MA/CSSE 474
Theory of Computation

More and More Reduction Proofs
Proof that a language is not in SD

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

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

↓ *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

$R(\langle M, w\rangle)$ =

1. Construct the description $\langle M\#\rangle$, 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 $\langle M\#\rangle$.

If Oracle decides L_2, then $C = \neg Oracle(R(\langle M, w\rangle))$ decides H:

- $\langle M, w\rangle \in$ 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.
- $\langle M, w\rangle \notin$ H: M does not halt on w. $M\#$ gets stuck in step 1.4. It accepts nothing. $L(M\#) = \emptyset$, which is regular. *Oracle* accepts. C rejects.

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

Or, Doing it Without Flipping

$R(\langle M, w\rangle)$ =

1. Construct the description $\langle M\#\rangle$, 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 $\langle M\#\rangle$.

If *Oracle* exists, $C = Oracle(R(\langle M, w\rangle))$ decides H:

- C is correct: $M\#$ immediately accepts all strings in A^nB^n:
  - $\langle M, w\rangle \in$ H: $M\#$ accepts everything else in step 1.5. So $L(M\#) = \Sigma^*$, which is regular. *Oracle* accepts.
  - $\langle M, w\rangle \notin$ 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*#) = Σ^*, 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*#>.

Essentially the same proof as for L(M) regular

Practical Impact of These Results

1. Does program *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 output 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 section S *S* on any input (in other words, can we chop section S 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?

{<*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*> ∈ H_{ANY}: There is some string on which *M* halts. So there is some string on which *M*# reaches state *h*. *Oracle* accepts.
  - <*M*> ∉ 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

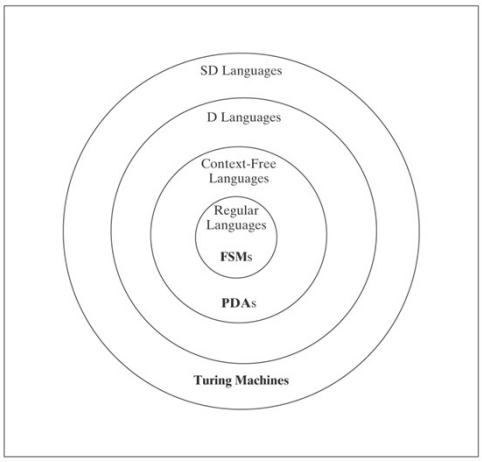
- `main(){char q=34, n=10,*a="main() {char q=34,n=10,*a=%c%s%c; printf(a,q,a,q,n);}%c";printf(a,q,a,q,n);}`
- `((lambda (x) (list x (list 'quote x))) (quote (lambda (x) (list x (list 'quote x)))))`
- Quine's paradox and a related sentence:

  "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:

- $\neg H = \{\langle M, w\rangle : \text{TM } M \text{ does } \textit{not} \text{ halt on } w\}$.
- $\{\langle M\rangle : L(M) = \Sigma^*\}$.
- $\{\langle M\rangle : \text{TM } M \text{ halts on nothing}\}$.

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 minimal 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)

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

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?

$\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}$

$\downarrow R$

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

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: $\neg H = \{<M, w> : \text{TM } M \text{ does not halt on } w\}$

$\downarrow R$

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

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 $\neg H$ 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 $\neg H$:
$M\#$ 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:

- $<M, w> \in \neg H$: 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**.
- $<M, w> \notin \neg H$: M halts on w, so $M\#$ accepts everything. ***Oracle* does not accept.**

But no machine to semidecide $\neg H$ can exist, so neither does *Oracle*.

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

$R(<M, w>)$ reduces $\neg H$ 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 $\neg H$:
$M\#$ 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:

- $<M, w> \in \neg H$: 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**.
- $<M, w> \notin \neg H$: M halts on w, so $M\#$ accepts everything. ***Oracle* does not accept.**

But no machine to semidecide $\neg H$ can exist, so neither does *Oracle*.

H_{ALL} = {<*M*> : TM halts on Σ^*}

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

R ↓

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

Reduction Attempt 1: *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*.
2. Return <*M*#>.

