Harvard CS 121 and CSCI E-207

Lecture 15: Decidability, Recognizability, and a Universal Turing Machine

Harry Lewis

October 26, 2010

- Reading: Sipser §4.1.

Equivalent Formalisms

Many formalisms for computation are equivalent in power to the TM formalism:

- TMs with multiple tapes, multiple heads, 2-dimensional tapes, addressable storage (RAMs) . . .
- TMs with nondeterminisim, randomness, . . .
- General Grammars
- Church's λ-calculus (μ-recursive functions)
- Markov algorithms
- Your favorite high-level programming language (C, Lisp, Java, . . .)
- . . .

Reduction of TMs to 2-CMs

A 2-counter machine (2-CM) has:

- A finite-state control
- Two counters, i.e. $C1$ and $C2$, which are registers containing integers ≥ 0 with only 3 operations:
  - Add 1 to $C1/C2$
  - Subtract 1 from $C1/C2$
  - Is $C1/C2 = 0$?

Theorem: For any TM, there is an equivalent 2-CM, in the sense that if you start the 2-CM with an encoding of the TM tape in its counters it will eventually halt with an encoding of what the TM computes.

Simulating a TM tape with 2 pushdown stores: Split the tape at the head position into two stacks

Moving TM head to left	$\equiv$	Pop from stack # 1 Push onto stack # 2
Moving TM head to right	$\equiv$	Pop from stack # 2 Push onto stack # 1
Change scanned symbol	$\equiv$	Change top of stack # 1

(So 2-PDAs are as powerful as TMs)

Simulating One Stack with Two Counters:
Think of the stack as number in a base = $|\Sigma| + 1$

[Assume ≤ 9 stack symbols]

Pop the stack	$\equiv$	Divide by 10 and discard the remainder
Push a_9	$\equiv$	Multiply by 10 and add 9
Is stack top $= a_3$?	$\equiv$	Is counter mod $10 = 3$?

$\rightarrow$ All of these can be calculated using a second counter.

Simulating Four Counters With Two: $(p, q, r, s) \to 2^p 3^q 5^r 7^s$

Add 1 to $C1$	$\equiv$	$p \leftarrow p + 1$
	$\equiv$	Double $C1'$
Is $C3 \neq 0$?	$\equiv$	$r \neq 0$?
	$\equiv$	Does 5 divide $C1'$ evenly?
Subtract 1 from s	$\equiv$	Divide $C1'$ by 7

The Church-Turing Thesis

The equivalence of each to the others is a mathematical theorem.

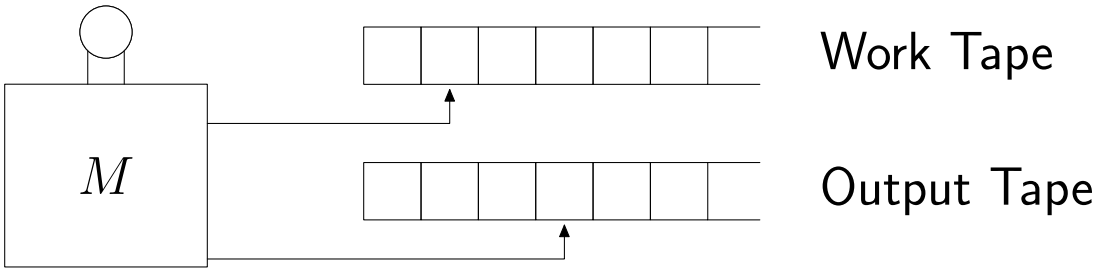
That these formal models of algorithms capture our intuitive notion of algorithms is the **Church–Turing Thesis**.

- Church's thesis $=$ partial recursive functions, Turing's thesis $=$ Turing machines
- Is the Church-Turing Thesis Provable?

Another TM Variant: Enumerators

Def: A TM M enumerates a language L if M, when started from a blank tape, runs forever and "emits" all and only the strings in L.

(For example, by writing the string on a special tape and passing through a designated state.)



Recognizable $\equiv$ enumerable

Theorem: L is Turing-recognizable iff L is enumerated by some TM.

Proof:

($\Rightarrow$) Suppose $L(M) = L$. We want to construct a TM M' that enumerates L.

M' dovetails all of the computations by M:

1. Do 1 step of M's computation on w_0
2. Do 2 steps of M on w_0 and w_1
3. Do 3 steps on each of w_0, w_1, w_2

where $w_0, w_1, \ldots =$ lexicographic enumeration of Σ^*.

