Recursively Enumerable & Recursive Sets

We summarize some of the previous results (each integer is identified with the \(n^n\) string \(\langle a(x) \rangle\) in \((0, 1)^n\)):

**Theorem 5.7 (Enumeration Theorem).**

a. A function \(f : ((0, 1)^n)^* \rightarrow (0, 1)\) is partial recursive iff \(f = \Phi_i\) for some \(i \geq 0\) (\(\Phi_i\) is the partial function computed by the machine \(M_i = \langle \Phi_i \rangle = (0, 1)^n \rightarrow (0, 1)^n\)).

b. A set \(A\) is r.e. (recursively enumerable) iff \(A = W_A\) \((W_A = L(M_i)\) is the domain of \(\Phi_i\)) for some \(n \geq 0\).

def.: \(\{W_A\}\) is said to be an effective enumeration of all r.e. sets.

We now have a second "named" result:

**Theorem 5.8 (Projection Theorem).** Let \(A \subseteq \{0, 1\}^*\). The following statements are equivalent:

a. There exists a recursive predicate \(R\) s.t. \(A = \{ x : (3y) R(x, y) \}\).

b. There exists a recursive predicate \(R\) s.t. \(A = \{ x : (3y) R(x, y) \}\).

c. \(A\) is r.e.

\[\text{Proof.}\] b. follows from a., since every primitive recursive function is recursive. Lemma 4.39 \((\phi_1(n) = \text{neg}(\text{neg}(1 + (\text{min}_m R(n, m))))\) provides a proof of c. from b. To obtain a. from c., \(A\) being r.e. implies that \(A = W_A\) for some \(n\). A previous Lemma proved that, for each \(n \geq 0\), the predicate \(R(x, a, t) \equiv \text{halt}(x, n, t) = \{M_i\text{ halts on }x\text{ in at most }t\text{ moves}\}\) is primitive recursive. \(W_A = \{ x : (3y) \text{halt}(x, n, t) \}\)

[Recall: slides 16 and 17 of previous lecture for \(\text{halt}(x, n, t)\)]

Recursively Enumerable & Recursive Sets

**Theorem.** Unions and intersections of recursively enumerable sets are also recursively enumerable.

\[\text{Proof.}\] The "simplest" proof follows from the Projection Theorem.

Let \(A = L(M) = W_A\), \(B = L(M_B) = W_B\). The proof of the Projection Theorem provides the characterizations \(A = \{ x \mid (3r) \text{halt}(x, n, t) \}\), \(B = \{ x \mid (3r) \text{halt}(x, m, t) \}\). Note that \(t_1\) and \(t_2\) will vary with \(x\). Using the characterizations, we observe

\[x \in A \cup B \iff (3r)[\text{halt}(x, n, t) \text{ or } \text{halt}(x, m, t)]\]

\[x \in A \cap B \iff (3r)[\text{halt}(x, n, t) \text{ and } \text{halt}(x, m, t)]\]

This provides us with the required recursive predicates to conclude c. of the Projection Theorem for both \(A \cup B\) and \(A \cap B\).

Recursively Enumerable & Recursive Sets

**Theorem.** Two other, superficially quite different, proofs are available.

**Product DTM proof.** Let \(A = W_A, B = W_B, A \cup B = ?\)

Construct a machine \(M\) that stops when one of the machines \(M_A\) and \(M_B\) stops. We cannot simulate the machines sequentially - the first one chosen may not stop, never letting us try the second.

For the product machine, the states would be \(Q = Q_A \times Q_B\), with transition function \(b([q_i, q_j], [a_k, b_j]) = ([q_k, q_j], [a_k, a_j], (D_1, D_2))\), where \(b([q_i, q_j], [a_k, b_j]) = ([q_k, q_j], [a_k, b_j], (D_1, D_2))\).

