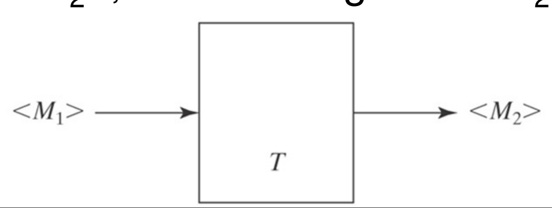
MA/CSSE 474
Theory of Computation

HW 12 due tomorrow,
13 Monday,
14 next Thursday
Claude out of town Wednesday
No class meeting on Thursday

The Halting Problem

Key ideas so far - 1

- Let Σ_U be
  {`(`, `)`, `a`, `q`, `y`, `n`, `0`, `1`, comma, →, ←},
  ordered as listed
- Any TM M may be encoded as a string <M> over alphabet Σ_U
- A TM T may take as input $\langle M_1 \rangle$, an encoding of one TM M_1, and produce as output $\langle M_2 \rangle$, an encoding of TM M_2



Key ideas so far - 2

- We can lexicographically enumerate:
  - All TM encodings
  - All TM encodings with a given input alphabet
  - All TM encodings with a given input alphabet and a given tape alphabet
- For any TM M and any string w over M's input alphabet, we can encode the pair M, w as a single string <M, w>
- There is a universal TM U with input alphabet $\Sigma_{U.}$

Key ideas so far 3

- There is a **universal TM** U whose input alphabet is $\Sigma_{U.}$
- If U is started with input <M,w>, it simulates the behavior of M, started with input w:
  - If M does not halt, U does not halt
  - If M halts and accepts, so does U
  - If M halts and rejects, so does U
  - If M is a "function computing" TM,
U leaves the same string on the tape as M,
so that U(<M, w>) = M(w)
- **Church-Turing Thesis:**
"Computable" is equivalent to "computable by a Turing machine"

Recap: D and SD

- A TM M with input alphabet Σ ***decides*** a language $L \subseteq \Sigma^*$ iff, for any string $w \in \Sigma^*$,
  - if $w \in L$ then M accepts w, and
  - if $w \notin L$ then M rejects w.

A language L is ***decidable*** (an element of D) iff there is a Turing machine M that decides it.

- A TM M with input alphabet Σ ***semidecides*** L iff for any string $w \in \Sigma^*$,
  - if $w \in L$ then M accepts w
  - if $w \notin L$ then M does not accept w. M may reject or loop.

A language L is ***semidecidable*** (an element of SD) iff there is a Turing
machine that semidecides it.

Defining the Universe

What is the complement of:

•$A^nB^n = \{a^n b^n : n \geq 0\}$

Depends on the universe:
That universe may be {a, b}*, or even {a, b, c, d}*, or could be $\{a^k b^m\}$

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

Universe may be Σ_U^*, or could be

{<*M*, *w*> : M is a TM and w is a string over M's input alphabet}

Defining the Universe

$L_1 = \{<M, w> : \text{TM } M \text{ halts on input string } w\}$.
$L_2 = \{<M> : M \text{ doesn't halt on any input string}\}$.
$L_3 = \{<M_a, M_b> : M_a \text{ and } M_b \text{ halt on the same strings}\}$.

For a string w to be in L_1, it must:

- be syntactically well-formed.
- encode a machine M and a string w such that M halts when started on w.

Define the universe from which we are drawing strings to contain only those strings that meet the syntactic requirements of the language definition.

This convention has no impact on the decidability of any of these languages since the set of syntactically valid strings is clearly in D.

A Different Definition of Complement

Our earlier definition:

$$\neg L_1 = \{x : x \text{ is not a syntactically well formed } <M, w> \text{ pair}\} \cup \{<M, w> : \text{TM } M \text{ does not halt on input string } w\}.$$

We will use a different definition:

Define the ***complement*** of any language L whose member strings include at least one Turing machine description to be with respect to a universe of strings that are of the same syntactic form as L.

