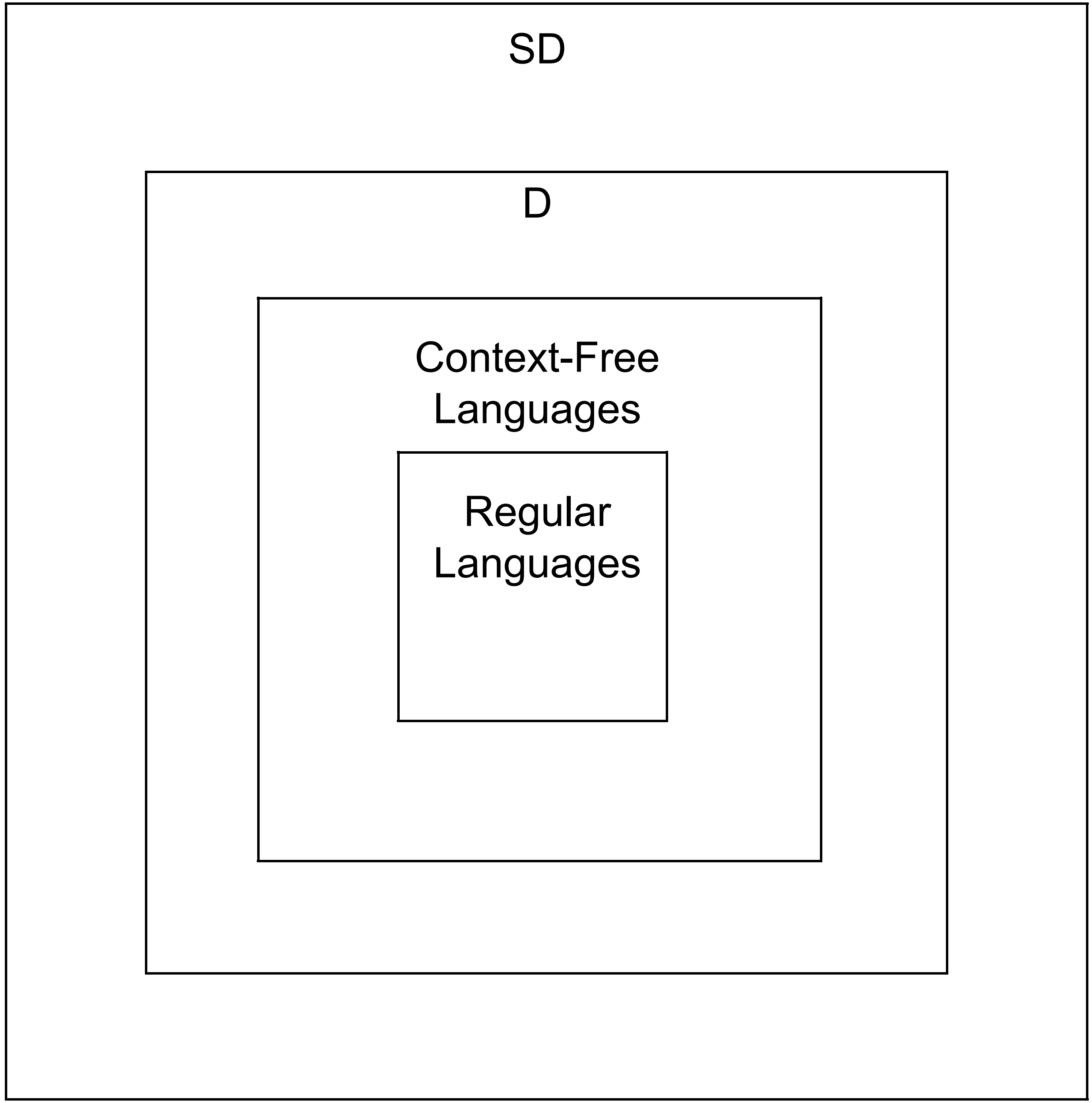
Decidable and Semidecidable Languages

Chapter 20

D and SD 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, AnBnCn, that is decidable

but not context-free.

