Lecture 33: Reductions and Undecidability

CSCI 81
Spring, 2012

Kim Bruce

SD & Turing Enumerable

- Theorem: A language is SD iff it is Turing enumerable.
  - Proof: Spose L is Turing enumerable. Show L is SD.
    - Let w be input. Start enumerating L. Every time enter state p, check to see if contents of tape is w. If yes then halt and stop. Otherwise keep going.
  - "Obvious" proof in other direction not work!

SD & Turing Enumerable

- Theorem: A language is in SD iff it is Turing enumerable.
  - Proof (*cont.*): Spose L is in SD. Show L can be enumerated.
    - Enumerate all $w \in \Sigma^*$ lexicographically: ε, a, b, aa, ab, ba, bb, ...
    - As each w_i is enumerated, start a copy of M to check with w_i as input.
    - Execute one step of each M with w_i started, excluding those that have halted
    - Whenever an M accepts a w_i, output w_i.
- Called dove-tailing

Lexicographically Enumerable

- M *lexicographically enumerates* L iff M enumerates the elements of L in lexicographic order.
- A language L is *lexicographically Turing-enumerable* iff there is a Turing machine that lexicographically enumerates it.

Lexicographically Enumerable

- Theorem: A language is in D iff it is lexicographically enumerable.
  - Proof: Spose L is in D. Show L can be enumerated lexicographically.
    - Enumerate all $w \in \Sigma^*$ lexicographically: ε, a, b, aa, ab, ba, bb, ...
    - As each w_i is enumerated, run M deciding L on w_i as input.
    - If M accepts a w_i, output w_i. Otherwise go to next.
    - Easier here because M always halts

Lexicographically Enumerable

- Theorem: A language is in D iff it is lexicographically enumerable.
  - Proof: Spose L can be enumerated lexicographically. Show L is in D.
  - To determine if w in L:
    - Start enumerating all elts of L lexicographically
    - If w is enumerated, then accept.
    - If go past w in lexicographic order, then reject
    - If halts before getting to w, then also reject.

Oops!

- Second part of proof has a hole.
  - Suppose M lexicographically enumerates L = {a,ba,aba} by enumerating three elements and then continuing forever without ever enumerating another element. If w comes after last element, then won't know if in or not!
  - Can only happen if L is finite.
  - But all finite languages are decidable.
  - Fixes proof
- But don't necessarily know whether L is finite!!

Undecidable Problems

The Problem View	**The Language View**
Does TM *M* halt on *w*?	H = {<*M*, *w*> : *M* halts on *w*}
Does TM *M* not halt on *w*?	¬H = {<*M*, *w*> : *M* does not halt on *w*}
Does TM *M* halt on the empty tape?	H_ε = {<*M*> : *M* halts on ε}
Is there any string on which TM *M* halts?	H_{ANY} = {<*M*> : there exists at least one string on which TM *M* halts }
Does TM *M* accept all strings?	A_{ALL} = {<*M*> : $L(M) = \Sigma^*$}
Do TMs M_a and M_b accept the same languages?	EqTMs = {<M_a, M_b> : $L(M_a) = L(M_b)$}
Is the language that TM *M* accepts regular?	TM_{reg} = {<*M*>:$L(M)$ is regular}

Text Reduction Example

- Theorem: There exists no general procedure to solve the following problem:
  - Given an angle A, divide A into sixths using only a straightedge and a compass.
- Proof:
  - Suppose that there were such a procedure, call it sixth. Then we could trisect an arbitrary angle using that procedure as follows ...
  - ... but know can't trisect an angle. Therefore sixth cannot exist .

Reductions

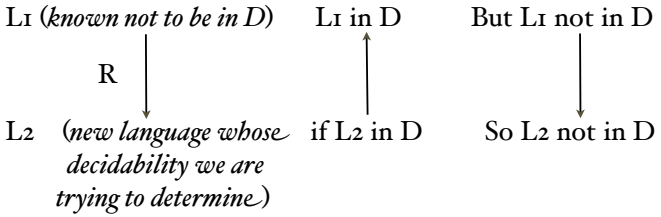
- A reduction R from L_1 to L_2 is one or more Turing machines such that:
  - If there exists a Turing machine Oracle that decides (or semidecides) L_2, then the Turing machines in R can be composed with Oracle to build a deciding (or a semideciding) Turing machine for L_1.
  - I.e., a solution to L_2 gives a solution to L_1
  - Write $L_1 \leq L_2$. (L_1 is easier than L_2)
- Last time showed reduction from H_{TM} to $L_{\varepsilon o}$

Using Reductions

- If R is a reduction from L_1 to L_2 and L_2 is in D, then L_1 is in D
- If R is a reduction from L_1 to L_2 and L_1 is not in D, then L_2 is not in D
- Because H_{TM} not decidable, neither is $L_{\varepsilon o}$

How to use reduction

- Showing that L2 is not in D:



To show L_2 undecidable

- Choose a language L_1:
  - that is already known not to be in D, and
  - that can be reduced to L_2.
- Suppose some *Oracle* solves L_2
- Use *Oracle* as subroutine in TM M to solve L_1.
  - Ensure that if $x \in L_1$, then M(x) accepts, and
  - If $x \notin L_1$, then M(x) rejects.

Mapping Reductions

- L_1 is *mapping reducible* to L_2 ($L_1 \leq_M L_2$) iff there exists some computable function f such that:
  - $\forall x \in \Sigma^* (x \in L_1 \leftrightarrow f(x) \in L_2)$.
- To decide whether x is in L_1, we transform it, using f, into a new object and ask whether that object is in L_2.

Mapping Reductions

- L_1 is *mapping reducible* to L_2 ($L_1 \leq_M L_2$) iff there exists some computable function f such that:
  - $\forall x \in \Sigma^* (x \in L_1 \leftrightarrow f(x) \in L_2)$.
- Example:
  - $H_{TM} \leq_M H_\varepsilon$ where $H_\varepsilon = \{<M> | M \text{ halts on empty tape}\}$
    - Take <M,w>, build new machine M'_w that writes w on tape and then simulates M. Then
    - $<M,w> \in H_{TM}$ iff $<M'_w> \in H_\varepsilon$

Equality of TM's

- Showed a few lectures back that $E_{TM} = \{M \mid M \text{ is a TM and } L(M)=\emptyset\}$ is undecidable.
- Use this to show
  $Eq_{TM} = \{<M_1,M_2> \mid L(M_1) = L(M_2)\}$
  is undecidable.
- Show $E_{TM} \leq Eq_{TM}$

Reduction

- To find if $<M> \in E_{TM}$:
  - Let $M_{\emptyset}$ be a TM that never accepts anything: $L(M_{\emptyset}) = \emptyset$.
  - Suppose Oracle solves EQ_{TM}. I.e., Oracle(<M,M'>) always halts and accepts iff L(M) = L(M').
  - Then $<M> \in E_{TM}$ iff $<M, M_{\emptyset}>$ accepted by Oracle.
  - Thus EQ_{TM} is undecidable.
- Showed $E_{TM} \leq_M Eq_{TM}$ via $f(M) = <M,M_{\emptyset}>$

Consequences

- If f and g are computable functions, then cannot decide if f = g.
- If V is program for virus, then cannot tell if suspicious program V' is equivalent.

Is everything undecidable?

- L = {<M> | M's program has less than 20 transitions }
- L = {<M> | M halts in exactly 47 steps }