Chapter 5: Church's Thesis

A. Introduction

- What do we know about TMs?

- **Church's Thesis:**
- Notes:
  - Alternate version: "...the mathematically defined class of *partial recursive* (μ-recursive) functions is exactly the intuitively conceived class of effectively, that is algorithmically, computable functions." (Machtey & Young, 1978).
  - A "thesis" not a "theorem":
    * based on an intuitive concept.
    * could conceivably be disproved.
    * the evidence for it includes history and the equivalence to TMs of several different models of computation.
- Goals for the chapter:
  - Build evidence for Church's Thesis by looking at two alternate models for computation:
  - Answer the question: "What about 'programmable' TMs?"

B. Grammars and Turing machines

1. Recall definition of a grammar $G = (V, \Sigma, R, S)$ where

   V is an alphabet,
   $\Sigma \subset V$ is a set of terminals; $V - \Sigma$ are nonterminals,
   $R \subset (V^*(V - \Sigma)V^*) \times V^*$ is a finite set of rules,
   $S \in V - \Sigma$ is start symbol.

   Recall also the idea of a **derivation** and the language $L(G)$ **generated** by G.

2. Simulating a TM by a grammer
   - View each configuration (q, u, a, v) (or $(q, u\underline{a}v)$) of M as a string $[uqav]$.
   - Define a grammar such that $\sigma_1 \Rightarrow \sigma_2$ iff the configuration corresponding to σ_1 yields the configuration corresponding to σ_2.

- From the proof of Lemma 5.2.1,
  a. If $\delta(q,a) = (p,b)$ and $b \in \Sigma$ then the grammar contains _____.
  b. If $\delta(q,a) = (p,R)$ then we have _____for each $b \in \Sigma$, and _____.
  c. If $\delta(q,\#) = (p,L)$ then _____for all $b,c \in \Sigma$, and _____for all $b \in \Sigma$.
  d. If $\delta(q,a) = (p,L)$ and $a \neq \#$ then _____for all $b \in \Sigma$ and $c \in \Sigma \cup \{[]\}$.
- Example:
- Note: start symbol of grammar has nothing to do with this; we are simply using grammars as re-write systems.

3. Grammars as computers of functions
   - **Definition**. A function $f : \Sigma_0^* \to \Sigma_1^*$ is **grammatically computable** iff there is a grammar $G = (V, \Sigma, R, S)$ and strings $x, y, x', y' \in V^*$ such that
   - Example:
   - **Theorem 5.2.1** Every Turing-computable function is grammatically computable.

C. Primitive recursive functions

1. Introduction
   - Now back to viewing computation as evaluating functions from $\mathcal{N}^k$ to $\mathcal{N}$.
   - The plan
     - carefully define a class of (intuitively) computable functions called μ-recursive;
     - start with a simpler class called primitive recursive;
     - show that every grammatically computable function is μ-recursive;
     - show that every μ-recursive function is Turing computable.
2. Definitions
   - A function is **primitive recursive** if it is an **initial function** or can be generated from other primitive recursive functions by **composition** or **primitive recursion**.
   - **Initial functions:**
     - The **zero function** $\zeta : \mathcal{N}^0 \to \mathcal{N}$,
     - The **projection function** $\pi_i^k : \mathcal{N}^k \to \mathcal{N}$,
     - The **successor function** $\sigma : \mathcal{N} \to \mathcal{N}$,
   - **Combining operations:**
     - **Composition:** $f(\bar{n}) = g(h_1(\bar{n}), \ldots, h_l(\bar{n}))$, where $\bar{n} = (n_1, \ldots, n_k)$; $f, h_1, \ldots, h_l$ are k-place functions; and g is an l-place function. We say "f is generated from g and $h_1, \ldots, h_l$ by composition."
     - **Primitive recursion:** $f(\bar{n}, m+1) = h(\bar{n}, m, f(\bar{n}, m))$, where $\bar{n} \in \mathcal{N}^k$, $m \in \mathcal{N}$, and $f(\bar{n}, 0) = g(\bar{n})$. We say "$f$ is generated from g and h by primitive recursion."
   - Claim: every primitive recursive function is "computable" in the intuitive sense.