Outputting any strings w_i whose computations have accepted.

Recognizable $\equiv$ enumerable, finis

($\Leftarrow$)

- The Turing-decidable sets are often called *recursive* because they can be computed using certain systems of recursive equations, rather than via TMs.
- The Turing-recognizable sets are usually called *recursively enumerable*, i.e. “computably enumerable.”

Decidability

- Recall that a language $L \subseteq \Sigma^*$ is decidable if there is a TM that always halts when started on an input in Σ^*, in either q_{accept} if $w \in L$ or q_{reject} if $w \notin L$.

Enumerability in order $\equiv$ decidability

Theorem: L is decidable iff L is enumerable in lexicographic order.

(lexicographic order has shorter strings before longer, and alphabetic order among strings of the same length)

Proof:

Asking questions about arbitrary finite automata

- **Proposition:** Every regular language is decidable.

  Proof: (By “coding” a DFA as a TM.)

What if the DFA D is part of the input?

- That is, can we design a single TM that, given two inputs, D and w, decides whether D accepts w?
  - The TM needs to use a fixed alphabet & state set for all inputs D, w.
- **Q:** How to represent $D = (Q, \Sigma_D, \delta, q_0, F)$ and w?
  List each component of the 5-tuple, separated by |'s.
  - Represent elements of Q as binary strings over $\{0, 1\}$, separated by ,'s.
  - Represent elements of Σ_D as binary strings over $\{0, 1\}$, separated by ,'s.
  - Represent $\delta : Q \times \Sigma_D \to Q$ as a sequence of triples (q, σ, q'), separated by ,'s, etc.

  We denote the encoding of D and w as $\langle D, w \rangle$.

A "Universal" algorithm for deciding regular languages

- **Proposition:** $A_{\text{DFA}} = \{\langle D, w\rangle : D \text{ a DFA that accepts } w\}$ is decidable.

  Proof sketch:

  - First check that input is of proper form.
  - Then simulate D on w. Implementation on a multitape TM:
    - Tape 2: String w with head at current position (or to be precise, its representation).
    - Tape 3: Current state q of D (i.e., its representation).

- Could work with other encodings, e.g. transition function as a matrix rather than list of triples.

Representation independence

- **General point:** Notions of computability (e.g. decidability and recognizability) are independent of data representation.
  - A TM can convert any reasonable encoding to any other reasonable encoding.
  - We will use $\langle \cdot \rangle$ to mean "any reasonable encoding".
  - We'll need to revisit representation issues again when we discuss computational *speed.*
  - For the moment when we are interested only in whether problems are decidable, undecidable, recognizable, etc., so we can be content knowing that there is *some* representation on which an algorithm could work.

More Decidable Problems

- $\{\langle R, w\rangle : R$ is a regular expression that generates $w\}$.
- $\{\langle X\rangle : X$ is an DFA/NFA/RE such that $L(X) = \emptyset\}$.
- $\{\langle X\rangle : X$ is a DFA/NFA/RE such that $|L(X)| = \infty\}$.
- $\{\langle M, w\rangle : M$ is a PDA that accepts $w\}$.
- Any given context-free language (what does this question mean?)

A Universal Turing machine

Theorem: There is a Turing machine U, such that when U is given $\langle M, w \rangle$ for any TM M and w, U produces the same result (accept/reject/loop) as running M on w.

Proof: Initially,

- First tape contains $\langle M \rangle$, including in particular its transition function δ_M.
- Second tape contains $\langle w \rangle$.
- Third tape contains $\langle q_{\text{start}} \rangle$.
- Simulate steps of M by multiple steps of U.

(Brief return to implementation description.)

$\Rightarrow$ Turing machines can be "programmed".

Consequences of the Existence of Universal Turing Machines

- **Corollary:** $A_{\mathsf{TM}} = \{\langle M, w\rangle : M \text{ accepts } w\}$ is Turing-recognizable.
- **Corollary:** $HALT_{\mathsf{TM}} = \{\langle M, w\rangle : M \text{ eventually halts on } w\}$ ("The Halting Problem") is Turing-recognizable.
- **Corollary:** "The Turing Machines that halt on some input are an r.e. set" (What does this mean?)
- **Q:** Are these sets decidable?
- **Q:** Are there undecidable languages?