Reducibility

CS154

Chris Pollett

Apr 19, 2006.

Outline

- Reducibility

Reducibility

- We next consider what other problem are undecidable.
- Our approach to showing languages are undecidable will be to use a notion called **reducibility**.
- A **reduction** r is a mapping from possible inputs I_A to a problem A, **instances** of A, to instances of problem B, with the property that $I_A \in A$ if and only if r(I_A)$\in B$.
- If the reduction can be computed by a TM, i.e., a **Turing reduction**, then if B is decidable then A will be too. Conversely, if A is not decidable, then B also won't be decidable.

Example

- Let $\text{HALT}_{\text{TM}} = \{<M,w> \mid M$ is a TM and M halts on input $w\}$.

Theorem. HALT_{TM} is undecidable.

Proof. Suppose *R* decides HALT_{TM} . From *R* we can construct a machine *S* which decides A_{TM} as follows:

S =“ On input <*M, w*> an encoding of a TM *M* and a string w:

1. Run TM *R* on input <*M, w*>
2. If *R* rejects, reject.
3. If *R* accepts, simulate *M* on *w* until it halts.
4. If *M* has accepted, then accept; if *M* has rejected, reject.”

So if *R* works *S* will decide A_{TM}. Therefore *R* can exist.

Another Example

- 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:

M_1= " 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 M_1 as above.
2. Run R on input $<M_1>$
3. If R accepts, reject; if R rejects, accept."

A Problem about Regular Languages

- Even problems about regular languages can sometimes be hard. Let:
  Regular$_{TM}$ = {<M> | M is a TM and L(M) is a regular language}.

Theorem. Regular$_{TM}$ is undecidable.

Proof. Suppose R decides Regular$_{TM}$. Then the following machine decides A_{TM}:

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

1. Construct the following machine M_2:

   M_2 = "On input *x*:

   - If *x* has the form 0^n1^n, accept.
   - If *x* does not have this form, run *M* on input *w* and accept if *M* accepts *w*."

   // So if *M* accepts *w*, then M_2 accepts all strings; otherwise, M_2 only accepts strings of the form 0^n1^n.

2. Run *R* on input $<M_2>$.
3. If *R* accepts, accept; otherwise, if *R* rejects, reject."

Using reducibility from languages other than A_{TM}

- We don't need to only use A_{TM} now to show a language is undecidable.
- For instance, if some E_{TM} reduces to some language A, then A will be undecidable. For example, let
  $EQ_{TM}=\{<M_1, M_2> \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Theorem. EQ_{TM} is undecidable.

Proof. Suppose R decides EQ_{TM} , then we can build an S solving E_{TM} as follows (hence, giving a contradiction):

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

1. Run R on $<M, M_1>$, where M_1 is the machine that rejects all inputs.
2. If R accepts, accept; otherwise if R rejects, reject."