- If <M, w> $\in \neg H$:
- If <M, w> $\notin \neg H$:

There May Be No Easy Way to Flip

$\neg H$ = {<*M*, *w*> : TM *M* does not halt on *w*}

R ↓

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

Reduction Attempt 1: *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*.
2. Return <*M*#>.

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

- <*M*, *w*> $\in \neg H$: *M* does not halt on *w*, so *M*# gets stuck in step 1.3 and halts on nothing. ***Oracle* does not accept.**
- <*M*, *w*> $\notin \neg H$: *M* halts on *w*, so *M*# halts on everything. ***Oracle* accepts**.

$H_{ALL} = \{\langle M \rangle : \text{TM halts on } \Sigma^*\}$

$R(\langle M, w \rangle)$ reduces $\neg H$ to H_{ALL}:

1. Construct the description $\langle M\# \rangle$, 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 for $|x|$ steps or until M naturally halts.
   - 1.5. If M naturally halted, then loop.
   - 1.6. Else halt.
2. Return $\langle M\# \rangle$.

If *Oracle* exists, $C = Oracle(R(\langle M, w \rangle))$ semidecides $\neg H$:

- $\langle M, w \rangle \in \neg H$: No matter how long x is, M will not halt in $|x|$ steps. So, for all inputs x, $M\#$ makes it to step 1.6. So it halts on everything. ***Oracle* accepts.**
- $\langle M, w \rangle \notin \neg H$: M halts on w in n steps. On inputs of length less than n, $M\#$ makes it to step 1.6 and halts. But on all inputs of length n or greater, $M\#$ will loop in step 1.5. ***Oracle* does not accept.**

$EqTMs = \{\langle M_a, M_b \rangle : L(M_a) = L(M_b)\}$

We've already shown it's not in D.

Now we show it's also not in SD.

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

$\neg$H = {<M, w> : TM M does not halt on w}

R ↓

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

R(<M, w>) =

1. Construct the description <M#>:

2. Construct the description <M?>:

3. Return <M#, M?>.

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

- <M, w> $\in$ $\neg$H:
- <M, w> $\notin$ $\neg$H:

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

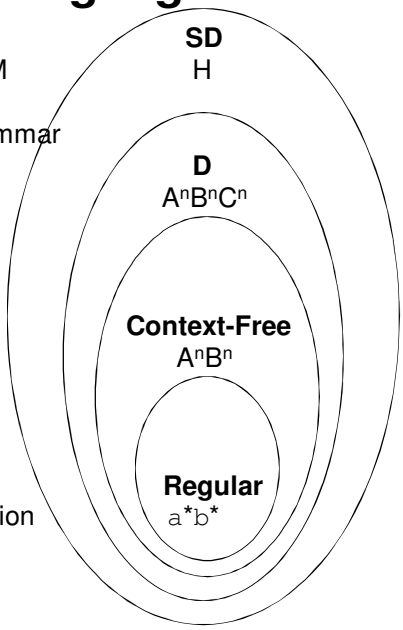
R(<M, w>) =

1. Construct the description <M#>:
   - 1.1 Erase the tape.
   - 1.2 Write w on the tape.
   - 1.3 Run M on w.
   - 1.4 Accept.
2. Construct the description <M?>:
   - 1.1 Loop.
3. Return <M#, M?>.

If *Oracle* exists, C = *Oracle*(R(<M, w>)) semidecides $\neg$H: M? halts on nothing.

- <M, w> $\in$ $\neg$H: M does not halt on w, so M# gets stuck in step 1.3 and halts on nothing. ***Oracle* accepts.**
- <M, w> $\notin$ $\neg$H: M halts on w, so M# halts on everything. ***Oracle* does not accept.**

Language Summary



IN	OUT
Semideciding TM Enumerable Unrestricted grammar	Reduction
Deciding TM Lexico. enum *L* and ¬*L* in SD	Diagonalize Reduction
CF grammar PDA Closure	Pumping Closure
Regular Expression FSM	Pumping Closure