Undecidability II

Sections 21.4 - 21.7

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 $w\}$.
- $A_{\varepsilon} = \{<M> :$ TM M accepts $\varepsilon\}$.
- $A_{ANY} = \{<M> :$ there exists at least one string that TM M accepts$\}$.
- $A_{ALL} = \{<M> :$ TM M accepts all inputs$\}$.

Rice's Theorem

Any nontrivial property of the SD languages is undecidable.

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.

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

Examples

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 ε}.
10. {<M_a, M_b> : $L(M_a) = L(M_b)$}.

Proof of Rice's Theorem

Proof: Let P be any nontrivial property of the SD languages.

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

$$R \downarrow$$

(?*Oracle*) $L_2 = \{<M> : P(L(M)) = T\}$

Either $P(\varnothing) = T$ or $P(\varnothing) = F$.
Assume it is $P(\varnothing) = F$ (a matching proof exists if it is T).

Since P is nontrivial, there is some SD language L_T such that $P(L_T) = T$. Let M_T be some Turing machine that semidecides L_T.

Proof (cont'd)

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

1. Construct <*M*#>, so *M*#(*x*) operates as follows:
    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 and run M_T on *x*.
2. Return <*M*#>.

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

<*M*, *w*> ∈ H: *M* halts on *w*. *M*# makes it to 1.5.
So it is equivalent to M_T.
$L(M\#) = L(M_T)$ and so $P(L(M\#)) = P(L(M_T)) = T$.
Oracle decides *P*. *Oracle* accepts.

<*M*, *w*> ∉ H: *M* does not halt on *w*. *M*# gets stuck in 1.4.
So it accepts nothing.
$L(M\#) = \oslash$ and so $\mathrm{P}(\mathrm{L}(M\#)) = P(\oslash) = F$.
Oracle decides *P*. *Oracle* rejects.

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

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

The language: Is {<*M*> : *L*(*M*) is regular} in D?

Rice's Theorem says no:

- *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*(a*) = *True*.
  - ♦ *P*(A^nB^n) = *False*.

Reduction from *H*

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

- <*M*, *w*> ∈ 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*> ∉ 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*.

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 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 Works

$R(<M, w>) =$

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: $L_3 = \{<M> : L(M) \text{ is context-free}\}$?

$R(<M, w>) =$

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 Implications on Programs

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 a coding segment *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)?

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.

TM Questions and Program Questions

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

Theorem: EqPrograms is not in D.

Proof: Reduction from EqTMs = $\{<M_a, M_b> : L(M_a) = L(M_b)\}$.

$R(<M_a, M_b>) =$

1. Build P_1, a PL program that, on w, returns $SimUM(M_a, w)$.
2. Build P_2, a PL program that, on w, returns $SimUM(M_b, w)$.
3. Return $<P_1, P_2>$.

If *Oracle* exists and decides EqPrograms, then $C = Oracle(R(<M_a, M_b>))$ decides EqTMs. C is correct. $L(P_1) = L(M_a)$ and $L(P_2) = L(M_b)$. So:

- $<M_a, M_b> \in$ EqTMs: $L(M_a) = L(M_b)$. So $L(P_1) = L(P_2)$. $Oracle(<P_1, P_2>)$ accepts.
- $<M_a, M_b> \notin$ EqTMs: $L(M_a) \neq L(M_b)$. So $L(P_1) \neq L(P_2)$. $Oracle(<P_1, P_2>)$ rejects.

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

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

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

$\downarrow R$

(?*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*>.

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*.

How many Turing machines does it take to change a light bulb?

One.

How can you tell whether your Turing machine is the one?

