CSE696 Lecture Monday 1 Feb. 2021: Turing Machines, Oracles, and Logic

[Syllabus, nature, and status ofi the course]

[Go over text materials on course webpage]

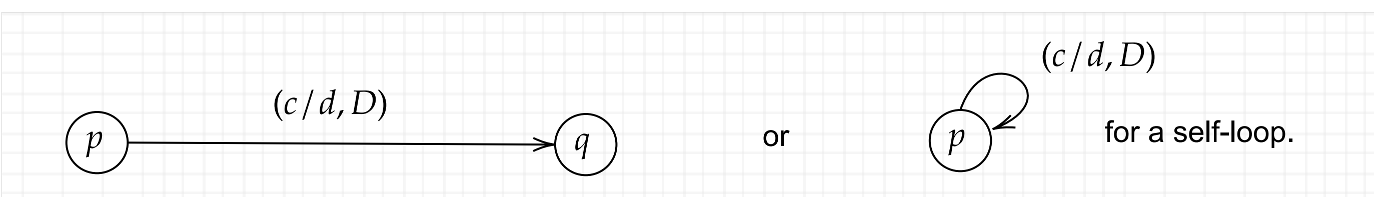
First Lecture, *in memoriam* Alan L. Selman, 1941--2021.

Definition: A *Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ where Q, s, F and Σ are as with a DFA, the *work alphabet* Γ includes Σ and the *blank* _, and

$$\delta \subseteq \left(Q \times \Gamma\right) \times \left(\Gamma \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\} \times Q\right) .$$

It is *deterministic* (a DTM) if no two instructions share the same first two components. A DTM is "in normal form" if F consists of one state q_{acc} and there is only one other state q_{rej} in which it can halt, so that δ is a function from $(Q \setminus \{q_{acc}, q_{rej}\}) \times \Gamma$ to $(\Gamma \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\} \times Q)$. The notation then becomes $M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{acc}, q_{rej})$.

To define the language $L(M)$ formally, especially when M is properly nondeterministic (an NTM), requires defining *configurations* (also called *ID*s for *instantaneous descriptions*) and *computations,* but especially with DTMs we can use the informal understanding that $L(M)$ is the set of input strings that cause M to end up in q_{acc}. Graphically, we will view instructions in δ like so:



We also regard the blank as an explicit character. I will represent it as _ in MathCha but in full LaTeX you can get "\text{\textvisiblespace}" which turns up the corners to look like more than just an underscore. My other notes call the blank B. The blank belongs not to the *input alphabet* Σ but rather to the work alphabet Γ (capital Gamma) which always includes Σ too.

Now we add various "bells and whistles":

- Multiple tapes, including a read-only *input tape* with the rest being *worktapes*.
- An *output tape*, so that TMs can compute (partial) functions $f(x) = y$.
- An **oracle tape**, so that when a string z is written there and a special **query state** $q_?$ is entered, a value $g(z)$ instantly and "magically" appears in place of y on that tape. When the oracle is a language B, the value is 1 if $z \in B$ and 0 otherwise; i.e., the value is $B(z)$.

Example 1:

My favorite example where the oracle is a function $g(\cdot)$ rather than a language; it is motivated by the question of whether integer multiplication $x \cdot y$ can be computed in linear time: Take $g(z) = z^2$ to be the oracle function. Then we can say that multiplication **Turing-reduces** to squaring (in 3 oracle calls that can be made in parallel, i.e., "non-adaptively") via an **oracle Turing Machine** (**OTM**) that embodies the following equation:

$$x \cdot y = \frac{1}{2}\left((x + y)^2 - x^2 - y^2\right).$$

Incidentally, this works not only for integers but also for arithmetic modulo 3, or modulo 5, or 7... But not modulo 2, because you cannot divide by 2 in mod-2 arithmetic. The upshot is that if there is a magic shortcut that allows one to square a number in linear time, then that suffices to multiply in linear time.

