Yet More Reducibility

CS154

Chris Pollett

Apr 26, 2006.

Outline

- More on Reductions via Computation Histories
- Start on Post Correspondence Problem

ALL_{CFG}

- Computation Histories can be used to show some problems about CFGs are undecidable.
- Let ALL_{CFG} be the language $\{\langle G\rangle \mid G$ is a CFG and $L(G) = \Sigma^*\}$.

Theorem. ALL_{CFG} is undecidable.

Proof. Assume ALL_{CFG} is decidable by machine N, we will show how you could use this along with computation histories to decide A_{TM}. Recall for the E_{LBA} reduction we showed that an LBA can recognize the language:

$L = \{w \mid w = C_1\#C_2..\#C_k$ is a accepting computation history of Turing Machine M on input $x\}$.

This language is not context-free (can prove this using the Pumping lemma). The basic is we'd like to verify pairs of configurations to see one follows the next, but if we were using a PDA and tried to push the characters of C_i onto the stack then when we pop them off they'd be in the wrong order to do the verification. Let u^R denote the string consisting of the characters of u in reverse order To solve our problem we redefine a computation history so it has the format $w = C_1\#C_2^R\# C_3 ..\#C_k$. Let L now mean the language before but with this definition…

More on ALL_{CFG}

(**Proof cont'd**) of computation history. Can show there is a PDA which given a string w can check:

1. it *does not* start with C_1; or that,
2. it *does not* end with an accepting configuration; or that,
3. there is an i such that C_i does not properly yield C_{i+1}.

i.e., there is a PDA that can recognize the complement of L. We can convert this PDA to some CFG G. Now we can give a Turing Machine for A_{TM} as follows:

S="On input <M, x> where M is a TM and x a string:

1. Construct the CFG G from M and x to recognize $\overline{L}$.
2. Run N (our decision proocdure for ALL_{CFG}) on <G>
3. If N rejects, **accept**; if N accepts, **reject**."

Post Correspondence Problem

- We now are going to show that there are problems other than problems for machines which are undecidable.
- We are going to consider problems involving dominos. These are pairs of strings [s|t].
- Given a set of dominoes $\{[s_1|t_1], \ldots [s_k|t_k]\}$ we want to know if we can arrange a subset of them (we allow repeats) so that $s_{i\_1},..s_{i\_j} = t_{i\_1},..t_{i\_j}$. This is called a **match** and the whole problem is called **Post's Correspondence Problem** (**PCP**).
- For example, given {[b|ca], [a|ab], [ca|a], [abc|c]}, the following is a match [a|ab][b|ca][ca|a][a|ab][abc|c].
- We can associate a language to this problem as:

  PCP = {<*P*> | *P* is an instance of Post's Correspondence Problem with a match}