MA/CSSE 474
Theory of Computation

More Reduction Proofs

Recap: Steps in a Reduction Proof

1. ✪ Choose an undecidable language to reduce from.
2. ✪ Define the reduction *R*.
3. Show that *C* (the composition of *R* with *Oracle*) is correct.

✪ indicates the choices that we make.

Undecidable Problems (Languages That Aren't In D)

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?	TMreg = {<M>:$L(M)$ is regular}

Tomorrow: We will prove some of these (most are also done in the book)

H_{ALL} = {<M> : TM M halts on all inputs}

We show that H_{ALL} is not in D by reduction from H_ε.

H_ε = {<M> : TM M halts on ε}

$\downarrow R$

(?*Oracle*) H_{ALL} = {<M> : TM M halts on all inputs }

R(<*M*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
   1.1. Erase the tape.
   1.2. Run *M*.
2. Return <*M#*>.

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*>)) decides H_ε:

- *R* can be implemented as a Turing machine.
- *C* is correct: *M#* halts on everything or nothing, depending on whether *M* halts on ε. So:
  - <*M*> ∈ H_ε: *M* halts on ε, so *M#* halts on all inputs. *Oracle* accepts.
  - <*M*> ∉ H_ε: *M* does not halt on ε, so *M#* halts on nothing. *Oracle* rejects.

But no machine to decide H_ε can exist, so neither does *Oracle*.

The Membership Question for TMs

We next define a new language:

A = {<*M*, *w*> : *M* accepts *w*}.

Note that A is different from H since it is possible that *M* halts but does not accept. An alternative definition of A is:

A = {<*M*, *w*> : *w* ∈ *L*(*M*)}.

A = {<*M*, *w*> : *w* ∈ *L*(*M*)}

We show that A is not in D by reduction from H.

H = {<*M, w*> : TM *M* halts on input string *w*}

$\downarrow R$

(?*Oracle*) A = {<*M, w* > : *w* ∈ *L*(*M*) }

R(<*M, w*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
    1.1. Erase the tape.
    1.2. Write *w* on the tape.
    1.3. Run *M* on *w*.
    1.4. **Accept**
2. Return <*M#, w*>.

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*, *w*>)) decides H:

- *R* can be implemented as a Turing machine.
- *C* is correct: *M#* accepts everything or nothing. So:
    - <*M, w*> ∈ H: *M* halts on *w*, so *M#* accepts everything. In particular, it accepts *w*. *Oracle* accepts.
    - <*M, w* > ∉ H: *M* does not halt on *w*. *M#* gets stuck in step 1.3 and so accepts nothing. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

A_ε, A_{ANY}, and A_{ALL}

Theorem: A_ε = {<*M*> : TM *M* accepts ε} is not in D.

Proof: Analogous to that for H_ε.

Theorem:
A_{ANY} = {<*M*> : TM *M* accepts at least one string} is not in D.

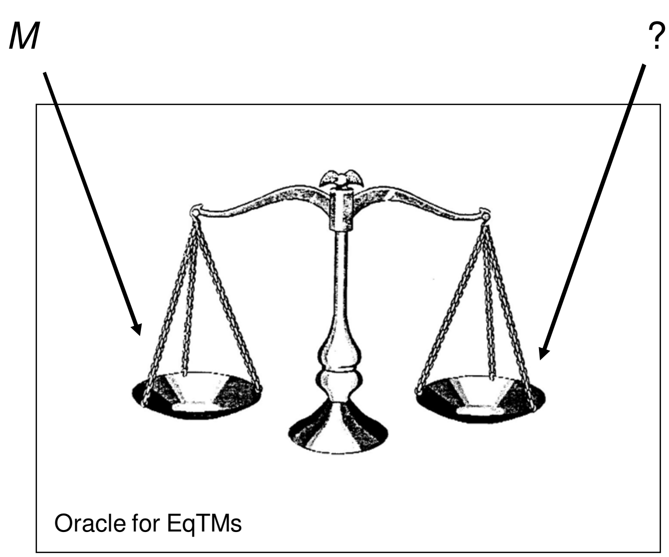
Proof: Analogous to that for H_{ANY}.

