ADVENTURES IN LOGIC FOR UNDERGRADUATES

by

Harvey M. Friedman

Distinguished University Professor

Mathematics, Philosophy, Computer Science

Ohio State University

Lecture 3. Turing Machines

LECTURE 1. LOGICAL CONNECTIVES. Jan. 18, 2011

LECTURE 2. LOGICAL QUANTIFIERS. Jan. 25, 2011

LECTURE 3. TURING MACHINES. Feb. 1, 2011

LECTURE 4. GÖDEL'S BLESSING AND GÖDEL'S CURSE. Feb. 8, 2011

LECTURE 5. FOUNDATIONS OF MATHEMATICS Feb. 15, 2011

SAME TIME - 10:30AM

SAME ROOM - Room 355 Jennings Hall

WARNING: CHALLENGES RANGE FROM EASY, TO MAJOR PARTS OF COURSES

TURING MACHINES - GENERAL STRUCTURE

Turing machines are the original theoretical model of computation, due to Alan Turing, 1937. It is extremely primitive. We follow the version by Emil Post, 1947. Remarkably, from the theoretical point of view, it is a "complete" model of computation - in various senses of "complete".

A TM consists of a "tape" which is infinite in both directions, divided into unit squares indexed by the integers. A TM also comes with a "reading head" which always hovers over a square of tape, and "reads" what is on that square of tape.

Computation begins based on a "program" and an "input". First, "initialization" takes place. At any stage of computation, the TM is in a "state", each square of tape has a "symbol" written on it, and the reading head is "on" a square of tape.

TURING MACHINES - GENERAL OPERATION

TM, infinite two way infinite tape, reading head, program, input, initialization, state, symbol.

The states (state symbols) are written $q_0, q_1, \ldots$. The symbols (tape symbols) are written $S_0, S_1, \ldots$.

Initialization is based on the input. It causes the TM to go into state q_0 (the initial state), with the reading head at square 0. The squares of tape have symbols placed on them according to the input.

Computation continues step by step, according to the program. The reading head may change symbols it is reading, and may move left or right one square. The TM state may change. Computation halts if and when no program instruction applies. The output is read off from the symbols on the tape. If computation never halts, then there is no output.

TURING MACHINES - INITIALIZATION

TM, infinite two way tape, reading head, program, input, initialization, state, symbol. Reading changing symbols, moving left/right. Halting when no instruction applies. Halting yields output. No halting means no output. States $q_0, q_1, \ldots$. Symbols $S_0, S_1, \ldots$.

The inputs for a TM are finite sequences of nonnegative integers. The outputs for a TM are nonnegative integers. (For some purposes, finite strings from a finite alphabet are used).

Let $n_1, \ldots, n_k \in N$. S_0 serves as the "blank", S_1 as the "1".

Initialize as follows: at square 0, put n_1+1 S_1's, followed by S_0, followed by n_2+1 S_1's, followed by S_0, ..., followed by n_k+1 S_1's, followed by all S_0's. To the left of square 0, put all S_0's. Put TM into state q_0 (initial state). Put heading head on square 0.

Note that if k = 0 then all squares have S_0 after initialization.

TURING MACHINES - INSTRUCTIONS

TM, infinite two way tape, reading head, program, input, initialization, state, symbol. Reading changing symbols, moving left/right. Halting when no instruction applies. Halting yields output. No halting means no output. States $q_0,q_1,\ldots$. Symbols $S_0,S_1,\ldots$. q_0 = initial state, S_0 = blank, S_1 = 1. Initialize with $n_1,\ldots,n_k$ from N. Output (if any) from N.

Instructions are quadruples. There are 3 types of instructions.

$q_iS_jLq_k$ If in state q_i, reading S_j, move reading head to the left and go into state q_k.

$q_iS_jRq_k$ If in state q_i, reading S_j, move reading head to the right and go into state q_k.

$q_iS_jS_mq_k$ If in state q_i, reading S_j, replace by S_m and go into state q_k.

Applicability of an instruction is based on the first two items. The last two items tell us what action is to be taken.

TURING MACHINES - COMPUTATIONS

TM, infinite two way tape, reading head, program, input, initialization, state, symbol. Reading changing symbols, moving left/right. Halting when no instruction applies. Halting yields output. No halting means no output. States $q_0,q_1,\ldots$. Symbols $S_0,S_1,\ldots$. q_0 = initial state, S_0 = blank, S_1 = 1. Initialize with $n_1,\ldots,n_k$ from N. Output (if any) from N. Instructions: $q_iS_jLq_k$ $q_iS_jRq_k$ $q_iS_jS_mq_k$

A *program* is a finite set of instructions, where any two instructions with the same first two items are the same. Computation proceeds after initialization, by executing the at most one instruction that applies.