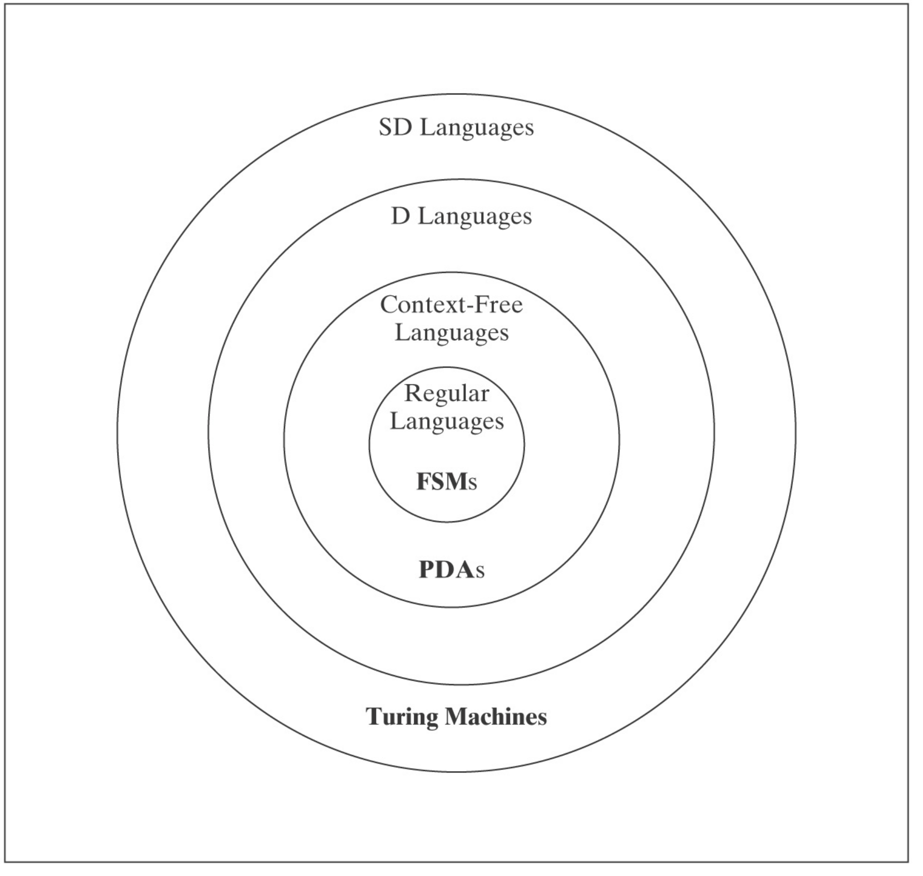
You can't.

- Tim Nodine

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 or knowing a TM will go to an infinite loop.

Examples:

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

Proving a language is not in SD:

- L is the complement of an SD\D Language.
    - Recall that $L, \neg L \in SD \Rightarrow L, \neg L \in D$
- Reduction from a known non-SD language

Complement is in SD/D

Theorem: **$H_{\neg ANY}$** = {<*M*> : there does ***not*** exist any string on which TM *M* halts} is not in SD.

Proof: $\neg H_{\neg ANY} = H_{ANY} =$
{<*M*> : there exists at least one string on which TM *M* halts}.

We already know:

- $\neg H_{\neg ANY}$ is in SD.
- $\neg H_{\neg ANY}$ is not in D.

So $H_{\neg ANY}$ is not in SD because, if it were, then H_{ANY} would be in D but it isn't.

Using Reduction

Theorem: If there is a reduction R from L_1 to L_2 and L_1 is not SD, then L_2 is not SD.

So, we must:

- Choose a language L_1 that is known not to be in SD.
- Hypothesize the existence of a *semideciding* TM *Oracle*.

Note: *R* may not swap accept for loop!

Using Reduction for $H_{\neg ANY}$

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

$$\Big\downarrow R$$

(?*Oracle*) $H_{\neg ANY} = \{<M> :$ there does not exist a string on which TM M halts$\}$

$R(<M, w>) =$

1. Construct the description $<M\#>$ of $M\#(x)$:
   1.1. Erase the tape.
   1.2. Write w on the tape.
   1.3. Run M on w.
2. Return $<M\#>$.

Using Reduction for $H_{\neg ANY}$

$R(<M, w>)$ =

1. Construct the description $<M\#>$ of $M\#(x)$:
    - 1.1. Erase the tape.
    - 1.2. Write w on the tape.
    - 1.3. Run M on w.
2. Return $<M\#>$.

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

- C is correct: $M\#$ ignores its input. It halts on everything or nothing, depending on whether M halts on w. So:
    - $<M, w> \in \neg$H: M does not halt on w, so $M\#$ halts on nothing. *Oracle* accepts.
    - $<M, w> \notin \neg$H: M halts on w, so $M\#$ halts on everything. *Oracle* does not accept.

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

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

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.

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

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

$$\big\downarrow R$$

(?*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 $\neg H$: !?!?

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*.

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

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

$$\downarrow R$$

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

Reduction Attempt 1: $R(\langle M, w \rangle) =$

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

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

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

Problem: cannot flip the answer.

H_{ALL} = {<*M*> : TM halts on Σ^*}

R(<*M*, *w*>) reduces $\neg$H to H_{ALL}:

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* for |*x*| steps or until *M* naturally halts.
   1.5. If *M* naturally halted, then loop.
   1.6. Else halt.
2. Return <*M#*>.