Example 2:

When there is just one query y, and when the oracle's answer to y gets copied as the whole machine's final answer, then this is equivalent to a *many-one* reduction. When the oracle is a language B, and the OTM is trying to decide a langauge A, this means we have that for all inputs x,

$$A(x) = B(z), \quad \text{i.e.,} \quad x \in A \iff f(x) \in B,$$

where $f(x) = z$ is the function that computes the string y to use for the oracle query. But when the oracle is a function $g(\cdot)$, we get the picture that many-one reduction is just like composition of functions: the machine outputs $h(x) = g(f(x)) = (g \circ f)(x)$.

So writing $h \leq_m g$ means $h = g \circ f$ for some computable function f. Thus saying $A \leq_m B$ via the reduction function f is the same as having $A = B \circ f$ when languages are identified with 0-1 valued Boolean functions.

When the OTM can do other processing after getting the query answer $g(z)$, then it is a more complicated kind of composition: $h(x, g \circ f(x))$.

Example 3:

The quintessential example (IMHO) of an *unbounded* and **adaptive** Turing reduction is *binary search.* For a topical example, let's use the language form of factoring as the oracle:

$$F = \{\langle x, w \rangle : x \text{ has a prime factor } p \leq w\}.$$

Using this oracle *language* and binary search, we can compute the factoring *function* via*:*

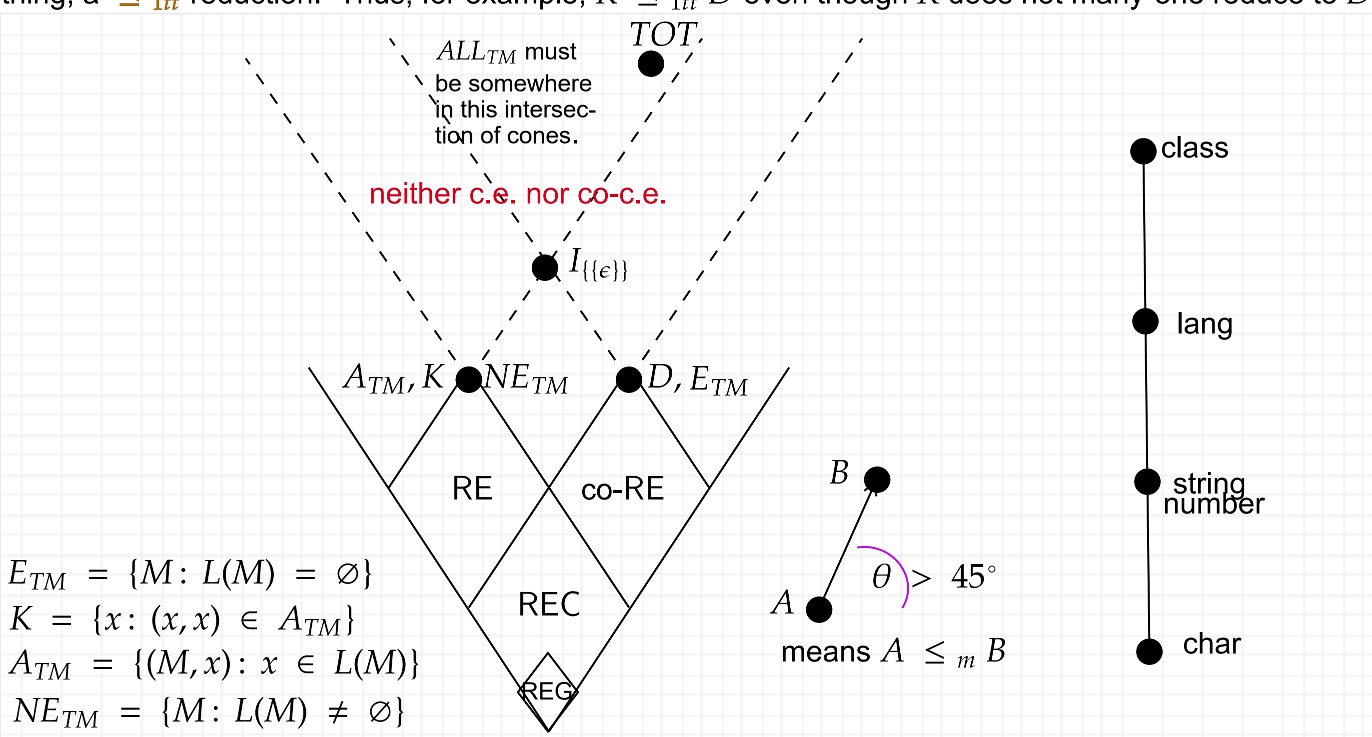
```
int left = 1;
int right = x;
while (left < right-1) {   //INV: left < p ⩽ right
   int mid = (left + right)/2;
   bool b = F(mid);
   if (b) { right = mid; } else { left = mid; }
}
return right;
```

