Yet More Undecidable Languages

CS154

Chris Pollett

May 9, 2007.

Outline

- Emptiness Testing (reductions to a co-r.e. set)
- Reductions via Computation Histories

Reductions from a co-R.E. Set

- Using reducibility is the most common way to show a language is undecidable.
- As another example, consider the language:
- $E_{TM} = \{<M> \mid M \text{ is a TM and } L(M)=\emptyset\}$.
  Theorem. E_{TM} is undecidable.
  Proof. First consider the following machine:
  M1= “ On input x:
  1. If $x \neq w$, reject.
  2. If $x = w$, run M on input w and accept if M does.”
  This machine is a modification of M and it accepts at most one input w, and it only accepts this if M does. Now suppose machine R decided E_{TM}. Then we could build the following machine to decide A_{TM} giving a contradiction:
  S = “On input <M,w>, an encoding of a TM M and a string w:
  1. Use the description of M and w to make a corresponding machine M1 as above.
  2. Run R on input <M1>
  3. If R accepts, reject; if R rejects, accept.”
- In this example, the map <M,w> --> <M1> can be computed by a TM, and it reduces $\overline{A}_{TM}$ to E_{TM}

Reductions via Computation Histories

- Consider the following language:

  R := {$<w,M, x>$ | w is the code of a sequence of configurations, beginning with a start configuration of M on x, where each configuration yields the next according to the transition table of TM M on input x. Further, the last configuration is accepting.}
- This language is decidable (we check whether the first configuration does match a starting configuration of M on x, then we examine pairs of adjacent configurations and see if one follows from the other, etc).
- Notice $A_{\mathrm{TM}} := \{<M, x> \mid \exists w <w,M, x> \in R \}$.
- The string w in the above can be viewed as a computation history.
- Such histories are often useful in doing reductions from one problem to another.

Formal Definition of a Computation History.

- Let *M* be a TM and *x* an input string.
- An **accepting computation history** for *M* on *x* is a sequence of configurations $C_1,.., C_k$, where C_1 is the start configuration of *M* on *x*, C_k is an accepting configuration of *M*, and each C_i legally follows from C_{i-1} according to the rules of *M*.
- A **rejecting computation history** for *M* is defined similarly, except that C_k is a rejecting configuration.

Linear Bounded Automata

- We will next work towards using Computation Histories to give undecidability proofs.
- Our first example will involve a new machine model which has strength between a PDA and a TM.
- A **linear bounded automata (LBA)** is a restricted type of TM wherein the tape head isn't permitted to move off the portion of the tape containing the input.
- If an LBA tries to move off this part of the tape to the right, the tape head stays where it is.

Strength of LBAs

- One can verify that each of the TMs we gave for the languages A_{DFA}, A_{CFG}, E_{DFA}, and E_{CFG} are either LBAs or easily modified into LBAs.
- For example, E_{CFG} involved marking each terminal, then marking a variable A if it appear in a $A \rightarrow B_1 \ldots B_n$ and the B_i's had already been marked. Finally, one checks if the start variable has been marked.
- This marking can be done without using any more tape squares so the above can be done by an LBA.

A Useful Lemma about LBAs

Lemma. Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.

Proof. A configuration consists of the state of the control of the LBA, the position of the tape head, and the contents of the tape. So there are q possibilities for the state, the head can be in one of at most n positions, each of the n tapes squares could have one of g symbols written in it (so g^n possibilities). All together this gives, qng^n.

Decidability and LBAs

Theorem. A_{LBA} is decidable.

Proof. The algorithm that decides A_{LBA} is as follows:

L="On input <*M*, *w*>, where *M* is an LBA and *w* is a string:

1. Simulate *M* on *w* for qng^n steps or until it halts.
2. If *M* has halted, *accept* if it accepted; and *reject* if it rejected. If it has not halted *reject*."

LBAs and Undecidability

- In contrast to the last theorem above, not all problems about LBAs are decidable:

Theorem E_{LBA} is undecidable.

Proof. The reduction is from A_{TM}. We show if E_{LBA} is decidable then A_{TM} also would be decidable. Let $L=\{w \mid w$ is a string of the form $C_1\#C_2..\#C_k$ given a legal accepting computation history of M on input x$\}$. One can show that L can be recognized by an LBA; let's call it B. Further, if L is empty, $\langle M, x\rangle$ is not in A_{TM}. So if E_{LBA} were decidable the following would be a decision procedure for A_{TM}:

S= "On input $\langle M,x\rangle$, where M is a TM and x is a string:

1. Construct LBA B from M on x as described in the proof idea.
2. Run R on input $\langle B\rangle$.
3. If R rejects, *accept*; if R accepts, *reject*."