Theorem: A_{ALL} = {<*M*> : = *L*(*M*) = Σ*} is not in D.

Proof: Analogous to that for H_{ALL}.

Are safety and security properties of complex systems decidable? (J.2)

EqTMs={<M_a, M_b>: $L(M_a)=L(M_b)$}



EqTMs={$<M_a, M_b>$: $L(M_a)=L(M_b)$}

A_{ANY} = {$<M>$: there exists at least one string on which TM M halts}

$\Big\downarrow R$

(*Oracle*) EqTMs = {$<M_a, M_b>$: $L(M_a)=L(M_b)$}

$R(<M>)$ =

1. Construct the description of $M\#(x)$:
   1.1. Accept.
2. Return $<M, M\#>$.

If *Oracle* exists, then C = *Oracle*($R(<M>)$) decides A_{ANY}:

- C is correct: $M\#$ accepts everything. So:
  - $<M> \in A_{ANY}$: $L(M) =? L(M\#)$. *Oracle* ? Oops.
  - $<M> \notin A_{ANY}$: $L(M) \neq L(M\#)$. *Oracle* rejects.

EqTMs={$<M_a, M_b>$: $L(M_a)=L(M_b)$}

A_{ALL} = {$<M>$: $L(M) = \Sigma^*$}

$\Big\downarrow R$

(*Oracle*) EqTMs = {$<M_a, M_b>$: $L(M_a)=L(M_b)$}

$R(<M>)$ =

1. Construct the description of $M\#(x)$:
   1.1. Accept.
2. Return $<M, M\#>$.

If *Oracle* exists, then C = *Oracle*($R(<M>)$) decides A_{ALL}:

- C is correct: $M\#$ accepts everything. So if $L(M) = L(M\#)$, M must also accept everything. So:
  - $<M> \in A_{ALL}$: $L(M) = L(M\#)$. *Oracle* accepts.
  - $<M> \notin A_{ALL}$: $L(M) \neq L(M\#)$. *Oracle* rejects.

But no machine to decide A_{ALL} can exist, so neither does *Oracle*.

A Practical Consequence

Consider the problem of virus detection. Suppose that a new virus V is discovered and its code is $<V>$.

- Is it sufficient for antivirus software to check solely for occurrences of $<V>$?
- Is it possible for it to check for equivalence to V?

Sometimes Mapping Reducibility Isn't Right

Recall that a mapping reduction from L_1 to L_2 is a computable function f where:

$$\forall x \in \Sigma^* \ (x \in L_1 \leftrightarrow f(x) \in L_2).$$

When we use a mapping reduction, we return:

$$Oracle(f(x))$$

Sometimes we need a more general ability to use *Oracle* as a subroutine and then to do other computations after it returns.

{<M> : M accepts no even length strings}

H = {< M, w> : TM M halts on input string w}

$\downarrow R$

(?*Oracle*) L_2 = {<M> : M accepts no even length strings}

R(<M, w>) =
1. Construct the description <$M\#$>, where $M\#(x)$ operates as follows:
   1.1. Erase the tape.
   1.2. Write w on the tape.
   1.3. Run M on w.
   1.4. Accept.
2. Return <$M\#$>.

If *Oracle* exists, then C = *Oracle*(R(<M, w>)) decides H:
- C is correct: $M\#$ ignores its own input. It accepts everything or nothing, depending on whether it makes it to step 1.4. So:
  - <M, w> $\in$ H: M halts on w. *Oracle*:
  - <M, w> $\notin$ H: M does not halt on w. *Oracle*:

Problem:

{<M> : M accepts no even length strings}

H = {< M, w> : TM M halts on input string w}

$\downarrow R$

(?*Oracle*) L_2 = {<M> : M accepts no even length strings}

R(<M, w>) =
1. Construct the description <$M\#$>, where $M\#(x)$ operates as follows:
   1.1. Erase the tape.
   1.2. Write w on the tape.
   1.3. Run M on w.
   1.4. Accept.
2. Return <$M\#$>.