The next observation has to do with halting: the states \(q_i\) and \(q_j\) are the highest numbered states in each machine (by the encoding convention), and the product machine must halt in either \([q_i, q_j]\) or \([q_j, q_i]\), where \(i\) and \(j\) can range over all appropriate states. Since, by convention, a Turing Machine has a single halt state (say, \(h\)), we must add some transitions which move us to the halt state, without altering either the tapes of the positions of the heads.
Recursively Enumerable & Recursive Sets

For $A \cap B$ we cannot halt as soon as one of the machines halts - the remaining machine must continue until it halts (if it does). We thus add instructions that will leave everything fixed under the head corresponding to the machine that has halted, while continuing the computation on the other.

Dovetailing (or interleaving) DTM proof. In this case we just run the two machines sequentially, but with a trick that guarantees that we can’t get caught in the machine that might execute forever: the simulating machine keeps a counter of instructions executed on each of the original machines. To start, execute one instruction on both machines. If the correct halting condition is reached, stop. If not, execute two instructions (from the very beginning, always). If the correct halting condition is reached, stop. If not, ....

Recursively Enumerable & Recursive Sets

Ex.: the set \{ $y \mid W_y \neq \emptyset$ \} is r.e.

Pf. We must take all possible strings $y$, and, for each such $y$ use the DTM $M_y$ on all of its possible inputs. Keep running it and halt the scheme when it halts $M_y$ on one input (if the scheme never halts, that’s OK that $W_y = \emptyset$). The trick is that we must increment both the number of instructions and the string set over which we run the simulations. We can simplify the algorithm via the Projection Theorem (and a pairing function): observe that $W_y \neq \emptyset \iff (\exists x) (x \in W_y) \iff (\exists x) (\exists t) (\text{halt}(x, y, t)) \iff (\exists z) (\text{halt}(z, y, r(z)))$, where we have used the pairing function $(a, b)$ and its “inverses” $z = l(z), r(z))$. The algorithm becomes:

1. Set $z := 0$.
2. Simulate $M_y$ on input $x = l(z)$ for $r(z)$ moves. If $M_y$ halts, then accept; otherwise go to step 3.
3. Reset $z := z + 1$, and go back to step 2.

Recursively Enumerable & Recursive Sets

In a similar vein, we have:

Ex. 5.12. The range of a partial recursive function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is a r.e. set.

Pf. Let $f = \phi_1$ (the existence of $y$ follow from $f$ being partial recursive), let $A = \text{range}(f)$. Then:

\[
\exists (x, z, y, t) \left[ f(x, z, y) = \phi_1\left(\exists w\left[\text{print}(l(w), z, y, t)\right]\right) \right]
\]

where $y$ is now fixed by the choice of $f$ and the predicate $\text{print}(l(w), z, y, r(w))$ depends only on $z$ and $w$. 

Recursively Enumerable & Recursive Sets

Def.: A set $A$ is Turing Enumerable if there exists a 2-tape Turing machine $M$, with one tape being a one-way write-only tape, such that, when $M$ is given the empty string as input, it will print an infinite sequence of strings $x_1, x_2, \ldots$ on the write-only tape, with every two strings separated by a blank, and $A = \{x_1, x_2, \ldots\}$. Repetitions are allowed in the output.

Theorem 5.13. Let $A \neq \emptyset$. The following are equivalent:

a. $A$ is Turing Enumerable
b. $A$ is the range of a primitive recursive function
c. $A$ is the range of a recursive function
d. $A$ is the range of a partial recursive function
e. $A$ is r.e.

Proof. b $\Rightarrow$ c $\Rightarrow$ d $\Rightarrow$ e are obvious from definitions and the previous example.
Recursively Enumerable & Recursive Sets

We will complete the proof by closing two cycles: \( e \Rightarrow b \) will close one; \( e \Rightarrow a \Rightarrow c \) will close the other.

\( e \Rightarrow b: \) A r.e. implies \( A = W'_r \) for some \( y \). Let \( x_0 \) be the least string in \( A \), and define

\[ f(x) = \begin{cases} l(x) & \text{if } \text{halt}(t(x), x, t(x)) \text{ halts} \\ l_c & \text{otherwise} \end{cases} \]

The range of \( f \) is \( A \). Since \( \text{hal} \) was shown to be primitive recursive, \( f \) is primitive recursive.