Now we have:

$$\neg L_1 = \{<M, w> : \text{TM } M \text{ does not halt on input string } w\}.$$

Q1

The Language H

Theorem: The language:

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

- is semidecidable, but
- is not decidable.

Proof soon!

Q2

Does This Program Always Halt?

times3(x: positive integer) =
  while $x \neq 1$ do:
    if x is even then $x = x/2$.
    else $x = 3x + 1$

times3(25) …

Lothar Collatz, 1937, conjectured that times3 halts for all positive integers n. Still an open problem.

Paul Erdős: "Mathematics is not yet ready for such confusing, troubling, and hard problems."

http://mathworld.wolfram.com/CollatzProblem.html

```
max = 100000
maxCount = 0
for i in range(1, max+1):
    current = i
    count = 0

    while current != 1:
        count += 1
        if current % 2 == 0:
            current /= 2
        else:
            current = 3 * current + 1

    print "%7d %7d" % (i, count)
    if count > maxCount:
        maxCount = count

print "maxCount = ", maxCount
```

Q3

H is Semidecidable

Lemma: The language:

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

is semidecidable.

Proof: The TM M_H semidecides H:

M_H(<*M*, *w*>) =
1. Run *M* on *w*.

M_H halts iff *M* halts on *w*. Thus M_H semidecides H.

Q4

The Undecidability of the Halting Problem

Lemma: The language:

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

is not decidable.

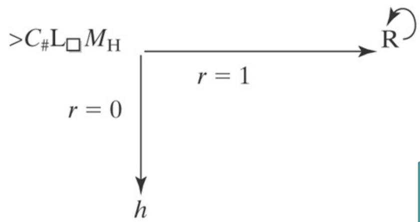
Proof: If H were decidable, then some TM M_H would decide it. M_H would implement the specification:

halts(<*M*, *w*>) =
  if <*M*> is a Turing machine description
    and *M* halts on input *w*
      then accept.
      else reject.

Trouble [in (Wabash) River City)]

Trouble(x: string) =
   if ***halts*(<*x*, *x*>)** then loop forever, else halt.

If there is an M_H that computes the function *halts*, *Trouble* exists:



Recall that $C_{\#}$ makes a copy of the non-blank characters on the tape.

Note that it is important to this proof that Trouble be constructible from M_H

Consider *halts*(<*Trouble*, *Trouble*>):
- If *halts* reports that *Trouble*(<*Trouble*>) halts, *Trouble* loops.
- But if *halts* reports that *Trouble*(<*Trouble*>) does not halt, then *Trouble* halts.

Q5-6

Viewing the Halting Problem as Diagonalization

- Lexicographically enumerate Turing machine encodings and input strings.
- Let 1 mean halting, blank mean non halting.

	i_1	i_2	i_3	…	<Trouble>	…
<machine$_1$>	1					
machine$_2$>		1				
machine$_3$>					1	
…				1		
<Trouble>			1			1
…	1	1	1			
…				1		

If M_H exists and decides membership in H, it must be able to correctly fill in any cell in this table.

What about the shaded square?

If H were in D, then SD would equal D

Recall: H = {<*M*, *w*> : TM *M* halts on input string *w*}
We know that H∈SD. If H were also in D, then there would exist a TM *O* that decides it.

Theorem: If H were in D then every SD language would be in D.

Proof: Let *L* be any SD language. There exists a TM M_L that semidecides it. The following machine M' decides whether *w* is in $L(M_L)$:

M'(*w*: string) =
1. Run *O* on $<M_L, w>$. (O will always halt)
2. If *O* accepts (i.e., M_L will halt on input w), then:
   2.1. Run M_L on *w*.
   2.2. If it accepts, accept. Else reject.
3. Else reject.

Q7

Recap: The Entscheidungsproblem

From Wikipedia: The Entscheidungsproblem ("decision problem", David Hilbert 1928) asks for an algorithm that will take as input a description of a formal language and a mathematical statement in the language, and produce as output either "True" or "False" according to whether the statement is true or false.

The algorithm need not justify its answer, nor provide a proof, so long as it is always correct.

Back to the Entscheidungsproblem

Theorem: The Entscheidungsproblem is unsolvable.