This is **adaptive** because the next query $z =$ the value of `mid` depends on the result of the previous query (and so on) accordingly as whether `left` or `right` was taken. So we can write $factoring \leq_T F$. This example actually completely typifies the general fact:

- Search problems Turing-reduce to decision problems adaptively in quadratic time.

We will later see cases where *randomness helps* make things non-adaptive. A non-adaptive reduction is called a **truth-table reduction**, so we can write $mult \leq_{tt} squaring$ if we wish. Search versus decision and (non-)adaptive were two focal themes of Alan Selman's research.

Example 4:
Now we will spend time up away from polynomial-time complexity theory into undecidable problems. A most basic fact is that every language A Turing-reduces to its complement $\widetilde{A}$ using just 1 query in which the oracle's answer is *inverted*---so it is not a many-one reduction but in a sense the next-closest thing, a $\leq_{1tt}$ reduction. Thus, for example, $K \leq_{1tt} D$ even though K does not many-one reduce to D.



Let I be the language $\{M : L(M) = \{\epsilon\}\}$. Then we get $I \leq_{2tt} K$. Use one query to K (or to A_{TM}) for the $\epsilon \in L(M)$ part. Then modify M to a machine M' that rejects ϵ up-front and otherwise simulates M, so that $L(M') = \emptyset$ if and only if M accepts no strings other than (possibly) ϵ. So the second query goes to the E_{TM} language and we want it to be answered *yes* rather than *no* in order to give our own yes answer to $L(M) = \{\epsilon\}$. Wait, isn't an OTM supposed to have just one language as oracle, and isn't it supposed to be K? Well, A_{TM} many-one reduces to K and so does NE_{TM} which is the complement of E_{TM}. We can use the reduction functions involved to morph the query (M, ϵ) to A_{TM} and then the query M' to E_{TM} into the actual query strings sent to K for answers. Thus, $I \leq_{2tt} K$, so $I \leq_T K$. This puts I into the class REC^K of languages that are "decidable in the Halting Problem." The class of all languages accepted by oracle TMs with K as oracle is similarly denoted by RE^K. This class, and its mirror-image class $\mathsf{co-RE}^K$, have pretty logical characterizations to be shown in the next lecture.

(Un-)Computability Relative to an Oracle

Let $\mathscr{C}$ be a collection of oracle Turing machines M. Then for any language A, define

$$\mathscr{C}^A = \left\{L\left(M^A\right) : M \in \mathscr{C}\right\}.$$

For example, taking the collection of all oracle Turing machines,

$$\mathsf{RE}^A = \left\{L\left(M^A\right)\right\} = \text{the set of all languages that can be accepted with oracle } A.$$

