Harvard CS 121 and CSCI E-207
Lecture 15: Recognizability & Decidability

Salil Vadhan
(lecture given by Bo Waggoner)

October 25, 2012

• Reading: Sipser §3.2, §3.3, §4.1.
Nondeterministic TMs

• Like TMs, but $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$

• It mainly makes sense to think of NTMs as recognizers

$L(M) = \{w : M \text{ has some accepting computation on input } w\}$

Example: NTM to recognize
$\{w : w \text{ is the binary notation for a product of two integers } \geq 2\}$
NTMs recognize the same languages as TMs

- Given a NTM $M$, we must construct a TM $M'$ that determines, on input $w$, whether $M$ has an accepting computation on input $w$.

- $M'$ systematically tries
  - all one-step computations
  - all two-step computations
  - all three-step computations
  - ...
Enumerating Computations by Dovetailing

• There is a bounded number of $k$-step computations, for each $k$.
  (because for each configuration there is only a constant number of “next” configurations in one step)

• Ultimately $M'$ either:
  • discovers an accepting computation of $M$, and accepts itself,
  • or searches forever, and does not halt.
Dovetailing Details

• Suppose that the maximum number of different transitions for a given \((q, a)\) is \(C\).

• Number those transitions \(1, \ldots, C\) (or less)

• Any computation of \(k\) steps is determined by a sequence of \(k\) numbers \(\leq C\) (the “nondeterministic choices”).

• How \(M'\) works: 3 tapes

  #1 | Original input to \(M\)
  #2 | Simulated tape of \(M\)
  #3 | 1213 ⊔· · · Nondeterministic choices for \(M'\)
Simulating one step of $M$

- Each major phase of the simulation by $M'$ is to simulate one finite computation by $M$, using tape #3 to resolve nondeterministic ambiguities.

- Between major phases, $M'$
  - erases tape #2 and copies tape #1 to tape #2
  - Replaces string in $\{1, \ldots, C\}^*$ on tape #3 with the lexicographically next string to generate the next set of nondeterministic choices to follow.

- **Claim:** $L(M') = L(M)$

- **Q:** Slowdown?
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 ≡ 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 \(\Sigma^*\).

Outputting any strings \(w_i\) whose computations have accepted.
Recognizable ≡ 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,” due to the above characterization in terms of enumerators.

- **Fact (ps7):** \(L\) is decidable iff it is enumerable in *lexicographic order.*
Three basic facts on the recursive vs. r.e. languages

1. If $L$ is recursive, then $L$ is r.e.
   
   Proof:

2. If $L$ is recursive then so is $\overline{L}$.
   
   Proof:

3. $L$ is recursive if and only if both $L$ and $\overline{L}$ are r.e.
   
   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 $\Sigma_D$ as binary strings over \{0, 1\}, separated by ,’s.
- Represent $\delta : Q \times \Sigma_D \rightarrow Q$ as a sequence of triples $(q, \sigma, q')$, 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 representations.

• 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.