***Halting* occurs if and when no instruction applies. Upon halting, the *output* is the total number of S_1's on the tape.**

We will use P for programs, and α for finite sequences from N (nonnegative integers). We write VAL(P,α) for the output of the TM computation with program P and input α.

CAUTION: VAL(P,α) may not be defined. VAL(P,α) is defined if and only if halting occurs with program P and input α.

TURING COMPUTABLE FUNCTIONS

VAL(P,α) is the output of the TM computation with program P and input α (a finite sequence from N).

Write N* for the set of all finite sequences from N. It is important to consider not only f:N* → N, but also partially defined f:N* → N; where f is defined only at some arguments from N*.

We say that a program P computes partial f:N* → N if and only if each f(α) is VAL(P,α).

We say that partial f:N* → N is Turing computable if and only if there is a program that computes f.

Obviously, partial f:N^k → N are partial f:N* → N.

CHURCH'S THESIS

After Turing machines were invented, many richer models of computation were studied. It was shown that all these models of computation lead to the same class of computable partial $f:N^k \rightarrow N$.

On this basis, Alonzo Church formulated the thesis that any "computable" partial $f:N^k \rightarrow N$ is Turing computable. This supports use of the intuitive notion of "computable" instead of Turing computable. This is a great simplification.

In particular, because of Church's Thesis, we never have to go back to the definition of a Turing machine.

We are going to use Church's Thesis to obtain a number of elegant results. But before this, we will prove a theorem that strongly supports Church's Thesis.

CHURCH'S THESIS - A JUSTIFICATION

"All computable partial $f:N^k \to N$ are Turing computable"

There is a standard system of axioms and rules for mathematics called ZFC, which we will discuss in some detail in Lecture 5.

Suppose we have an intuitively computable partial $f:N^k \to N$. It seems clear that we should have a definition of f within ZFC which is not only correct, but also has the following properties.

If $f(n_1,...,n_k) = m$ then ZFC proves "$f(n_1,...,n_k) = m$".
If ZFC proves "$f(n_1,...,n_k) = m$" then $f(n_1,...,n_k) = m$.

THEOREM. The above holds for f iff f is Turing computable. Furthermore, if we replace ZFC here by any "reasonable" system with finitely many axioms, then f is Turing computable.

We will now assume Church's Thesis.

TERMINOLOGY: RECURSIVE, PARTIAL RECURSIVE

In light of Church's Thesis, we stop mentioning "Turing", since all reasonable models of computation are the same as intuitive computability.

The standard terminology has become

Recursive function **$f:\mathbf{N}^k \to \mathbf{N}$, for Turing computable $f:\mathbf{N}^k \to \mathbf{N}$.**

Partial recursive function **$f:\mathbf{N}^k \to \mathbf{N}$ for Turing computable partial $f:\mathbf{N}^k \to \mathbf{N}$.**

In some modern texts and papers, *Computable function* and *partial computable function* are used instead of recursive function and partial recursive function.

$\cong$ is used for *partial equality*. $a \cong b$ means "either $a = b$, or a,b are both undefined".

RECURSIVE FUNCTIONS - COMPOSITION

THEOREM. If $f,g:\mathbb{N} \to \mathbb{N}$ are recursive, then so is fog.

To compute $fog(n)$, first compute $g(n)$. Then compute $f(g(n)) = fog(n)$. This is an intuitive algorithm based on those for f,g.

THEOREM. Suppose $f:\mathbb{N}^r \to \mathbb{N}$ and $g_1,\ldots,g_r:\mathbb{N}^k \to \mathbb{N}$ are recursive. Then $h:\mathbb{N}^r \to \mathbb{N}$ is recursive, where $h(n_1,\ldots,n_r) = f(g_1(n_1,\ldots,n_k),\ldots,g_r(n_1,\ldots,n_k))$.

CHALLENGE: Prove this Theorem.

PARTIAL RECURSIVE FUNCTIONS - COMPOSITION

Consider f(g(x)). In order for this to be defined, g(x) must be defined. More generally, in order for $f(g_1(x),...,g_r(x))$ to be defined, we require that $g_1(x),...,g_r(x)$ be ALL defined.

THEOREM. If $f,g:N \rightarrow N$ are partial recursive, then so is fog.

To compute fog(n), first compute g(n). If and when g(n) gets computed, compute f(g(n)) = fog(n). This is an intuitive algorithm based on those for f,g.

THEOREM. Suppose $f:N^r \rightarrow N$ and $g_1,...,g_r:N^k \rightarrow N$ are partial recursive. Then $h:N^r \rightarrow N$ is partial recursive, where $h(n_1,...,n_r) \cong f(g_1(n_1,...,n_k),...,g_r(n_1,...,n_k))$.