\( a \Rightarrow c: \) Let \( M \) enumerate \( A \). Design a 4-tape DTM \( M' \) that, on input \( n \), prints the \( n \text{th} \) string enumerated by \( M \). Tape 1 is the input tape, where the input \( n \) is used as a counter. Tape 4 is the output tape. Tapes 2 and 3 simulate \( M \). Whenever the simulation of \( M \) prints a string \( x \) followed by a blank on tape 2 (the one-way write-only tape), \( M' \) examines tape 1. If the counter on tape 1 is 0, \( M' \) copies \( x \) to tape 4 and halts, otherwise it decrements the counter and continues the simulation of \( M \).

Theorem 5.14. An infinite set \( A \) is r.e. iff \( t. A \) is the range of a 1-to-1 recursive function.

Proof. We note that \( f \Rightarrow c \) (\( A \) is the range of a recursive function) in the previous result, and so one of the directions (\( = \)) of the proof is immediate. For the other direction, construct a multi-tape machine which will enumerate the strings (according to \( a. \) ) and check if they have been written before. If so, erase the just written string. The map \( f(n) = x \) will follow from \( x \) being the \( n \text{th} \) string left on the tape.

Theorem 5.15. \( A \) is recursive \( \iff \) \( A \) and \( \bar{A} \) are r.e.

Proof.

\[ \Rightarrow A \text{ recursive } \Rightarrow \chi_A \text{ recursive } \Rightarrow (\exists n)[n \geq 0 \text{ and } \chi_A = \phi] \]

Thus

\[ x \in A \iff (\exists n)\text{print}(x, 1, n, t) \]

\[ x \in \bar{A} \iff (\exists n)\text{print}(x, 0, n, t) \]

By the Projection Theorem, both sets are r.e.

\[ \iff A \text{ and } \bar{A} \text{ r.e. } \Rightarrow A = W_{\bar{A}} \text{ and } \bar{A} = W_A \text{ for some } m, n \geq 0. \]

Define

\[ \text{time}(x) = (\min)(\text{halt}(x, n, m, t)) \text{ or } \text{halt}(x, m, t)) \]

Since \( x \in W_{\bar{A}} \) or \( x \in W_A \), \( \text{time} \) is a recursive function. But \( \chi_A(x) = \text{halt}(x, n, \text{time}(x)) \), and \( A \) is recursive.
Recursively Enumerable & Recursive Sets

**Corollary 5.16.** A and B recursive sets ⇒ \( A \cup B, A \cap B \) and \( \overline{A} \) are recursive sets.

**Pf.**: \( \{ x | x \in A \cup B \} = \{ x | \chi_A(x) = 1 \text{ or } \chi_B(x) = 1 \} \) etc...

**Theorem 5.17.** A nonempty set \( A \) is recursive iff it is the range of an increasing recursive function \( f: \{0, 1\}^* \rightarrow \{0, 1\}^* \) (increasing = \( f(x) \leq f(x+1) \) \( \forall x \)).

**Pf.:** ⇒. Let \( x_0 \) be the least string in \( A \) under the lexicographic ordering. Define: \( f(x) = \text{ if } x \leq x_0 \text{ then } x_0 \text{ else } \max_y y \leq x \left[ \chi_A(y) = 1 \right] \).

⇐. Let \( f \) be recursive monotone increasing with range \( A \). Then \( A \) is r.e. (doesn’t need monotone here). If \( A \) is finite, then \( A \) is recursive and so \( \overline{A} \) is r.e. If \( A \) is infinite, \( \overline{A} = \{ x | x < f(0) \text{ or } (\exists y)[f(y) < x < f(y+1)] \} \) so \( \overline{A} \) is r.e. With both \( A \) and \( \overline{A} \) r.e. we conclude \( A \) is recursive.