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

Decidable and Semidecidable Languages

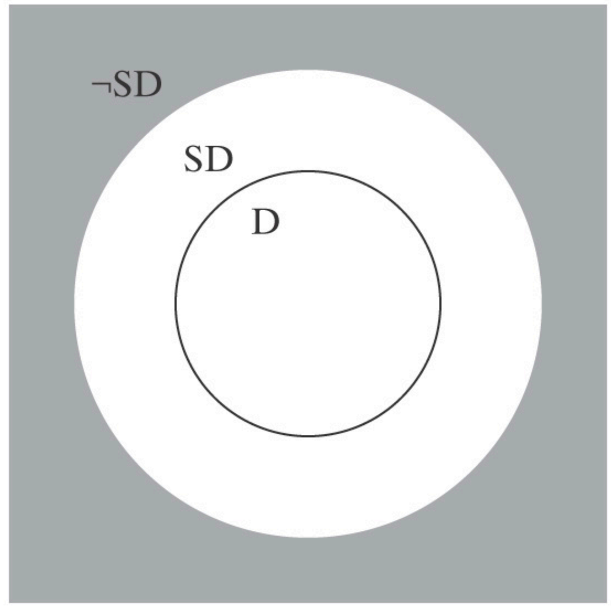
Almost every obvious language that is in SD is also in D:

- AnBnCn = $\{\mathtt{a}n\mathtt{b}n\mathtt{c}n,\ n \geq 0\}$
- $\{w\mathtt{c}w,\ w \in \{\mathtt{a}, \mathtt{b}\}^*\}$
- $\{ww,\ w \in \{\mathtt{a}, \mathtt{b}\}^*\}$
- $\{w = x \times y = z:\ x,y,z \in \{\mathtt{0}, \mathtt{1}\}^*$ 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:

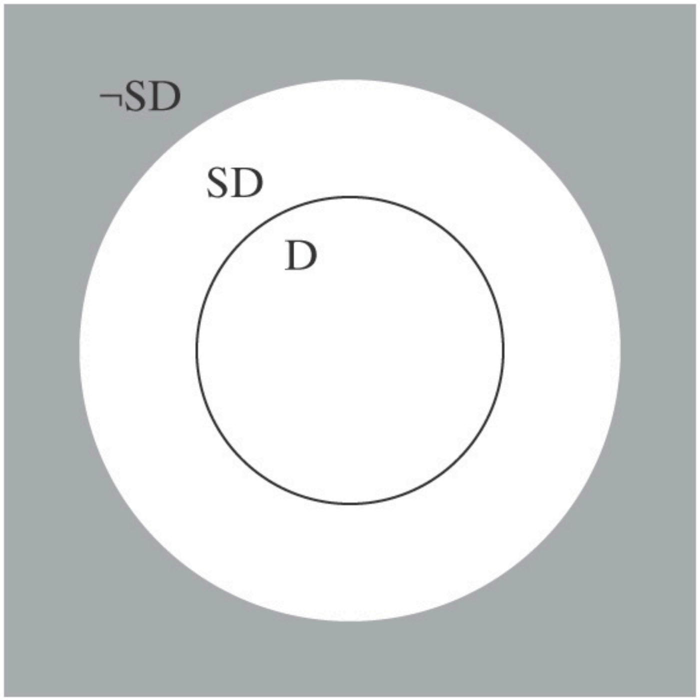
- H = $\{\langle M, w\rangle : M$ halts on input $w\}$

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 that is in SD/D, the donut in the picture.
3. There exist languages that are not in SD. In other words, the gray area of the figure is not 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 □.

1. There is a countably infinite number of SD languages over □.

2. There is an uncountably infinite number of languages over □.

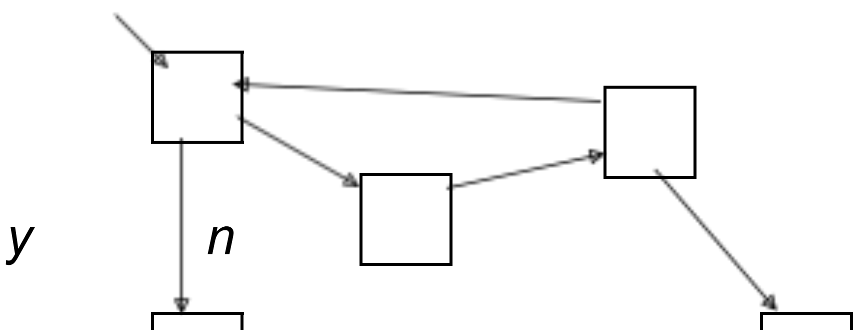
So there are more languages than there are languages in SD. Thus there must exist at least one language that is in □SD.