If *Oracle* exists, then $C = \neg$*Oracle*(R(<M, w>)) decides H:
- R and $\neg$ can be implemented as Turing machines.
- C is correct:
  - <M, w> $\in$ H: M halts on w. $M\#$ accepts everything, including some even length strings. *Oracle* rejects so C accepts.
  - <M, w> $\notin$ H: M does not halt on w. $M\#$ gets stuck. So it accepts nothing, so no even length strings. *Oracle* accepts. So C rejects.

But no machine to decide H can exist, so neither does *Oracle*.

Are All Questions about TMs Undecidable?

Let L = {<M> : TM M contains an even number of states}

Let L = {<M, w> : M halts on w within 3 steps}.

Let L_q = {<M, q> : there is some configuration

(p, $u\underline{a}v$) of M, with $p \neq q$,

that yields a configuration whose state is q }.

Is L_q decidable?

Is There a Pattern?

- Does L contain some particular string w?
- Does L contain ε?
- Does L contain any strings at all?
- Does L contain all strings over some alphabet Σ?

- A = {<M, w> : TM M accepts w}.
- A_ε = {<M> : TM M accepts ε}.
- A_{ANY} = {<M> : there exists at least one string that TM M accepts}.
- A_{ALL} = {<M> : TM M accepts all inputs}.

Rice's Theorem

No nontrivial property of the SD languages is decidable.

or

Any language that can be described as:

{<M>: $P(L(M))$ = *True*}

for any nontrivial property P, is not in D.

A ***nontrivial property*** is one that is not simply:

- *True* for all languages, or
- *False* for all languages.

Because of time constraints, we will skip the proof of this theorem.

Applying Rice's Theorem

To use Rice's Theorem to show that a language L is not in D we must:

- Specify property P.
- Show that the domain of P is the SD languages.
- Show that P is nontrivial:
  - P is true of at least one language
  - P is false of at least one language

Applying Rice's Theorem

1. $\{<M> : L(M)$ contains only even length strings$\}$.
2. $\{<M> : L(M)$ contains an odd number of strings$\}$.
3. $\{<M> : L(M)$ contains all strings that start with `a`$\}$.
4. $\{<M> : L(M)$ is infinite$\}$.
5. $\{<M> : L(M)$ is regular$\}$.
6. $\{<M> : M$ contains an even number of states$\}$.
7. $\{<M> : M$ has an odd number of symbols in its tape alphabet$\}$.
8. $\{<M> : M$ accepts ε within 100 steps$\}$.
9. $\{<M>: M$ accepts $\varepsilon\}$.
10. $\{<M_a, M_b> : L(M_a) = L(M_b)\}$.

Given a TM *M*, is *L*(*M*) Regular?

The problem: Is $L(M)$ regular?

As a language: Is $\{<M> : L(M)$ is regular$\}$ in D?

No, by Rice's Theorem:

- P = *True* if L is regular and *False* otherwise.
- The domain of P is the set of SD languages since it is the set of languages accepted by some TM.
- P is nontrivial:
  - $P(\mathbf{a}^*)$ = *True*.
  - $P(A^nB^n)$ = *False*.

We can also show it directly, using reduction. (Next slide)

Given a Turing Machine *M*, is *L*(*M*) Regular?

H = {<*M*, *w*> : TM *M* halts on input string *w*}

$\downarrow R$

(*Oracle*) L_2 = {<*M*> : *L*(*M*) is regular}

R(<*M*, *w*>) =

1. Construct *M#*(*x*):
   - 1.1. Copy its input *x* to another track for later.
   - 1.2. Erase the tape.
   - 1.3. Write *w* on the tape.
   - 1.4. Run *M* on *w.*
   - 1.5. Put *x* back on the tape.
   - 1.6. If $x \in A^nB^n$ then accept, else reject.
2. Return <*M#*>.