CHALLENGE: Prove this Theorem.

RECURSIVE FUNCTIONS - CONDITIONAL BRANCHING

Conditional branching is written using IF THEN ELSE.

IF n = 5 THEN m ELSE r

defines a function of n,m,r. It is m if n = 5; r otherwise

IF n = 5 THEN m ELSE IF m = 7 THEN r ELSE t

is a function of n,m,r,t. It is m if n = 5; r if m = 7 ∧ n ≠ 5; t otherwise.

THEOREM. Let $f_1,...,f_r,g_1,...,g_r,h:N^k \rightarrow N$ be recursive. Then $F:N^k \rightarrow N$ is recursive, where $F(n_1,...,n_k)$ = IF $f_1(n_1,...,n_k) = 0$ THEN $g_1(n_1,...,n_k)$ ELSE IF $f_2(n_1,...,n_k) = 0$ THEN $g_2(n_1,...,n_k)$ ELSE ... ELSE IF $f_r(n_1,...,n_k) = 0$ THEN $g_r(n_1,...,n_k)$ ELSE $h(n_1,...,n_k)$.

CHALLENGE: Prove this Theorem.

RECURSIVE SETS

Let $A \subseteq N^k$. The *characteristic function* of A, written $\chi(A)$, is defined by

$$\chi(A)(n) = 1 \text{ if } n \in A;\ 0 \text{ otherwise.}$$

We say that $A \subseteq N^k$. is *recursive* if and only if $\chi(A)$ is recursive.

THEOREM. $A \subseteq N^k$ is recursive if and only if there is an algorithm which, at every x in N^k, eventually determines whether or not x is in A.

THEOREM. The union (intersection) of finitely many recursive subsets of N^k is recursive. The complement of every recursive subset of N^k (relative to N^k) is recursive. Every finite subset of N^k is recursive.

CHALLENGE: Prove these Theorems.

RECURSIVELY ENUMERABLE SETS

$A \subseteq N^k$ is recursive if and only if $\chi(A)$ is recursive.

We say that $A \subseteq N^k$ is *recursively enumerable* (r.e.) if and only if A is the domain of a partial recursive function.

THEOREM. $A \subseteq N^k$ is r.e. if and only if there is an algorithm which, when presented with x in N^k, halts if x is in A, and runs forever if x is not in A.

THEOREM. If $A \subseteq N^k$ is recursive then A is r.e.

THEOREM. The union (intersection) of any finite number of r.e. subsets of N^k is r.e.

CHALLENGE: Prove these Theorems.

RECURSIVE SETS AS R.E. SETS

$A \subseteq N^k$ is recursive if and only if $\chi(A)$ is recursive.

$A \subseteq N^k$ is r.e. if and only if A is the domain of a partial recursive function.

The following Theorem defines the recursive sets in terms of the r.e. sets.

THEOREM. $A \subseteq N^k$ is recursive if and only if A and $N^k \backslash A$ are r.e.

Assume $A \subseteq N^k$ be recursive. Then $N^k \backslash A$ is recursive (previous challenge). Hence A and $N^k \backslash A$ are r.e. (previous challenge).

Assume A,$N^k \backslash A$ be r.e. Given x in N^k, look for x in A, and also look for x in $N^k \backslash A$. You must "sequentially time share" between the two algorithms, so that you don't get stuck forever running just one of the two. (Perfect time sharing requires concurrency, which we are not allowing.)

CHALLENGE: Be more explicit about the time sharing used here.

RANGES OF (PARTIAL) RECURSIVE FUNCTIONS

THEOREM. The range of every recursive $f:N^k \rightarrow N$ is r.e.

Given n in N, apply f successively to all of N^k and wait until you get the value n.

CHALLENGE: Make "apply f successively to all of N^k" rigorous.

THEOREM. The range of every partial recursive $f:N^k \rightarrow N$ is r.e.

CHALLENGE: Prove the above Theorem. You must use a lot of sequential time sharing.

THEOREM. Every nonempty r.e. subset of N is the range of some recursive $f:N \rightarrow N$.

CHALLENGE: Prove the above Theorem. You must consider "number of steps of computation".

PARTIAL RECURSIVE ENUMERATION OF PARTIAL RECURSIVE FUNCTIONS

A partial recursive enumeration of the partial recursive functions is a listing $f_n:N \rightarrow N$ of all partial recursive functions from N into N, such that the partial function

$$f(n,m) \cong f_n(m)$$

is partial recursive.

There is a reasonable enumeration without repetition of the Turing machine programs, $P_0,P_1,\ldots$. It is customary to write $\varphi_0,\varphi_1,\ldots$ for the partial recursive functions from N to N that are computed by $P_0,P_1,\ldots$, respectively. Note that

$$\varphi(n,m) \cong \varphi_n(m)$$

is partial recursive. Hence $\varphi_0,\varphi_1,\ldots$ is a partial recursive enumeration of the partial recursive functions.

R.E. ENUMERATION OF THE R.E. SETS

We now have a partial recursive enumeration $\varphi_0,\varphi_1,\ldots$ of the partial recursive functions from N into N.

An r.e. enumeration of the r.e. sets is a listing $A_0,A_1,\ldots$ of all r.e. subsets of N, such that the set

$$\{(n,m): m \in A_n\}$$

is an r.e. subset of N^2.

CHALLENGE: Prove that $\mathrm{dom}(\varphi_0),\mathrm{dom}(\varphi_1),\ldots$ is an r.e. enumeration of the r.e. subsets of N.

It is very common to write W_e for $\mathrm{dom}(\varphi_e)$.

Thus we have an r.e. enumeration $W_0,W_1,\ldots,$ of the r.e. subsets of N.

AN R.E. SET THAT IS NOT RECURSIVE

We have an r.e. enumeration $W_0, W_1, \ldots,$ of the r.e. subsets of N.

What about

$$\{e: e \in W_e\}?$$

CHALLENGE: Show that the above set is r.e.

Its complement (relative to N) is

$$\{e: e \notin W_e\}.$$

CHALLENGE: Show that the above set is not among the $W_0, W_1, \ldots$. I.e., it is not r.e.

Hence $\{e: e \in W_e\}$ is an r.e. set whose complement is not r.e. Since the complement of a recursive set is recursive, clearly $\{e: e \in W_e\}$ is an r.e. set that is not recursive.

A MOST POWERFUL R.E. SET

The set $\{e: e \in W_e\}$ turns out to be a "most powerful" r.e. set in the following sense.

Let $A,B \subseteq N$. We say that A is *reducible* to B if and only if membership in A can be algorithmically reduced to membership in B, in the following sense.

There is a recursive $f:N \rightarrow N$ such that for all $n \in N$,

$$n \in A \text{ if and only if } f(n) \in B.$$

CHALLENGE: Every r.e. subset of N is reducible to $\{e: e \in W_e\}$.

We say that $B \subseteq N$ is *complete r.e.* if and only if B is r.e. and every r.e. subset of N is reducible to B.

Thus $\{e: e \in W_e\}$ is complete r.e.

COMPLETE R.E. SETS ARE ALL THE "SAME"

We say that A,B ⊆ N are *recursively isomorphic* if and only if there is a recursive bijection f:N → N which maps A onto B.

CHALLENGE: Show that recursively isomorphic is an equivalence relation.

THEOREM. Any two complete r.e. sets are recursively isomorphic.

CHALLENGE: Prove the above Theorem.

HILBERT'S TENTH PROBLEM

Hilbert's 10th Problem calls for an algorithm that determines whether a given polynomial with integer coefficients has a zero at some integer arguments.

We can rigorously formulate this problem as follows.

We can naturally view any polynomial with integer coefficients and variables $v_1, v_2, \ldots$ as a finite sequence of elements of N (i.e., an element of N*): Read the polynomial as a sequence of items from left to right. The items are

i. Integers (subscripts of variables, coefficients, and exponents).
ii. The symbols +,-,v.

There is no need for parentheses or the multiplication symbol.

HILBERT'S TENTH PROBLEM

Algorithm determining whether integral polynomials have integral zeros?
i. Integers (subscripts of variables, coefficients, and exponents).
ii. The symbols +,-,v.

We arbitrarily assign 0,1,2 to +,-,v. In this way, every integral polynomial is represented as an element of N*.

We can now consider the set

H10 = the set of all integral polynomials with an integral zero, viewed as an element of N*.

THEOREM. The set H10 $\subseteq$ N* is r.e.

CHALLENGE: Rework this Lecture using N* instead of the N^k.

CHALLENGE: Prove the above Theorem.

HILBERT'S TENTH PROBLEM

THEOREM. H10 $\subseteq$ N* is complete r.e.

COROLLARY. There is no algorithm for determining whether an integral polynomial has an integral zero.

THEOREM. Every r.e. subset of N is the set of values in N of some integral polynomial.

Nobody thinks that we have the "right" proof of these results. The proofs raise more questions than they answer. The situation has been roughly constant in this respect since the 1970s.

CHALLENGE: Find a new proof of the Corollary.

CHALLENGE: Is there an algorithm for determining whether an integral polynomial has a rational zero? Probably not.