MA/CSSE 474
Theory of Computation

Halting Problem
Decidable and Semidecidable

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
```

Collatz function example

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, ***9232***, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

The Language H

Theorem: The language:

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

- is semidecidable, but
- is not decidable.

Proof soon! *via* two lemmas ..

We know that we can decide the halting question for *specific* simple TMs.

Or can we … ?

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*.
2. Accept.

Details of step 1:
- Write <M,w> on U's first tape.
- Run U

U is the Universal Turing Machine

M'_H accepts <*M*, *w*> if and only if *M* halts on *w*.
Thus M'_H semidecides H.

What do we mean by "halts on w"?

H is Not Decidable

Lemma: The language:

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

is not decidable.

Outline of proof:
By contradiction…
Specification of *halts function.*
Trouble ***[in (Wabash) River City)]***
halts(<*Trouble*, *Trouble*>) - what happens?

The Undecidability of the Halting Problem

Lemma: The language:

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

is not decidable.

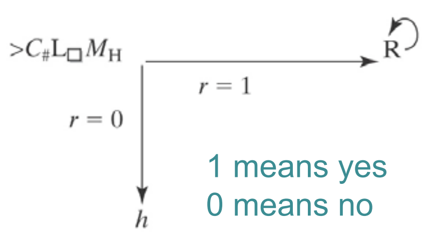
Proof (by contradiction): Assume that H is decidable. Then some TM M_H would decide it. M_H would implement the specification:

halts(<*M*, *w*>) =
  if <*M*> is a Turing machine description
      and *M* halts on *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:



1 means yes
0 means no

C# is the machine from several class sessions ago that 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 M_H reports that *Trouble*(<*Trouble*>) halts, *Trouble* loops.
- But if M_H reports that *Trouble*(<*Trouble*>) does not halt, then *Trouble* halts.

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?

Decidable and Semidecidable Languages

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 M_H 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 M_H on <M_L, w>. (M_H will always halt)
2. If M_H accepts (i.e., M_L will halt on input w), then:
    - 2.1. Run M_L on w.
    - 2.2. If it accepts, accept.
    - 2.3 Else reject.
3. Else reject.

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

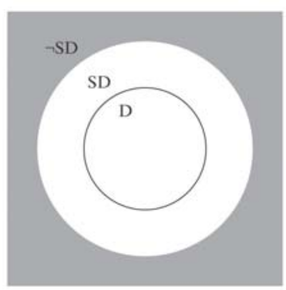
Almost every obvious language that is in SD is also in D:

- $A^nB^nC^n = \{a^nb^nc^n, n \geq 0\}$
- $\{wcw, w \in \{a, b\}^*\}$
- $\{ww, w \in \{a, b\}^*\}$
- $\{x*y=z: x,y,z \in \{0, 1\}^*$ 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:

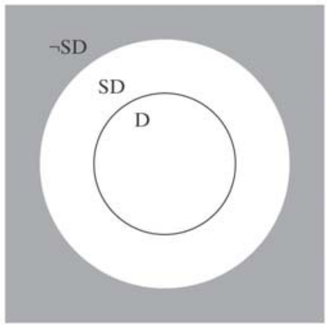
- H = {<*M*, *w*> : *M* halts on input *w*}

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 (namely, H) that is in SD-D, the donut in the picture.

Subset Relationships between D and SD



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

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 of D). *M* also semidecides L.

3. What about languages that are not in SD? Is the gray area of the figure empty?

There are Languages That Are Not in SD

Theorem: 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.

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

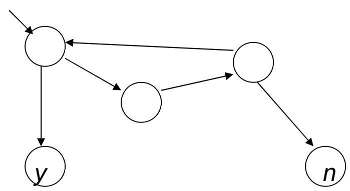
So there are more languages than there are languages in SD. Thus there must exist at least one language that is in ¬SD.

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:



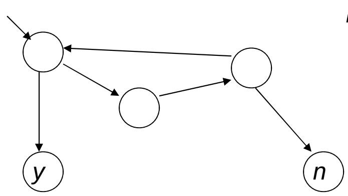
From *M*, we construct *M′* to decide ¬*L*:

Closure of D Under Complement

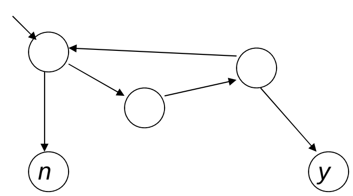
Theorem: The set D is closed under complement.