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

- <*M*, *w*> $\in \neg$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.
- <*M*, *w*> $\notin \neg$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 = {<M_a, M_b> : $L(M_a) = L(M_b)$}

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

$\downarrow R$

(?*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 ¬H: *M?* halts on nothing.

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

The Details Matter

L_1 = {<M>: M has an even number of states}.

L_2 = {<M>: |<M>| is even}.

L_3 = {<M>: $|L(M)|$ is even}.

L_4 = {<M>: M accepts all even length strings}.

The Details Matter

L_3 = {<*M*>: $|L(M)|$ is even}.

$\neg$H $\leq_M L_3$: 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 If x = ε then accept. Else loop.
2. Return <*M#*>.

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

The Details Matter

$L_4 = \{<M>: M$ accepts all even length strings$\}$

$\neg H \leq_M L_4$: $R(<M, w>) =$

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 for $|x|$ steps or until M naturally halts.
    1.5 If M halted naturally, then loop.
    1.6 Else accept.
2. Return $<M\#>$.

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

Accepting, Rejecting, Halting, and Looping

Consider :

L_1 = {<M, w>: M rejects w}.

L_2 = {<M, w>: M does not halt on w} (¬H)

L_3 = {<M, w>: M is a deciding TM and rejects w}.

{<*M*, *w*>: *M* is a Deciding TM and Rejects *w*}

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

$\Big\downarrow R$

(?*Oracle*) {<*M*, *w*>: *M* is a deciding TM and rejects *w*}

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 Reject.
2. Return <*M*#, ε>.

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

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

Problem:

{<*M*, *w*>: *M* is a Deciding TM and Rejects *w*}

H_{ALL} = {<*M*> : TM *M* halts on Σ^*}

$\big\downarrow R$

(?*Oracle*) {<*M*, *w*>: *M* is a deciding TM and rejects *w*}

R(<*M*>) =

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

If *Oracle* exists, *C* = *Oracle*(*R*(<*M*>)) semidecides H_{ALL}:

- <*M*> $\in H_{ALL}$: *M*# halts and rejects all inputs. *Oracle* accepts.
- <*M*> $\notin H_{ALL}$: There is at least one input on which *M* doesn't halt. So *M*# is not a deciding TM. *Oracle* does not accept.

No machine to semidecide H_{ALL} can exist, so neither does *Oracle*.

What About These?

$L_1 = \{a\}$.

$L_2 = \{<M> : M \text{ accepts a}\}$.

$L_3 = \{<M> : L(M) = \{a\}\}$.

$\neg H \leq_M L_3$: $R(<M, w>) =$

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

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

$\{<M_a, M_b> : \varepsilon \in L(M_a) - L(M_b)\}$

R is a reduction from ¬H. *R*(<*M*, *w*>) =
1. Construct the description of *M*#(*x*) that operates as follows:
    1.1. Erase the tape.
    1.2. Write *w*.
    1.3. Run *M* on *w*.
    1.4. Accept.
2. Construct the description of *M*?(*x*) that operates as follows:
    2.1. Accept.
3. Return <*M*?, *M*#>.

If *Oracle* exists and semidecides *L*, *C* = *Oracle*(*R*(<*M*, *w*>)) semidecides ¬H: *M*? accepts everything, including ε. So:

- <*M*, *w*> ∈ ¬H: *L*(*M*?) - *L*(*M*#) =

- <*M*, *w*> ∉ ¬H: *L*(*M*?) - *L*(*M*#) =

The Problem View	*The Language View*	Status
Does TM M have an even number of states?	$\{<M> : M$ has an even number of states$\}$	D
Does TM M halt on w?	$H = \{<M, w> : M$ halts on $w\}$	SD/D
Does TM M halt on the empty tape?	$H_\varepsilon = \{<M> : M$ halts on $\varepsilon\}$	SD/D
Is there any string on which TM M halts?	$H_{ANY} = \{<M> :$ there exists at least one string on which TM M halts $\}$	SD/D
Does TM M halt on all strings?	$H_{ALL} = \{<M> : M$ halts on $\Sigma^*\}$	¬SD
Does TM M accept w?	$A = \{<M, w> : M$ accepts $w\}$	SD/D
Does TM M accept ε?	$A_\varepsilon = \{<M> : M$ accepts $\varepsilon\}$	SD/D
Is there any string that TM M accepts?	A_{ANY} $\{<M> :$ there exists at least one string that TM M accepts $\}$	SD/D