Problem:

But We Can Flip

R(<*M*, *w*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
   - 1.1. Save *x* for later.
   - 1.2. Erase the tape.
   - 1.3. Write *w* on the tape.
   - 1.4. Run *M* on *w.*
   - 1.5. Put *x* back on the tape.
   - 1.6. If $x \in A^nB^n$ then accept, else reject.
2. Return <*M#*>.

If Oracle decides L_2, then $C = \neg$*Oracle*(*R*(<*M*, *w*>)) decides H:

- <*M*, *w*> $\in$ H: *M#* makes it to step 1.5. Then it accepts *x* iff $x \in A^nB^n$. So *M#* accepts A^nB^n, which is not regular. *Oracle* rejects. *C* accepts.
- <*M*, *w*> $\notin$ H: *M* does not halt on *w*. *M#* gets stuck in step 1.4. It accepts nothing. *L*(*M#*) = $\varnothing$, which is regular. *Oracle* accepts. *C* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

Or, Doing it Without Flipping

R(<M, w>) =

1. Construct the description <M#>, where M#(x) operates as follows:
   1.1. If $x \in A^nB^n$ then accept, else:
   1.2. Erase the tape.
   1.3. Write w on the tape.
   1.4. Run M on w.
   1.5. Accept
2. Return <M#>.

If *Oracle* exists, C = *Oracle*(R(<M, w>)) decides H:

- C is correct: M# immediately accepts all strings in A^nB^n:
  - <M, w> $\in$ H: M# accepts everything else in step 1.5. So $L(M\#) = \Sigma^*$, which is regular. *Oracle* accepts.
  - <M, w> $\notin$ H: M# gets stuck in step 1.4, so it accepts nothing else. $L(M\#) = A^nB^n$, which is not regular. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

Any Nonregular Language Will Work

R(<M, w>) =

1. Construct the description <M#>, where M#(x) operates as follows:
   1.1. If $x \in$ **WW** then accept, else:
   1.2. Erase the tape.
   1.3. Write w on the tape.
   1.4. Run M on w.
   1.5. Accept
2. Return <M#>.

If *Oracle* exists, C = *Oracle*(R(<M, w>)) decides H:

- C is correct: M# immediately accepts all strings WW:
  - <M, w> $\in$ H: M# accepts everything else in step 1.5. So $L(M\#) = \Sigma^*$, which is regular. *Oracle* accepts.
  - <M, w> $\notin$ H: M# gets stuck in step 1.4, so it accepts nothing else. $L(M\#)$ = WW, which is not regular. *Oracle* rejects.

But no machine to decide H can exist, so neither does *Oracle*.

Is *L*(*M*) Context-free?

How about: L_3 = {<*M*> : *L*(*M*) is context-free}?

R(<*M*, *w*>) =
1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
   1.1. If $x \in A^nB^nC^n$ then accept, else:
   1.2. Erase the tape.
   1.3. Write *w* on the tape.
   1.4. Run *M* on *w*.
   1.5. Accept
2. Return <*M#*>.

Practical Impact of These Results

1. Does *P*, when running on *x*, halt?
2. Might *P* get into an infinite loop on some input?
3. Does *P*, when running on *x*, ever output a 0? Or anything at all?
4. Are P_1 and P_2 equivalent?
5. Does *P*, when running on *x*, ever assign a value to *n*?
6. Does *P* ever reach *S* on any input (in other words, can we chop it out?
7. Does *P* reach *S* on every input (in other words, can we guarantee that *S* happens)?

- Can the Patent Office check prior art?
- Can the CS department buy the definitive grading program?

Turing Machine Questions Can be Reduced to Program Questions

EqPrograms =

{<P_a, P_b> : P_a and P_b are *PL* programs and $L(P_a) = L(P_b)$}.

We can build, in any programming language *PL*, *SimUM*:

- that is a *PL* program
- that implements the Universal TM U and so can simulate an arbitrary TM.

