Computability Theory: Vocabulary Lesson

We call a set $S \subseteq \Sigma^*$ a language.

We say the language S is decidable or recursive if there is a program P such that:

$P(x)$ = yes, if $x \in S$

$P(x)$ = no, if $x \notin S$

We already know: the halting set K is undecidable

Decidable and Computable

Subset S of Σ^* $\Leftrightarrow$ Function f_S

x in S $\Leftrightarrow$ $f_S(x) = 1$

x not in S $\Leftrightarrow$ $f_S(x) = 0$

Set S is decidable $\Leftrightarrow$ function f_S is computable

Sets are "decidable" (or undecidable), whereas functions are "computable" (or not)

Some Important Terminology

We say the language S is recognizable or recursively enumerable if there is a program P such that:
$P(x)$ = yes, if $x \in S$

Claim: The Halting Set K is recognizable.
K = { TM P | P(P) halts }

Claim: K^c = {TM P| P(P) doesn't halt} is not recognizable.

Some Important Terminology

We say the language S is c.e. (computably enumerable) (or sometimes just enumerable) if there is a TM P such that, when started with a blank tape, lists all and only the strings in S (separated by blanks).

We call P an enumerator for S.

Theorem: A language is recognizable iff it is c.e.

Some Important Terminology

Theorem: A language is recognizable iff it is c.e.

Proof:

<=

Suppose there is an enumerator E for L.

How would you build a recognizer for L using E?

Some Important Terminology

Theorem: A language is recognizable iff it is c.e.

Proof:

=>

Suppose that M recognizes L.

Let $s_1, s_2, \ldots$ be a list of all strings in Σ^*

Repeat the following for i= 1,2,3,...

- Run M for i steps on each input $s_1\, s_2 \ldots\, s_i$
- If any of the computations accept, output corresponding s_j

Oracles and Reductions

Use slides from lecture 1 here.

More undecidable problems

We've shown the following undecidable:

- K= {<P> | P is TM and P(P) halts}
- K_0= {<P> | P is TM that takes no input and halts}
- Hello, Equal…

Let's do a few more:

- A_{TM}={(<P>,w)| P accepts w} is undecidable.
- E_{TM}={<P>| L(P) is empty} is undecidable.
- REG_{TM}={<P>| P is a TM and L(P) is a regular language} is undecidable.

Reduction via computation histories (Sipser Section 5.2)

Post Correspondence Problem (PCP)

Input: collection of dominos

Output: yes, if there is a list of these dominos (with repetition) so that the string on top = string on bottom.

Theorem: PCP is undecidable

Computation history

Let M be a Turing machine and w an input string.

The computation history of M on w is the sequence of configurations the machine goes through as it processes the input.

It is a complete record of the computation.

Undecidability of PCP

For any (<P>,w), we'll construct a PCP instance such that there is a match iff P(w) accepts.

Idea: put together a set of dominos that will correspond to a computation history.

Proof on board.

Reducibility (formally)

A function f: Σ^* --> Σ^* is a computable function if there is a TM M that, on every input w, halts with f(w) on its output tape.

Language A is mapping reducible (write A <= B) to language B if there is a computable function f: Σ^* --> Σ^* where for every w, $w \in A$ iff $f(w) \in B$

Reducibility (formally)

Language A is mapping reducible (write $A \leq B$) to language B if there is a computable function f: Σ^* --> Σ^* where for every w, $w \in A$ iff $f(w) \in B$

$A \leq B$ and B is decidable ==> A is decidable

$A \leq B$ and A is undecidable ==> B is undecidable.

$A \leq B$ and B is recognizable ==> A is recognizable.

$A \leq B$ and A is not recognizable ==> B is not recognizable.

Rado's Busy Beaver

We can classify Turing machines by how many rules they have in the tape head.

Of the ones with n rules, some halt and others run forever when started on a blank tape.

What's the maximum number of steps S(n) that any machine with n rules takes before it halts?

Call this number S(n) = nth "Busy Beaver" number.

S(n): finds the busiest beaver with n rules, albeit not infinitely busy.

Rado's Busy Beaver

What's the maximum number of steps S(n) that any machine with n rules takes before it halts?

S(n) = nth "Busy Beaver" number.

n	S(n)
1	1
2	6
3	21
4	107
5	> 47,176,870
6	> 8,690,333,381,690,951

In fact, they grow so fast that we can prove:
Theorem: S(n) is not computable.

Rado's Busy Beaver

What's the maximum number of steps S(n) that any machine with n rules takes before it halts?

S(n) = nth "Busy Beaver" number.

Theorem:

There is no computable function C such that $S(n) \leq C(n)$ for all n.

i.e., S(n) grows faster than any computable function.

Some of the big ideas we've seen so far

- The Turing Machine model and the Church-Turing thesis
- Universality via duality
- Undecidability.
- Diagonalization and the different types of infinity
- Notion of reduction.

Next up: Complexity

We focus next on efficiency of computation.

Let T: N --> N
DTIME(T(n)) is the set of Boolean functions that are computable in O(T(n)) time.

Our notion of efficiently solvable: polynomial time computable.

$P = \cup_c \text{DTIME}(n^c)$

Circuit Complexity

Question:

- Given a Boolean function f: $\{0,1\}^n \dashrightarrow \{0,1\}$, what is the size of the smallest circuit that computes it? (how many gates?)
- Warmup: XOR of n inputs given 2-input XOR gates. How many do we need?

Shannon's Counting Argument

Is there a Boolean function with n inputs that requires a circuit of exponential size in n ?

Yes, in fact, most functions.

Very complex functions exist, but this argument doesn't give us a single example!!!
Called nonconstructive.

Hartmanis-Stearns

The QuickHalt Problem:
Given as input a TM P, int n, does P(P) halt in $\le n^3$ steps?

Claim: Any TM to solve this problem needs at least n^3 steps.

THEOREM: There is no program to solve the QuickHalt problem in $< n^3$ steps.

Suppose a program QHALT existed that solved the quick halting problem in say $n^{2.99}$.

QHALT(P,n) = yes, if P(P) halts in $\le n^3$
QHALT(P,n) = no, otherwise.

We will call QHALT as a subroutine in a new program called CONFUSE.

CONFUSE

CONFUSE(P)
{ if (QHALT(P,n))
then loop forever;
// i.e., P(<P>) halts in n^3 steps
else exit; // in this case, Confuse halts in $\le n^{2.99}$ steps.
}

What happens with CONFUSE(CONFUSE)?

CONFUSE

CONFUSE(P)
{ if (QHALT(P,n))
then loop forever;
// i.e., P(<P>) halts in n^3 steps
else exit; // in this case, Confuse halts in $\le n^{2.99}$ steps.
}

Suppose CONFUSE(CONFUSE) halts in $\le n^3$ steps:
then QHALT(CONFUSE,n) = TRUE
$\Rightarrow$ CONFUSE(CONFUSE) will loop forever
Suppose CONFUSE(CONFUSE) doesn't halt in $\le n^3$
then QHALT(CONFUSE,n) = FALSE
$\Rightarrow$ CONFUSE(CONFUSE) will halt in $\le n^3$

CONTRADICTION

Theorems we skipped from Arora/Barak Chap 1

Robustness of TM definition (alphabet size, number of work tapes, bidirectional tapes)

Efficient Universal Turing Machine

Many others in Sipser Chapters 3-5.

Extra Problems if there is time

Rice's Theorem

Problems from homework