Proof: (Due to Turing)

1. If we could solve the problem of determining whether a given Turing machine ever prints the symbol 0, then we could solve the problem of determining whether a given Turing machine halts.
2. But we can't solve the problem of determining whether a given Turing machine halts, so neither can we solve the problem of determining whether it ever prints 0.
3. Given a Turing machine *M*, we can construct a logical formula *F* that is true iff *M* ever prints the symbol 0.
4. If there were a solution to the Entscheidungsproblem, then we would be able to determine the truth of any logical sentence, including *F,* and thus be able to decide whether *M* ever prints the symbol 0.
5. But we know that there is no procedure for determining whether *M* ever prints 0.
6. So there is no solution to the Entscheidungsproblem.

Decidable and Semidecidable Languages

Every CF Language is in D

Theorem: The set of context-free languages is a *proper* subset of D.

Proof:

- Every context-free language is decidable, so the context-free languages are a subset of D.
- There is at least one language, $A^nB^nC^n$, that is decidable but not context-free.

So the context-free languages are a *proper* subset of D.

Decidable and Semidecidable Languages

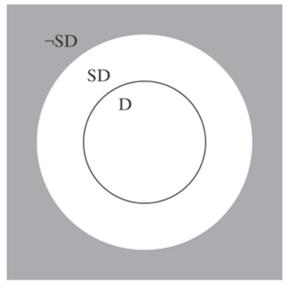
Almost every obvious language that is in SD is also in D:

- $A^nB^nC^n = \{\texttt{a}^n\texttt{b}^n\texttt{c}^n, n \geq 0\}$
- $\{w\texttt{c}w, w \in \{\texttt{a}, \texttt{b}\}^*\}$
- $\{ww, w \in \{\texttt{a}, \texttt{b}\}^*\}$
- $\{x*y=z$: $x,y,z \in \{\texttt{0}, \texttt{1}\}^*$ and, when x, y, and z are viewed as binary numbers, $xy = z\}$

But there are languages that are in SD but not in D:

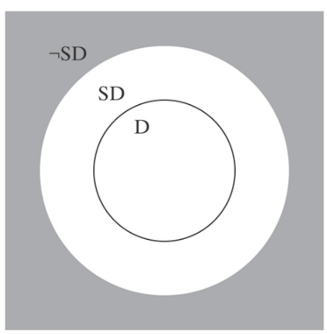
- H = {<*M*, *w*> : *M* halts on input *w*}
- {*w*: *w* is the email address of someone who will respond to a message you just posted to your newsgroup}

D and SD



1. D is a subset of SD. In other words, every decidable language is also semidecidable.
2. There exists at least one language that is in SD/D, the donut in the picture.
3. What about languages that are not in SD? Is the gray area of the figure empty?

Subset Relationships between D and SD



1. There exists at least one SD language that is not D.

2. Every language that is in D is also in SD: If *L* is in D, then there is a Turing machine *M* that decides it (by definition).

But *M* also semidecides it.

Languages That Are Not in SD

Theorem: 3. There are languages that are not in SD.

Proof: Assume any nonempty alphabet Σ.

Lemma: There is a countably infinite number of SD languages over Σ.

Proof:

Lemma: There is an uncountably infinite number of languages over Σ.

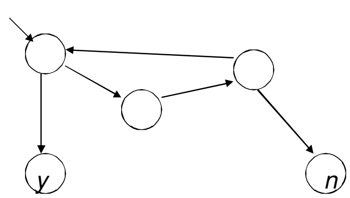
So there are more languages than there are languages in SD. Thus there must exist at least one language that is in ¬SD.

Closure of D Under Complement

Theorem: The set D is closed under complement.

Proof: (by construction) If *L* is in D, then there is a deterministic Turing machine *M* that decides it.