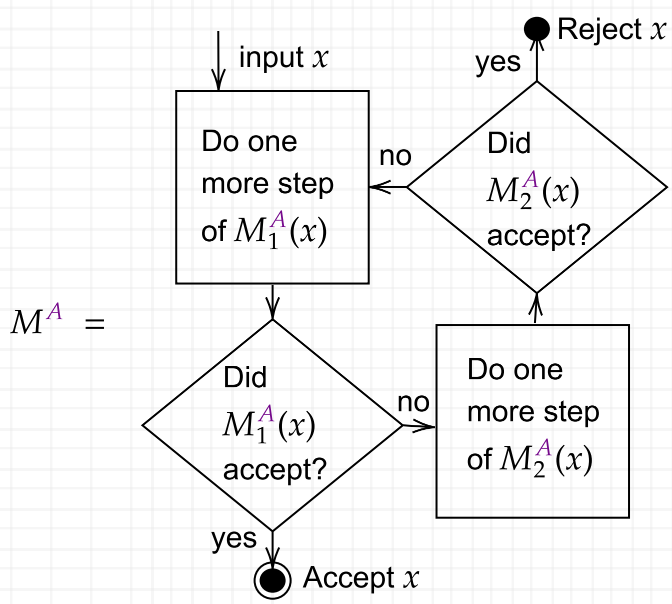
And $\mathsf{REC}^A = \left\{L\left(M^A\right) : M \textit{ is total with oracle } A\right\}$. Now here is a fun puzzle. Saying that M is total---i.e., halts for all inputs---with oracle A is a statement that depends on the particular oracle A. It does not come from M belonging to a collections of machines by themselves. The natural collection to use is that of OTMs that are total with *all* oracles---indeed, that have an associated *time clock* $t(n)$ that shuts them off after $t(|x|)$ steps independent of any answers from the oracle. Call this collection $\mathscr{T}$.

For any A, clearly $\mathscr{T}^A \subseteq \mathsf{REC}^A$ since every machine in $\mathscr{T}$ is of course total with oracle A. But are they equal? That is to say, if you have M and A such that M halts for all inputs when A is the oracle, can we replace M by M' such that $L\left(M'^A\right) = L\left(M^A\right)$ and M' is total with *all* oracles---indeed, has a computable running time bound apart from the oracle? To model this, identify all languages A with branches of the infinite binary tree $\mathcal{B}$. Now see if you can frame the problem in a way that leverages [König's Lemma](): every subtree of $\mathcal{B}$ that has no infinite branch is finite, and in particular, has finite depth.

The possibly-larger definition of REC^A suffices, in any event, for the following two theorems that were proved without an oracle in CSE596.

Theorem 1: For all oracles A, $\mathrm{RE}^A \cap co-\mathrm{RE}^A = \mathrm{REC}^A$.
Theorem 2: For all oracles A, $\mathrm{RE}^A \neq \mathrm{REC}^A$.

To prove Theorem 1, suppose $L \in \mathrm{RE}^A \cap co-\mathrm{RE}^A$. Then there are OTMs M_1, M_2 such that $L(M_1^A) = L$ and $L(M_2^A) = \sim L$. Build an OTM M to carry out the following routine---for any oracle A, not just the given one:



And to prove Theorem 2, define $D_A = \{\langle M\rangle : \langle M\rangle \notin L(M^A)\}$. If D_A were in RE^A, then there would be an OTM Q such that $L(Q^A) = D_A$. But:

Q^A accepts $\langle Q\rangle \iff \langle Q\rangle \in D_A$ by definition of $L(Q^A) = D_A$,
$\iff \langle Q\rangle \notin D_A$ by definition of D_A.

This contradiction shows $D_A \notin \mathrm{RE}^A$, yet its complement (call it K_A) is $\{\langle M\rangle : \langle M\rangle \in L(M^A)\}$, which does belong to RE^A --- but not to REC^A. ☒

The point---which Turing realized in his original 1936 paper---is that these proofs are really the same as the original ones without the oracle. The pink As are not really *used* in the proof. They just "ride along." A further theorem whose proof is the same is

Theorem 3: For all oracles A, K_A is complete for RE^A under reductions that are computable in linear time *without* using the oracle.

Proof: Given any language $L \in \mathrm{RE}^A$, take an OTM M such that $L(M^A) = L$. To reduce L to K_A, map any w to the code of an OTM M_w that on any input x first runs $M^A(w)$, and if and when that accepts, accepts x. Thus either $L(M_w^A) = \Sigma^*$ or $L(M_w^A) = \emptyset$, depending on whether $w \in L$, and only in the "yes" case does M_w^A accept its own code, so $w \in L \iff \langle M_w\rangle \in K_A$. The code of M_w is just a flowchart that ignores x and plugs in "run M on w" without involving anything about the oracle A on the inside. ☒

So we get exactly the same picture with extra "pink As" added: