Lecture 15:
The Church-Turing Thesis

Lecture Outline

- Turing-Computable Functions
- The Church-Turing Thesis
- Turing-Computability of Partial Recursive Functions
- The Partial Recursiveness of Turing Machines

Reading

Chapter 4.3 of Brookshear

Chapter 13of Martin, 2nd ed. (slightly different treatment)

N.J. Cutland *Computability: An Introduction to Recursive Function Theory*, Cambridge University Press, Cambridge, 1980. Chapters 1-2.

Turing-Computable Functions

- In the same way that Turing machines have been conjectured to have equivalent or greater computing power than any computing machine (**Turing's thesis**), the class of partial recursive functions has been conjectured to contain all computable functions.

  This conjecture is known as **Church's thesis** after the American logician Alonzo Church who did much of the early work in recursive function theory.

- A natural question to ask is: How do Church's thesis and Turing's thesis relate?

- To formulate this question more precisely we first need to see how Turing machines can be used to compute functions.

- A Turing machine can be viewed as computing a function from its initial tape configuration to its final tape configuration.

  More precisely we say a Turing machine $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ computes the partial function $f : \Sigma^{*^m} \to \Sigma_1^{*^n}$, where Σ_1 is a subset of the non-blank symbols in Γ, if for each $(w_1, \ldots, w_m)$ in Σ^{*^m} starting M with tape configuration

  $$\underline{\Delta} w_1 \Delta w_2 \Delta \cdots \Delta w_m \Delta\Delta\Delta$$

  has the following results:

  1. if $f(w_1, \ldots, w_m)$ is defined then M halts with tape configuration $\Delta v_1 \Delta v_2 \Delta \cdots \Delta v_n \Delta\Delta\Delta$ where $(v_1, \ldots, v_n) = f(w_1, \ldots, w_m)$;

  2. if $f(w_1, \ldots, w_m)$ is undefined then M never halts (though it may abnormally terminate).

  Any partial function that is computed by a Turing machine is said to be **Turing-computable**.

Turing-Computable Functions (cont)

- Since partial recursive functions are functions $f : \mathbf{N}^m \rightarrow \mathbf{N}^n$ we need a convention for representing n-tuples of integers on a TM's tape, if it is to compute these functions.

  We use binary representations of the integers and blank-delimit them as before. E.g. the tape configuration

  $$\Delta 10 \Delta 11 \Delta 111 \Delta\Delta$$

  would represent the triple

  $$(2, 3, 7).$$

- Note that while we have not insisted that the machine halt with its head over the leftmost cell is easy to add this requirement without altering the class of functions computed.

The Church-Turing Thesis

- Now that we have precisely defined what it is for a Turing machine to compute a function, we can ask how the class of Turing-computable functions relates to the classes of recursive functions we have been examining.

- As it turns out, the class of Turing-computable functions is exactly the same as the class of partial recursive functions.

  To prove this we need to prove two propositions:

  > *Every partial recursive function is Turing-computable.*
  >
  > *Every Turing-computable function is partial recursive.*

- It follows that Turing's thesis and Church's thesis are equivalent – hence they are frequently collectively referred to as the **Church-Turing thesis**.

  Both propose formally equivalent limits as to what can be calculated by an **algorithm** or **effective procedure**.

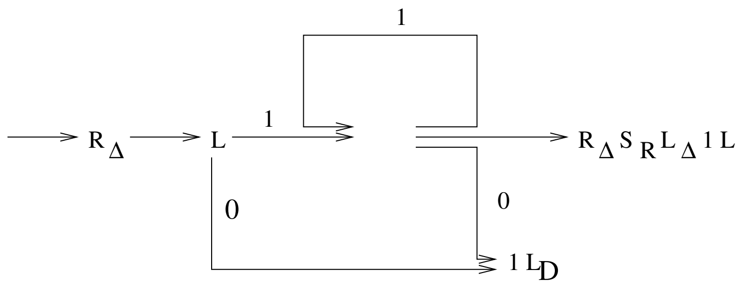
Turing-Computability of Partial Recursive Functions

Proposition 15.1 *Every partial recursive function is Turing-computable.*

Proof

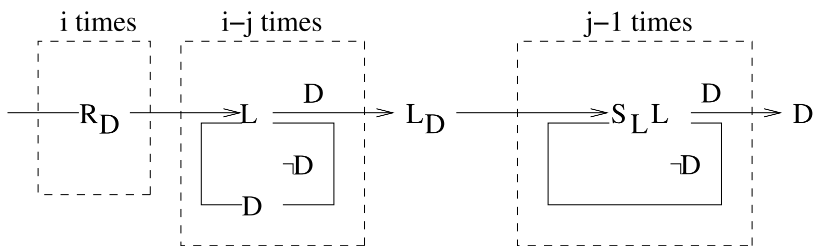
The first step is to show that each of the initial functions is Turing-computable.

A machine to compute



A machine to compute

A machine to compute π^i_j



Turing-Computability of Partial Recursive Functions (cont)

Given that the initial functions are Turing-computable we now need to show that partial functions constructed from them using combination, composition, primitive recursion and minimalization are Turing-computable.

1. Combination. Suppose we have TM's M_1 and M_2 that compute the partial functions f_1 and f_2 respectively.

   Design a three-tape TM M to compute $f_1 \times f_2$ as follows:

   (a) M starts by copying its input from tape 1 to tape 2 and to tape 3;

   (b) M then simulates M_1 using tape 2 and M_2 using tape 3;

   (c) if each of the simulations halts M erases tape 1 then copies the output on tape 2 to tape 1 writes a blank and then follows it with the output from tape 3.

2. Composition. Suppose we have TM's M_1 and M_2 that compute the partial functions f_1 and f_2 respectively. Provided M_1 finishes by leaving its tape head over the leftmost cell (it can always be modified to do this without altering the function it computes) then the machine $\to M_1 M_2$ computes $f_2 \circ f_1$.

3. Primitive recursion. Suppose f is defined by

$$\begin{aligned} f(\bar{x}, 0) &= g(\bar{x}) \\ f(\bar{x}, y+1) &= h(\bar{x}, y, f(\bar{x}, y)) \end{aligned}$$

   where TM's M_1 and M_2 compute the partial functions g and h respectively and return their tape heads to the leftmost tape cell before halting.

   f can be computed by a TM designed to proceed as follows:

Turing-Computability of Partial Recursive Functions (cont)

(a) if the last component of the input is 0 erase it, return the tape head to the leftmost cell and simulate M_1;

(b) if the last component of the input is not 0 then the tape contains a sequence of the form: $\Delta, \bar{x}, \Delta, y+1, \Delta, \Delta \cdots$ for some $y \in \mathbf{N}$:

   i. Using the copying and subtracting machines introduced in Lecture 10 transform the tape contents into the sequence:

   $$\Delta, \bar{x}, \Delta y, \Delta, \bar{x}, \Delta, y-1, \Delta, \bar{x}, \Delta, \cdots, \Delta, \bar{x}, \Delta, 0, \Delta, \bar{x}, \Delta, \Delta \cdots$$

   Position the tape head over the cell following the 0 and simulate M_1.

   ii. The tape will now contain

   $$\Delta, \bar{x}, \Delta y, \Delta, \bar{x}, \Delta, y-1, \Delta, \bar{x}, \Delta, \cdots, \Delta, \bar{x}, \Delta, 0, \Delta, g(\bar{x}), \Delta, \Delta \cdots$$

   which is equivalent to

   $$\Delta, \bar{x}, \Delta y, \Delta, \bar{x}, \Delta, y-1, \Delta, \bar{x}, \Delta, \cdots, \Delta, \bar{x}, \Delta, 0, \Delta, f(\bar{x}, 0), \Delta, \Delta \cdots$$

   Position the tape head over the Δ preceding the last $\bar{x}$ and simulate $M_{2.}$.
   The tape will now contain

   $$\Delta, \bar{x}, \Delta y, \Delta, \bar{x}, \Delta, y-1, \Delta, \bar{x}, \Delta, \cdots, \Delta, \bar{x}, \Delta, 1, \\ \Delta, h(\bar{x}, 0, f(\bar{x}, 0)), \Delta, \Delta \cdots$$

   which is equivalent to

   $$\Delta, \bar{x}, \Delta y, \Delta, \bar{x}, \Delta, y-1, \Delta, \bar{x}, \Delta, \cdots, \Delta, \bar{x}, \Delta, 1, \Delta, f(\bar{x}, 1), \Delta, \Delta \cdots$$

   iii. Repeat the application of M_2 to the end portion of the tape until M_2 is applied to

   $$\Delta, \bar{x}, \Delta y, \Delta, f(\bar{x}, y), \Delta, \Delta \cdots$$

   and the tape is reduced to the form:

   $$\Delta, h(\bar{x}, y, f(\bar{x}, y)), \Delta, \Delta \cdots$$

   which is equivalent to $\Delta, f(\bar{x}, y+1), \Delta, \Delta$.

Turing-Computability of Partial Recursive Functions (cont)

4. Minimalization. Suppose we have a TM M that computes g and we wish to compute $\mu y[g(\bar{x}, y) = 0]$.

   Design a three tape TM which proceeds as follows:

   (a) writes a 0 on tape 2;

   (b) copies $\bar{x}$ from tape 1 onto tape 3 followed by the contents of tape 2;

   (c) simulates M using tape 3;

   (d) if tape 3 contains a 0, erases tape 1, copies tape 2 to tape 1 and halts; otherwise, increments the value on tape 2 by 1, erases tape 3, and returns to step (b) .

Thus we have shown that Turing machines have the power to compute any partial recursive function.

∎

The Partial Recursiveness of Turing Machines – Introduction

- Rather than showing that Turing-computable functions (as we have defined them) are partial recursive we show something even more general – that every computation carried out by a Turing machine can be viewed as the computation of partial recursive function.

  From this it follows that every Turing-computable function is partial recursive.

- To do this we first describe a convention that allows us to interpret the contents of the tape of any Turing machine as a single integer.

- With this convention *every* Turing machine can viewed as computing a function $f : \mathbf{N} \to \mathbf{N}$ **from** the integer corresponding by our convention to the contents of its tape when it starts **to** the integer corresponding to the contents of its tape when it halts.

- We show that any such f is partial recursive by showing that the behaviour of a Turing machine can be precisely modeled by partial recursive functions, i.e. that there are partial recursive functions that mimic the behaviour of the machine and compute precisely what it does.

The Partial Recursiveness of Turing Machines – Introduction (cont)

- How do we interpret the tape contents of a Turing machine as an integer?

  Suppose M is the Turing machine

  $$M = (S, \Sigma, \Gamma, \delta, \iota, h)$$

  We assign each symbol in Γ a unique integer in the range 0 to $|\Gamma|$ (the number of symbols in Γ), reserving 0 for Δ (which we have assumed occurs in every Γ).

  We then interpret the tape contents of M as an integer base $|\Gamma|$ written backwards.

  For example if $\Gamma = \{a, b, c, \Delta\}$ then by interpreting Δ as 0, a as 1, b as 2, c, as 3, a tape containing

  $$\Delta ab\Delta\Delta cba\Delta\Delta\ldots$$

  would be equivalent to

  $$01200321000\ldots$$

  which when written backwards is

  $$\ldots 12300210$$

  which is the base 4 ($|\Gamma| = 4$) representation of

  $$0+(1\times4)+(2\times16)+(0\times64)+(0\times256)+(3\times1024)+(2\times4096)+(1\times16384)$$

  or 27684 in decimal notation.

  We interpret the tape contents as a number written *backwards,* because if we interpreted them as a number written forwards then since we are interpreting the Δ as 0 this would give us meaningless infinitely large numbers (as the infinitely extended unused tape to the right is filled with Δ's).

The Partial Recursiveness of Turing Machines (cont)

Proposition 15.2 *Every Turing machine computation is the computation of a partial recursive function.*

Proof
Let $M = (S, \Sigma, \Gamma, \delta, \iota, h)$ be a Turing machine and let $f : \mathbf{N} \to \mathbf{N}$ be the partial function defined by the computation that M carries out for each $n \in \mathbf{N}$ when given n written backwards in its base $b = |\Gamma|$ representation and interpreting the tape contents similarly when the machine halts. f is not defined when M either does not halt or terminates abnormally.

We show that f is partial recursive.

Proceed as follows:

1. Assign an integer between 0 and $k-1$ to each state of M where $k = |S|$ (the number of states in M), reserving 1 for the start state and 0 for the halt state.

   Note that we now have an integer code for each state in S and each symbol in Γ.

2. Now define these functions which describe M's behaviour. We understand p' to be the uncoded form of state p, x' to be the uncoded form of tape symbol x, etc.

$$mov(p, x) = \begin{cases} 2 & \text{if } \delta(p', x') = (q', R) \\ 1 & \text{if } \delta(p', x') = (q', L) \\ 0 & \text{otherwise} \end{cases}$$

$$sym(p, x) = \begin{cases} y & \text{if } \delta(p', x') = (q', y'), \text{ for } y' \in \Gamma \\ x & \text{otherwise} \end{cases}$$

$$state(p, x) = \begin{cases} q & \text{if } \delta(p', x') = (q', y'), y' \in \Gamma \cup \{L, R\} \\ k & \text{if } p = 0 \text{ or } (p, x) \text{ is not a valid state/symbol pair} \end{cases}$$

Note that all these functions are partial recursive since they are **tabular** functions which we showed to be primitive recursive in Lecture 14.

The Partial Recursiveness of Turing Machines (cont)

3. The configuration of M at any point in a computation may be represented by a triple (w, p, n) where
   - w is the coded (integer) representation of the tape;
   - p is the coded (integer) representation of the machine's current state;
   - n is an integer representing the position of the tape head (number of tape cells from the left, numbering from 1)

   Given this representation we can define a function that given such a coded representation of the current configuration will return the (coded representation of the) current symbol under the tape head.

   $$cursym(w, p, n) = quo(w, b^{n-1}) \dot{-} mult(b, quo(w, b^n))$$

   Note that this function is primitive recursive since it is built up from primitive recursive functions using composition only.

   Example: If as before $\Gamma = \{a, b, c, \Delta\}$ and we interpret Δ as 0, a as 1, b as 2, c, as 3 then a tape containing $\Delta ab\Delta\Delta cba\Delta\Delta\ldots$ would be equivalent to $01200321000\ldots$ which when written backwards is $\ldots 12300210$

   So, suppose $w = 12300210$ and we want to work out the 6-th symbol on the tape – 3 or c. Recall $b = 4$.

$$\begin{aligned} cursym(12300210, p, 6) &= quo(12300210, 4^5) \dot{-} mult(4, quo(12300210, 4^6)) \\ &= 123 \dot{-} mult(4, 12) \\ &= 123 \dot{-} 120 \\ &= 3 \end{aligned}$$

The Partial Recursiveness of Turing Machines (cont)

4. The tools assembled now enable us to define functions which work on the integer coded representations of the machine and tape to tell us, given a triple representing the current configuration, what the next tape head position will be, the next state, and the new tape contents:

$$\begin{aligned} nexthead(w,p,n) &= n \dot{-} eq(mov(p, cursym(w,p,n)), 1) + \\ & \quad eq(mov(p, cursym(w,p,n)), 2) \end{aligned}$$

$$\begin{aligned} nextstate(w,p,n) &= state(p, cursym(w,p,n)) + \\ & \quad mult(k, \neg nexthead(w,p,n)) \end{aligned}$$

$$\begin{aligned} nexttape(w,p,n) &= (w \dot{-} mult(b^n, cursym(w,p,n))) + \\ & \quad mult(b^n, sym(p, cursym(w,p,n))) \end{aligned}$$

In words:

$nexthead(w,p,n)$ works out the next integer tape head position either subtracting 1, if it's a move to the left, or adding 1, if it's a move to the right, or staying the same if there is no move.

$nextstate(w,p,n)$ works out the next integer state code by supplying the $state$ function with the current tape symbol – it will be an illegal state code ($> k$) if the tape head position ever goes to 0 (abnormal termination).

$nexttape(w,p,n)$ works out the next integer representation of the tape contents by removing the current symbol from the coded representation and replacing it with the new symbol as computed by the sym function.

Note again that $nexthead$, $nextstate$ and $nexttape$ are all primitive recursive.

The Partial Recursiveness of Turing Machines (cont)

5. By combining the functions $nexthead$, $nextstate$ and $nexttape$ we can define a new function $step$ that given the current configuration of the machine returns the next configuration of the machine, i.e. a function that simulates a single step of the machine:

   $$step(w, p, n) = nexttape \times nextstate \times nexthead\ (w, p, n)$$

   $step$ is primitive recursive since it is defined from primitive recursive functions using combination.

6. Using $step$ and primitive recursion we can define a function $run(w, p, n, t)$ that given a configuration triple (w, p, n) and an integer t produces the configuration triple describing the machine's configuration after it has executed t steps – i.e. it mimics t transitions.

   In the base case, $t = 0$, the configuration does not change:

   $$run(w, p, n, 0) = (w, p, n)$$

   After $t + 1$ transitions the machine will be one $step$ beyond where it was after t transitions:

   $$run(w, p, n, t + 1) = step(run(w, p, n, t))$$

   run is primitive recursive since it is defined from primitive recursive functions using primitive recursion.

The Partial Recursiveness of Turing Machines (cont)

7. The output of the function computed by the Turing machine is the value on the tape when the machine halts, i.e. when state 0 (in our coded representation scheme) is reached.

   The number of transitions or steps required to reach this state is the smallest t such that the second element in the triple computed by $run(w, p, n, t)$ is 0 (i.e. such that the halt state is reached). This can be expressed as

   $$\mu t[\pi_2^3(run(w, p, n, t)) = 0]$$

   So, we can define a partial recursive function $stoptime$ that tells us the number of steps from the start state with input w to the halt state:

   $$stoptime(w) = \mu t[\pi_2^3(run(w, 1, 1, t)) = 0]$$

8. Finally, f, the function that M computes will be the function

   $$f(w) = \pi_1^3(run(w, 1, 1, stoptime(w)))$$

   i.e. what is left on the tape after the machine is run with input w for $stoptime(w)$ transitions.

   Clearly f is partial recursive.

∎