language of the) Halting Problem is undecidable.

We get it as a corollary of the following general theorem.

Theorem 2: Suppose A and B are languages and $A \leq_m B$. Then:

1. if B is decidable then A is decidable;
2. if B is c.e. then A is c.e.;
3. if B is co-c.e. then A is co-c.e.

Moreover, the relation $\leq_m$ is transitive.

The items in this theorem are equivalent to their contrapositives:

Theorem 2$'$: Suppose A and B are languages and $A \leq_m B$. Then:

1. if A is undecidable then B is undecidable;
2. if A is not c.e. then B is not c.e.;
3. if A is not co-c.e. then B is not co-c.e.

So to apply it, note we showed that $A = K_{TM}$ is undecidable because it is the complement of D_{TM} which is not even c.e. The first "B" we use is A_{TM}. By $K_{TM} \leq_m A_{TM}$ we get that the Acceptance Problem is undecidable---moreover, its language is (c.e. but) not co-c.e. Then by transitivity we continue with $B = HP_{TM}$ to get that the Halting Problem is undecidable too. [Most sources follow history by showing the Halting Problem to be undecidable "from the beginning", with the diagonalization embedded among other stuff in the proof, but I was confused when I read it that way as a teenager.]

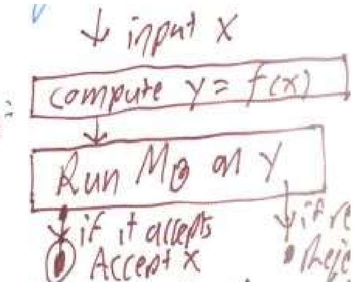
To prove Theorem 2, we can draw more pictures:

<u>Lemma</u>: Suppose $A \le_m B$. Then:

(a) If B is decidable, then so is A.

(b) If B is <u>recognizable</u>, then so is A.

(c) If B is co-c.e. then so is A.



<u>Proof</u>: (a) By B being decidable, we can <u>take</u> a total TM M_B s.t. $L(M_B) = B$. <u>Goal</u>: build a total TM M_A such that $L(M_A) = A$.

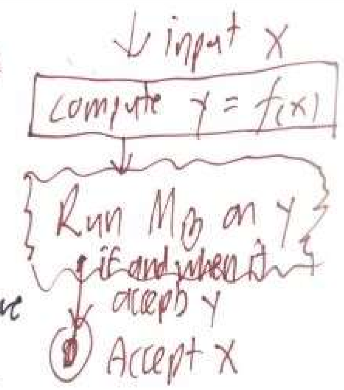
Then M_A is total and M_A accepts $x \iff M_B$ accepts y $(= f(x))$

$\iff f(x) \in B$ by $L(M_B) = B$

$\iff x \in A$ by $A \le_m B$.

So M_A accepts $x \iff x \in A$ and M_A is total, so A is decidable.

(b) Suppose B is merely c.e. Then we can take M_B s.t. $L(M_B) = B$, but M_B might not be total. Can diagram using "fuzzy box".



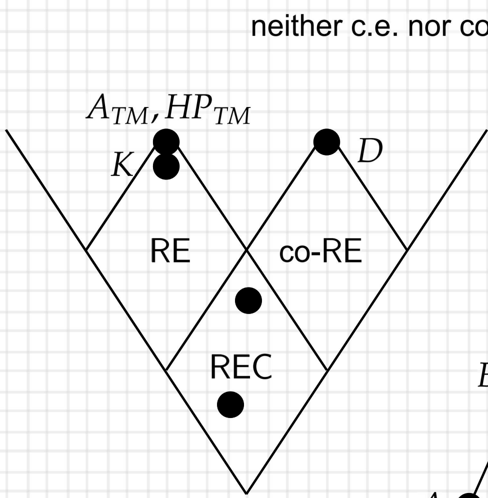
3

Then M_A might not be total either, but we still have M_A accepts $x \iff M_B$ accepts $y \iff y \in B \iff x \in A$.

So $L(M_A) = A$, so A is recognizable/c.e./r.e./etc.

(c) Suppose B is co-c.e. This means $\tilde{B}$ is c.e. By $A \le_m B$, we also have $\tilde{A} \le_m \tilde{B}$. By part (b), $\tilde{A}$ is c.e. Thus A is co-c.e. ☒

The last bit uses the equivalence of $x \in A \iff f(x) \in B$ to $x \notin A \iff f(x) \notin B$. The import of these facts can be conveyed by the last convention in our class "landscape" diagram:

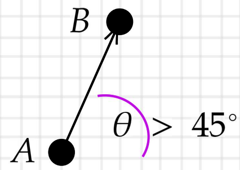


This diagram conveys some extra information:

◎ REC is closed under complements,

◎ RE $\cap$ co $-$ RE $=$ REC, and

◎ All three classes are closed downward under *computable many-one/mapping reductions*.



means $A \leq_m B$