M:



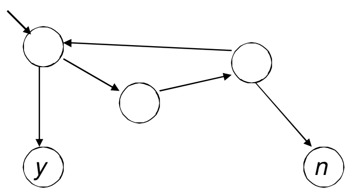
From *M*, we construct *M´* to decide ¬*L*:

Closure of D Under Complement

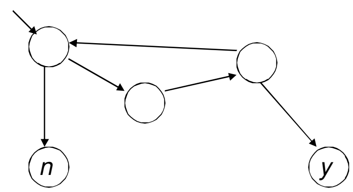
Theorem: The set D is closed under complement.

Proof: (by construction)

M:



M′:



This works because, by definition, *M* is:

- deterministic
- complete

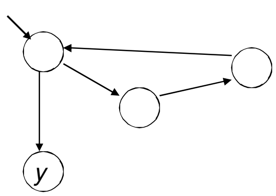
Since *M′* decides $\neg L$, $\neg L$ is in D.

Q8

SD is Not Closed Under Complement

Can we use the same technique?

M:



M′:

SD is Not Closed Under Complement

Suppose we had:

M_L:
Accepts if input is in L.

$M_{\neg L}$:
Accepts if input not in L.

Then we could decide L. How?

So every language in SD would also be in D.

But we know that there is at least one language (H) that is in SD but not in D. Contradiction.

D and SD Languages

Theorem: A language is in D iff both it and its complement are in SD.

Proof:

- L in D implies L and $\neg L$ are in SD:
  - L is in SD because D $\subset$ SD.
  - D is closed under complement
  - So $\neg L$ is also in D and thus in SD.

- L and $\neg L$ are in SD implies L is in D:
  - M_1 semidecides L.
  - M_2 semidecides $\neg L$.
  - To decide L:
    - Run M_1 and M_2 in parallel on w.
    - Exactly one of them will eventually accept.

A Particular Language that is Not in SD

Theorem: The language $\neg H$ =

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

is not in SD.

Proof:

- H is in SD.
- If $\neg H$ were also in SD then H would be in D.
- But H is not in D.
- So $\neg H$ is not in SD.

Q9

Enumeration

Enumerate means "list, in such a way that for any element, it appears in the list within a finite amount of time."

We say that Turing machine M ***enumerates*** the language L iff, for some fixed state p of M:

$$L = \{w : (s, \varepsilon) \mid\!-_M{}^* (p, w)\}.$$

"p" stands for "print"

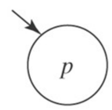
A language is **Turing-enumerable** iff there is a Turing machine that enumerates it.

Another term that is often used is **recursively enumerable**.

Q10

A Printing Subroutine

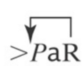
Let *P* be a Turing machine that enters state *p* and then halts:



Example of Enumeration

Let L = a*.

M_1:



M_2:

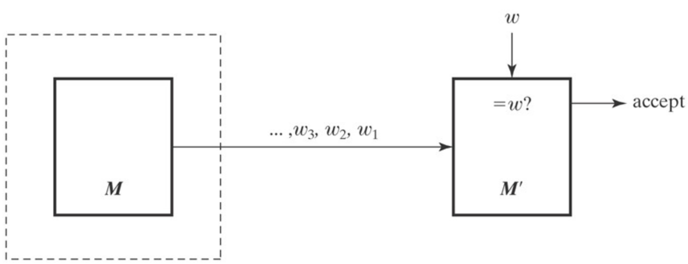
Q11

SD and Turing Enumerable

Theorem: A language is SD iff it is Turing-enumerable.

Proof that Turing-enumerable implies SD*:* Let *M* be the Turing machine that enumerates *L*. We convert *M* to a machine *M'* that semidecides *L*:

1. Save input *w on another tape*.
2. Begin enumerating *L*. Each time an element of *L* is enumerated, compare it to *w*. If they match, accept.