3. Examples

4. Building up the set of primitive recursive functions

   a. Constant functions

      - Definition. **Constant function**
      - Lemma. The constant functions are primitive recursive.

   b. Explicit transformation

      - Definition. An l-place function f is obtained from a k-place function g by **explicit transformation** if $f(\bar{n}) = g(h_1(\bar{n}), \ldots, h_k(\bar{n}))$, where each of the h_i are either l-place projection or constant functions.
      - Lemma. Any function obtained from a primitive recursive function by explicit transformation is primitive recursive.

   c. Bounded product

      - Definition. A function f is obtained from a function g by **bounded product** if
      - Lemma. Any function obtained from a primitive recursive function by bounded product is primitive recursive.

   d. Predicates

      - Definitions:
        - A **predicate** $P \subseteq \mathcal{N}^k$, $k \geq 0$, is a relation on $\mathcal{N}$. We write $P\bar{n}$ if $\bar{n} \in P$.
        - The **characteristic function** of a predicate $P \subseteq \mathcal{N}^k$ is a k-place function f satisfying
        - A predicate is primitive recursive iff its characteristic function is.
      - Examples.
      - Lemma(s). If Q and R are primitive recursive predicates, then so is
        - any predicate obtained from Q by explicit transformation,
        - the conjunction $Q\bar{n}$ and $R\bar{m}$,
        - the disjunction $Q\bar{n}$ or $R\bar{m}$,
        - the negation of Q.

- Lemma. If $P_1, \ldots, P_l$ are primitive recursive predicates, where for every $\bar{n} \in \mathcal{N}^k$ exactly one of $P_1\bar{n}, \ldots, P_l\bar{n}$ is true, and if $g_1, \ldots, g_l$ are primitive recursive functions, then the following function f is also primitive recursive:

$$f(\bar{n}) = \begin{cases} g_1(\bar{n}) & \text{if } P_1\bar{n} \\ \vdots & \\ g_l(\bar{n}) & \text{if } P_l\bar{n} \end{cases}$$

e. Bounded quantification and minimalization.

- Definition. The **bounded existential quantification** of a $(k+1)$-place predicate Q is the $(k+1)$-place predicate R satisfying
- Definition. The **bounded universal quantification** of a $(k+1)$-place predicate Q is the $(k+1)$-place predicate S satisfying
- Examples.
- Lemma. If Q is a primitive recursive predicate then so are its bounded existential and universal quantifications.
- Definition. The **bounded minimalization** of a $(k+1)$-place predicate Q is the $(k+1)$-place function f satisfying:

$$f(\bar{n}, m) = \begin{cases} \text{smallest } p \leq m \ni Q\bar{n}p \\ 0, \text{ if } \nexists p \leq m \ni Q\bar{n}p \end{cases}$$

- Example.
- Lemma. If Q is a primitive recursive predicate then its bounded minimalization is a primitive recursive function.

D. Gödel numbering

1. The problem. We need a "numbering" for strings over a finite alphabet, because we want to relate grammatically computable functions to primitive recursive functions.
2. The solution: Gödel numbers. If $\Delta = \{d_1, \ldots, d_{\beta-1}\}$ is an alphabet and $w = d_{i_1} \cdots d_{i_k}$, then let $gn : \Delta^* \to \mathcal{N}$ be defined by
3. An example:
4. The inverse function $gn^{-1} : \mathcal{N} \to \{\varepsilon\} \cup \Delta(\Delta \cup \{d_0\})^*$, where d_0 is a new symbol needed to correspond to the digit 0.

