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

\[
\delta \subseteq (Q \times \Gamma) \times (\Gamma \times \{L, R, S\} \times Q).
\]

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 \( q_{\text{acc}} \) and there is only one other state \( q_{\text{rej}} \) in which it can halt, so that \( \delta \) is a function from \((Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \) to \((\Gamma \times \{L, R, S\} \times Q)\). The notation then becomes \( M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{\text{acc}}, q_{\text{rej}}) \).

To define the language \( L(M) \) formally, especially when \( M \) is properly nondeterministic (an NTM), requires defining configurations (also called IDs 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 \( q_{\text{acc}} \). Graphically, we will view instructions in \( \delta \) 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 \( \Sigma \) but rather to the work alphabet \( \Gamma \) (capital Gamma) which always includes \( \Sigma \) 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:

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

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 language $A$, 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 \leq 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 $z = \text{the value of mid}$ depends on the result of the previous query (and so on) accordingly as whether $\text{left}$ or $\text{right}$ was taken. So we can write $\text{factoring} \leq^T \text{F}$. 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 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 $\overline{A}$ 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 $\leq_{1^T}$ reduction. Thus, for example, $K \leq_{1^{T_1}} D$ even though $K$ does not many-one reduce to $D$.

\[
\begin{align*}
E_{TM} &= \{M: L(M) = \emptyset\} \\
K &= \{x: (x,x) \in A_{TM}\} \\
A_{TM} &= \{(M,x): x \in L(M)\} \\
NE_{TM} &= \{M: L(M) \neq \emptyset\}
\end{align*}
\]
Let $I$ be the language $\{M : L(M) = \{\epsilon\}\}$. Then we get $I \leq_{2t} K$. 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 $\epsilon$ up-front and otherwise simulates $M$, so that $L(M') = \emptyset$ if and only if $M$ accepts no strings other than (possibly) $\epsilon$. 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 $(M, \epsilon)$ to $A_{TM}$ and then the query $M'$ to $E_{TM}$ into the actual query strings sent to $K$ for answers. Thus, $I \leq_{2t} K$, 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 $co - RE^K$, have pretty logical characterizations to be shown in the next lecture.

(Un-)Computability Relative to an Oracle

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

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

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

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

And $REC^A = \{L(M^A) : M \text{ is total with oracle } A\}$. 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 $T$. For any $A$, clearly $T^A \subseteq REC^A$ since every machine in $T$ 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 $L(M'^A) = L(M^A)$ 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 $B$. Now see if you can frame the problem in a way that leverages König's Lemma: every subtree of $B$ that has no infinite branch is finite, and in particular, has finite depth.
The possibly-larger definition of $\text{REC}^A$ suffices, in any event, for the following two theorems that were proved without an oracle in CSE596.

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

**Theorem 2:** For all oracles $A$, $\text{RE}^A \neq \text{REC}^A$.

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

- **M** =
  - input $x$
  - Do one more step of $M_1^A(x)$
  - Did $M_1^A(x)$ accept?
    - yes
      - Accept $x$
    - no
      - Did $M_2^A(x)$ accept?
        - yes
          - Reject $x$
        - no
          - Do one more step of $M_2^A(x)$
  - And to prove Theorem 2, define $D_A = \{\langle M \rangle : \langle M \rangle \notin L(M^A)\}$.
    - If $D_A$ were in $\text{RE}^A$, 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 $D_A \notin \text{RE}^A$, yet its complement (call it $K_A$) is
      - $\{\langle M \rangle : \langle M \rangle \in L(M^A)\}$, which does belong to $\text{RE}^A$ --- but not to $\text{REC}^A$. □

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 $A$s 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 $\text{RE}^A$ under reductions that are computable in linear time without using the oracle.

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

\[E_T^A = \{M : L(M^A) = \emptyset\}\]

\[NE_T^A = \{M : L(M^A) \neq \emptyset\}\]

\[ALL_T^A\text{ must be somewhere in this intersection of cones.}\]

neither c.e. in A nor co-c.e. in A

\[I_{\{e\}}\]

\[A \theta > 45^\circ\]

means \(A \leq_m B\)