Does TM *M* accept all strings?	$A_{ALL} = \{<M> : L(M) = \Sigma^*\}$	¬SD
Do TMs M_a and M_b accept the same languages?	$EqTMs = \{<M_a, M_b> : L(M_a) = L(M_b)\}$	¬SD

Does TM *M* not halt on any string?	$H_{\neg ANY} = \{<M> :$ there does not exist any string on which M halts$\}$	¬SD
Does TM *M* not halt on its own description?	$\{<M> :$ TM M does not halt on input $<M>\}$	¬SD
Is the language that TM *M* accepts regular?	$TMreg = \{<M> : L(M)$ is regular$\}$	¬SD
Does TM *M* accept the language A^nB^n?	$A_{anbn} = \{<M> : L(M) = A^nB^n\}$	¬SD

Language Summary

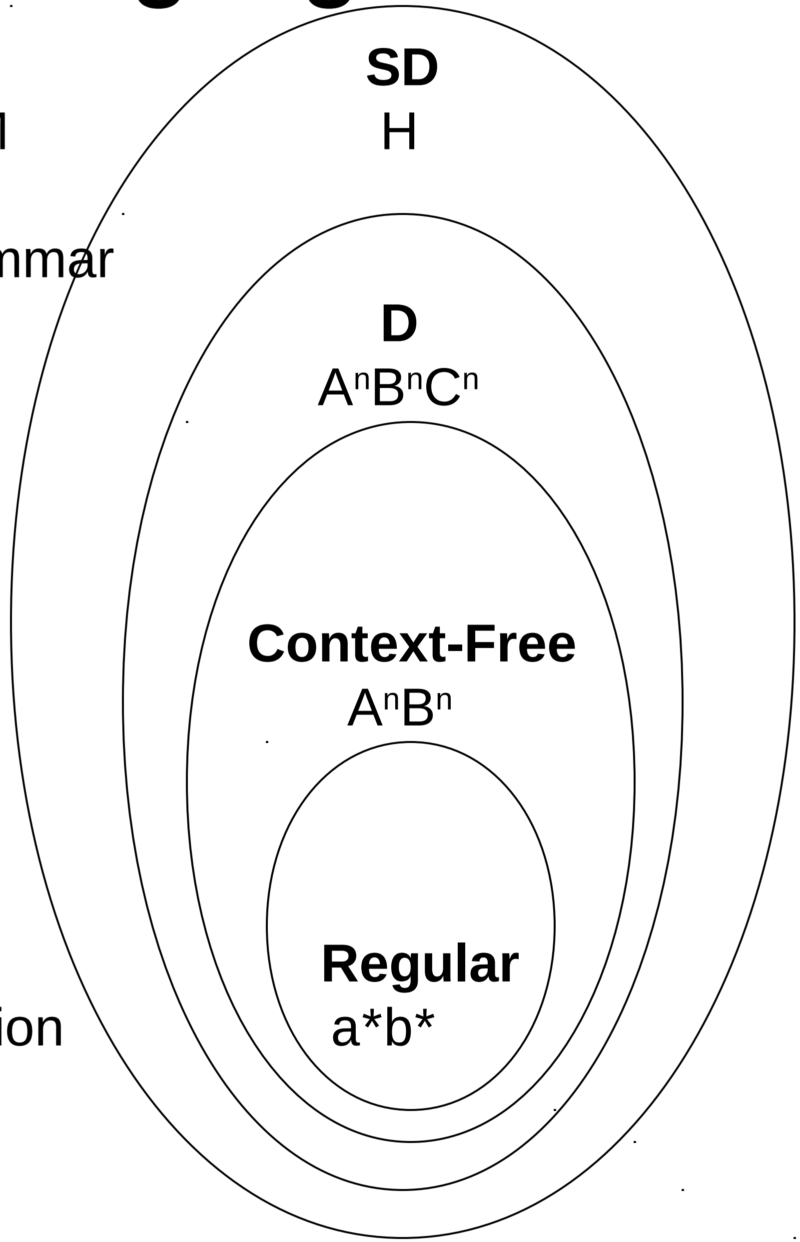
IN

Semideciding TM
Enumerable
Unrestricted grammar

Deciding TM
Lexico. enum
L and ¬*L* in SD

CF grammar
PDA
Closure

Regular Expression
FSM



OUT

Reduction

Diagonalize
Reduction

Pumping
Closure

Pumping
Closure