5. Some primitive recursive functions and predicates corresponding to functions on strings and derivations in grammars. Given a grammar $G = (V, \Sigma, R, S)$, let $\Delta = V \cup \{\$\}$, the following can be shown to be primitive recursive:
   - $\text{length}(n) = |gn^{-1}(n)|$
   - $\text{concat}(m, n) = gn(gn^{-1}(m)gn^{-1}(n))$
   - $\Delta^* n$ iff $gn^{-1}(n) \in \Delta^*$ (member of Δ^*)
   - Ynm iff $gn^{-1}(n), gn^{-1}(m) \in V^*$ and $gn^{-1}(n) \Longrightarrow gn^{-1}(m)$ (yields in one step)
   - Dn iff $gn^{-1}(n) = \$x_0\$x_1\$\cdots\$x_k\$$ where $x_i \in V^*$ and $x_i \Longrightarrow x_{i+1}$ (is derivation)

E. μ-recursive functions

1. Question. Does the set of primitive recursive functions include all that are computable in the intuitive sense? No, since ...
   - We can list all primitive recursive functions.
   - We can define a function that is clearly computable but is not in the list (diagonalization).

2. Definitions.
   - The **unbounded minimalization** of a $(k+1)$-place function g is the k-place function f defined by

$$f(\bar{n}) = \begin{cases} \text{the least } m \text{ such that } g(\bar{n}, m) = 0, \\ \quad \text{if such an } m \text{ exists;} \\ 0 \quad \text{otherwise.} \end{cases}$$

   - A $(k+1)$-place function g is **regular** iff for every $\bar{n} \in \mathcal{N}^k$ there is an m such that $g(\bar{n}, m) = 0$.
   - A function from $\mathcal{N}^k$ to $\mathcal{N}$ is **μ-recursive** iff it is primitive recursive or it can be obtained by unbounded minimalization from a regular primitive recursive function.
   - A function $f : \Sigma_0^* \to \Sigma_1^*$ is **μ-recursive** iff the function $f' : \mathcal{N} \to \mathcal{N}$ is μ-recursive, where

$$f'(n) = \begin{cases} gn(f(gn^{-1}(n))) & \text{if } \Sigma_0^* n; \\ 0 & \text{else.} \end{cases}$$

3. Remarks.
   - The unbounded minimalization f of a primitive recursive function g might not be primitive recursive, or even intuitively computable.
   - For f to be intuitively computable, we have to know that eventually we will find an m such that $g(\bar{n}, m) = 0$. This is only true if g is regular.
   - If f is μ-recursive, we are claiming it is computable in the intuitive sense.
   - There is no foolproof (finite) way to check if a function is regular. If there were, we could construct a list of all μ-recursive functions, and then by diagonalization construct a new function that isn't in the list!
   - This strange fact anticipates a famous conflict in formal mathematical systems in general: consistency vs. completeness. If you claim to have a formal system for listing all well-defined functions, then either your list isn't complete (there are some functions not on the list) or it isn't consistent (some of the functions on your list aren't well-defined).

4. Theorem 5.5.1. Every grammatically computable function from strings to strings or from numbers to numbers is μ-recursive.

   Idea of the proof: define a bunch of μ-recursive functions, using unbounded minimalization at one key point, and simulate the steps in a derivation with these functions.

5. Theorem 5.6.1. Every μ-recursive function from numbers to numbers or from strings to strings is Turing-computable.

   Idea of the proof:
   - show initial functions are Turing-computable.
   - show composition generates Turing-computable functions.
   - show primitive recursion generates Turing-computable functions.
   - show unbounded minimalization (of regular functions) generates Turing-computable functions.
   - to complete proof for μ-recursive functions from strings to strings, need TM that converts strings to (Gödel) numbers, a TM which computes on the numbers, and a TM which converts back to strings.

F. Universal Turing machines

1. Question. Can we include the idea of programmability (i.e., the "stored program" computer, or "programs as data") in any of our models of computation?

2. Need an encoding (over a fixed alphabet) for Turing machines.
   - Can define a unique encoding $\rho(M)$ for every TM using only the alphabet $\{c, I\}$.
   - Also need an encoding for input strings $\rho(w)$.

3. Can define a **universal machine** U such that, for any TM M, M halts on input w with output w' iff U halts on input $\rho(M)\rho(w)$ with output $\rho(w')$.