{<*M*, *q*> : *M* reaches *q* on some input}

H_{ANY} = {<*M*> : there exists some string on which TM *M* halts}

↓ *R*

(?*Oracle*) L_2 = {<*M*, *q*> : *M* reaches *q* on some input}

R(<*M*>) =

1. Build <*M#*> so that *M#* is identical to *M* except that, if *M* has a transition $((q_1, c_1), (q_2, c_2, d))$ and q_2 is a halting state other than *h*, replace that transition with:
   $((q_1, c_1), (h, c_2, d))$.
2. Return <*M#*, *h*>.

A good example, but the term is flying by, so we will skip it for now.

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*>)) decides H_{ANY}:

- *R* can be implemented as a Turing machine.
- *C* is correct: *M#* will reach the halting state *h* iff *M* would reach some halting state. So:
  - <*M*> $\in H_{ANY}$: There is some string on which *M* halts. So there is some string on which *M#* reaches state *h*. *Oracle* accepts.
  - <*M*> $\notin H_{ANY}$: There is no string on which *M* halts. So there is no string on which *M#* reaches state *h*. *Oracle* rejects.

But no machine to decide H_{ANY} can exist, so neither does *Oracle*.

Side Road with a purpose: obtainSelf

From Section 25.3:

In section 25.3, the author proves the existence of a very useful computable function: *obtainSelf*. When called as a subroutine by any Turing machine M, *obtainSelf* writes <M> onto M's tape.

Related to quines

Some quines

- ```
  main(){char q=34, n=10,*a="main() {char
  q=34,n=10,*a=%c%s%c;
  printf(a,q,a,q,n);}%c";printf(a,q,a,q,n);}
  ```

- ```
  ((lambda (x) (list x (list 'quote x)))
   (quote (lambda (x) (list x (list 'quote x)))))
  ```

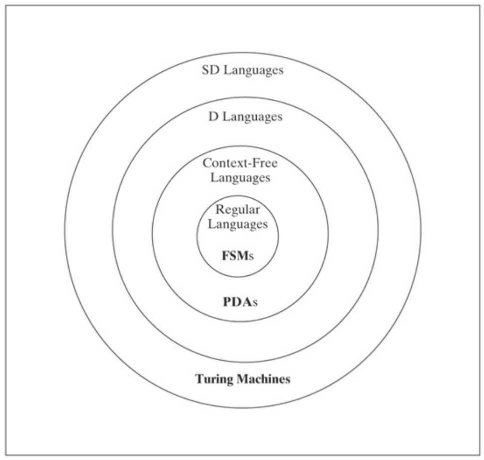
- Quine's paradox and a related sentence:

  "Yields falsehood when preceded by its quotation" yields falsehood when preceded by its quotation.

  "quoted and followed by itself is a quine." quoted and followed by itself is a quine.

Non-SD Languages

There is an uncountable number of non-SD languages, but only a countably infinite number of TM's (hence SD languages). ∴The class of non-SD languages is <u>much</u> bigger than that of SD languages!



Non-SD Languages

Intuition: Non-SD languages usually involve either infinite search (where testing each potential member could loop forever) or determining whether the a TM will infinite loop.

Examples:

- ¬H = {<*M*, *w*> : TM *M* does *not* halt on *w*}.
- {<*M*> : *L*(*M*) = Σ*}.
- {<*M*> : TM *M* halts on nothing}.

Proving Languages are not SD

- Contradiction
- *L* is the complement of an SD/D Language.
- Reduction from a known non-SD language

Contradiction

Theorem: TM_{MIN} =
{<*M*>: Turing machine *M* is minimal} is not in SD.

Proof: If TM_{MIN} were in SD, then there would exist some Turing machine *ENUM* that enumerates its elements. Define the following Turing machine:

M#(*x*) =
1. Invoke *obtainSelf* to produce <*M#*>.
2. Run *ENUM* until it generates the description of some Turing machine *M´* whose description is longer than |<*M#*>|.
3. Invoke *U* on the string <*M´*, *x*>.