Proof: (by construction)

M:



M′:



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

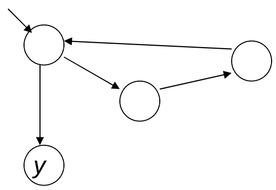
- deterministic
- complete

Since *M′* decides ¬*L*, ¬*L* is in D.

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:

$\Rightarrow$

- 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.

$\Leftarrow$

- 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 (dovetail) on w.
    - Exactly one of them will eventually accept.

A Particular Language that is Not in SD

Theorem: The language $\neg H$ =

{<M, w> : TM M 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.

Enumeration

To enumerate a set means "list its elements, 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:

$$L = \{w : (s, \varepsilon) \vdash_M^* (p, w)\}.$$

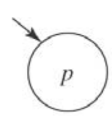
"p" stands for "print"

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

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

A Printing Subroutine

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



Example of Enumeration

Let *L* = a*.

M_1:

$>PaR$

M_2:

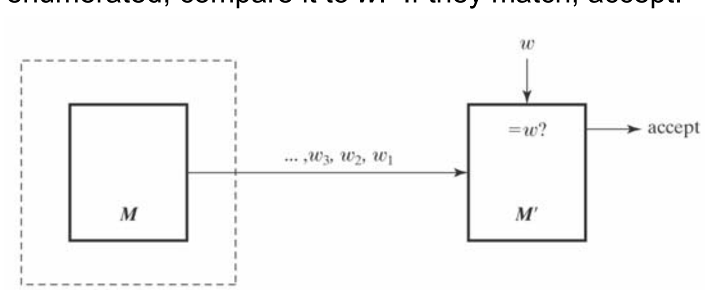
$>PaP\square RaRaRaP\square P$

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.



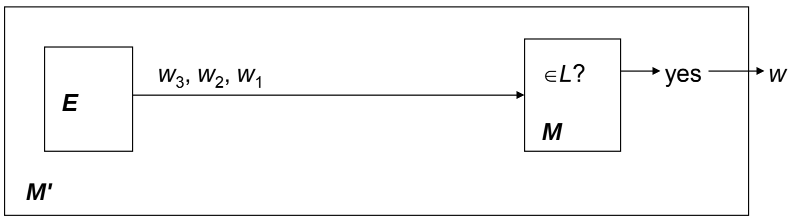
The Other Way

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

We have an infinite number of calculations C1, C2, C3, ..., each of which may or may not halt. We want to enumerate the results of those that halt.
A naive approach would be to run C1, then C2, ...
The problem with this is that C1 may not halt, so we may never get to try C2.

Solution: Run them in this order.
Step 1 of C1
Step 2 of C1
Step 1 of C2
Step 3 of C1
Step 2 of C2
Step 1 of C3
Step 4 of C1
Step 3 of C2
Step 2 of C3
Step 1 of C4
...
Then if Ci halts after j steps, we are guaranteed to eventually get to that step.

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 those M's that have already 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: $A^nB^nC^n = \{a^nb^nc^n : n \geq 0\}$

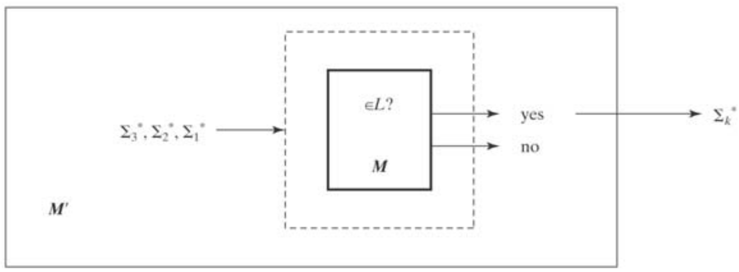
Lexicographic enumeration:

How would a TM do this.

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*.

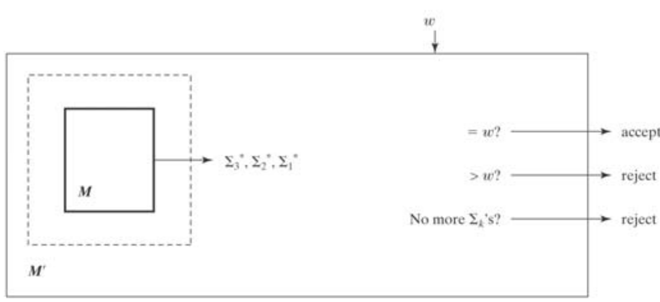


Proof, Continued

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		OUT
Semideciding TM Enumerable Unrestricted grammar	**SD** H	Reduction
Deciding TM Lexic. enum *L* and ¬*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