The Other Way

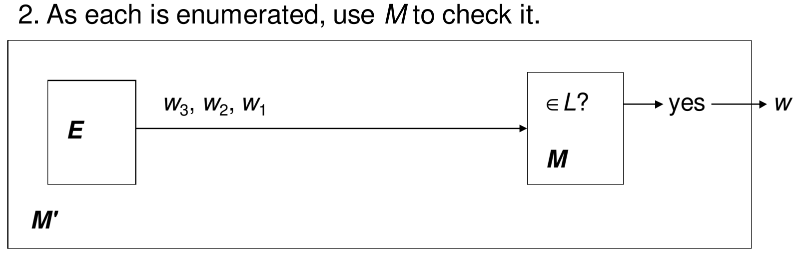
Proof that SD implies Turing-enumerable:

If $L \subseteq \Sigma^*$ is in SD, then there is a Turing machine *M* that semidecides *L*.

A procedure *E* to enumerate all elements of *L*:

1. Enumerate all $w \in \Sigma^*$ lexicographically.
   e.g., ε, `a`, `b`, `aa`, `ab`, `ba`, `bb`, …

2. As each is enumerated, use *M* to check it.



Problem?

Dovetailing

ε [1]					
ε [2]	a [1]				
ε [3]	a [2]	b [1]			
ε [4]	a [3]	b [2]	aa [1]		
ε [5]	a [4]	b [3]	aa [2]	ab [1]	
ε [6]	a [5]		aa [3]	ab [2]	ba [1]

The Other Way

Proof that SD implies Turing-enumerable:

If $L \subseteq \Sigma^*$ is in SD, then there is a Turing machine M that semidecides L.

A procedure to enumerate all elements of L:

1. Enumerate all $w \in \Sigma^*$ lexicographically.
2. As each string w_i is enumerated:
   1. Start up a copy of M with w_i as its input.
   2. Execute one step of each M_i initiated so far, excluding only those that have previously halted.
   3. Whenever an M_i accepts, output w_i.

Lexicographic Enumeration

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.

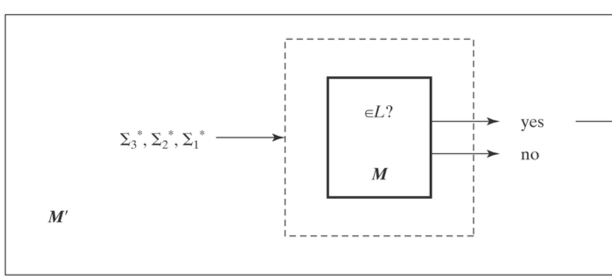
Example: $A^nB^nC^n = \{a^nb^nc^n : n \geq 0\}$

Lexicographic enumeration:

Lexicographically Enumerable = D

Theorem: A language is in D iff it is lexicographically Turing-enumerable.

Proof that D implies lexicographically TE: Let *M* be a Turing machine that decides *L*. Then *M'* lexicographically generates the strings in Σ^* and tests each using *M*. It outputs those that are accepted by *M*. Thus *M'* lexicographically enumerates *L*.



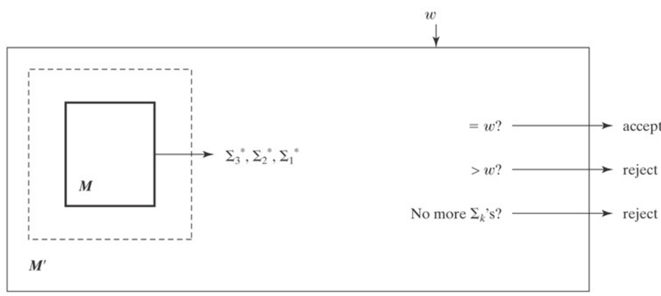
Proof, Continued

Proof that lexicographically Turing Enumerable implies D:

Let M be a Turing machine that lexicographically enumerates L. Then, on input w, M' starts up M and waits until:

- M generates w (so M' accepts),
- M generates a string that comes after w (so M' rejects), or
- M halts (so M' rejects).

Thus M' decides L.



Language Summary

IN
Semideciding TM
Enumerable
Unrestricted grammar

SD
H

OUT
Reduction

Deciding TM
Lexic. enum
L and $\neg L$ in SD

D
$A^nB^nC^n$

Diagonalize
Reduction

CF grammar
PDA
Closure

Context-Free
A^nB^n

Pumping
Closure

Regular Expression
FSM

Regular
`a*b*`

Pumping
Closure