Since TM_{MIN} is infinite, *ENUM* must eventually generate a string that is longer than |<*M#*>|. So *M#* makes it to step 3 and thus M# is Equivalent to *M´* since it simulates *M´*. But, since |<*M#*>| < |<*M´*>|, *M´* cannot be minimal.
But M#'s description was generated by *ENUM*. Contradiction.

The Complement of *L* is in SD/D

Suppose we want to know whether *L* is in SD and we know:

- ¬*L* is in SD, and
- At least one of *L* or ¬*L* is not in D.

Then we can conclude that *L* is not in SD, because, if it were, it would force both itself and its complement into D, which we know cannot be true.

Example:
- ¬H (since ¬(¬H) = H is in SD and not in D)

$\mathbf{A_{anbn} = \{<\textit{M}> : \textit{L}(\textit{M}) = A^nB^n\}}$

A_{anbn} contains strings that look like:

(`q00,a00,q01,a00,`→),
(`q00,a01,q00,a10,`→),
(`q00,a10,q01,a01,`←),
(`q00,a11,q01,a10,`←),
(`q01,a00,q00,a01,`→),
(`q01,a01,q01,a10,`→),
(`q01,a10,q01,a11,`←),
(`q01,a11,q11,a01,`←)

It does not contain strings like `aaabbb`.

But A^nB^n does.

A_{anbn} = {<*M*> : *L*(*M*) = A^nB^n}

What's wrong with this proof that A_{anbn} is not in SD?

$$\begin{array}{lcl} & \neg\mathrm{H} = \{<M, w> : \text{TM } M \text{ does not halt on } w\} \\ & \Big\downarrow R \\ (?Oracle) & A_{anbn} = \{<M> : L(M) = A^nB^n\} \end{array}$$

R(<*M*, *w*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
   1.1. Erase the tape.
   1.2. Write *w* on the tape.
   1.3. Run *M* on *w*.
   1.4. Accept.
2. Return <*M#*>.

If *Oracle* exists, *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides ¬H:

A_{anbn} = {<*M*> : *L*(*M*) = A^nB^n} is not SD

What about: ¬H = {<*M*, *w*> : TM *M* does not halt on *w*}

$$\begin{array}{lcl} & \Big\downarrow R \\ (?Oracle) & A_{anbn} = \{<M> : L(M) = A^nB^n\} \end{array}$$

R(<*M*, *w*>) =

1. Construct the description <*M#*>, where *M#*(*x*) operates as follows:
   1.1 Copy the input *x* to another track for later.
   1.2. Erase the tape.
   1.3. Write *w* on the tape.
   1.4. Run *M* on *w*.
   1.5. Put *x* back on the tape.
   1.6. If $x \in A^nB^n$ then accept, else loop.
2. Return <*M#*>.

If *Oracle* exists, *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides ¬H:

A_{anbn} = {<*M*> : *L*(*M*) = A^nB^n} is not SD

R(<*M*, *w*>) reduces ¬H to A_{anbn}:

1. Construct the description <*M#*>:
   1.1. If $x \in A^nB^n$ then accept. Else:
   1.2. Erase the tape.
   1.3. Write *w* on the tape.
   1.4. Run *M* on *w*.
   1.5. Accept.
2. Return <*M#*>.

If *Oracle* exists, then *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides ¬H:

M# immediately accepts all strings in A^nB^n. If *M* does not halt on *w*, those are the only strings *M#* accepts. If *M* halts on *w*, *M#* accepts everything:

- <*M*, *w*> ∈ ¬H: *M* does not halt on *w*, so *M#* accepts strings in A^nB^n in step 1.1. Then it gets stuck in step 1.4, so it accepts nothing else. It is an A^nB^n acceptor**.** ***Oracle* accepts**.
- <*M*, *w*> ∉ ¬H: *M* halts on *w*, so *M#* accepts everything**.** ***Oracle* does not accept.**

But no machine to semidecide ¬H can exist, so neither does *Oracle*.