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:

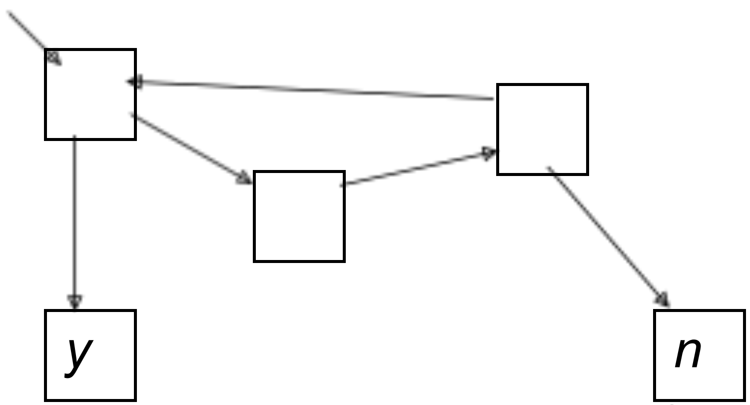


From *M*, we construct M' to decide $\neg L$:

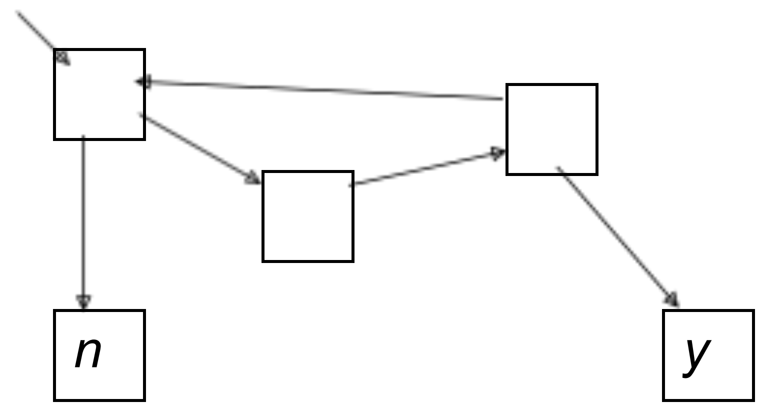
Closure of D Under Complement

Proof: (by construction)

M:



M':



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

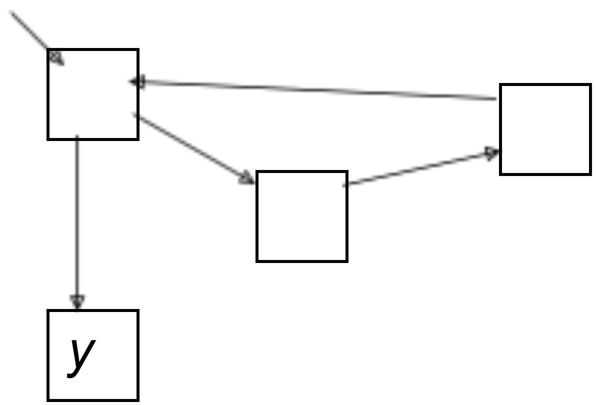
1. deterministic
2. complete

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:

ML: *M*□*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 the language and its complement are in SD.

Proof:

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

2. L and $\neg L$ are in SD implies L is in D:
   - $M1$ semidecides L.
   - $M2$ semidecides $\neg L$.
   - To decide L:
     - Run $M1$ and $M2$ in parallel on w.
     - Exactly one of them will eventually accept.

A 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

Enumerate means list. We look at Turing Machines as generators.

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

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

Whenever the machine enters *p*, the string on the tape is enumerated.
If *L* is finite, then *M* eventually halts.

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

Example of Enumeration

Consider a printing subroutine: *P* be a Turing machine that enters state *p* and then halts:

Let *L* = `a`*. *M1* and *M2* both enumerate *L:*

M_1:

$> P\mathrm{a}\mathrm{R}$

M_2:

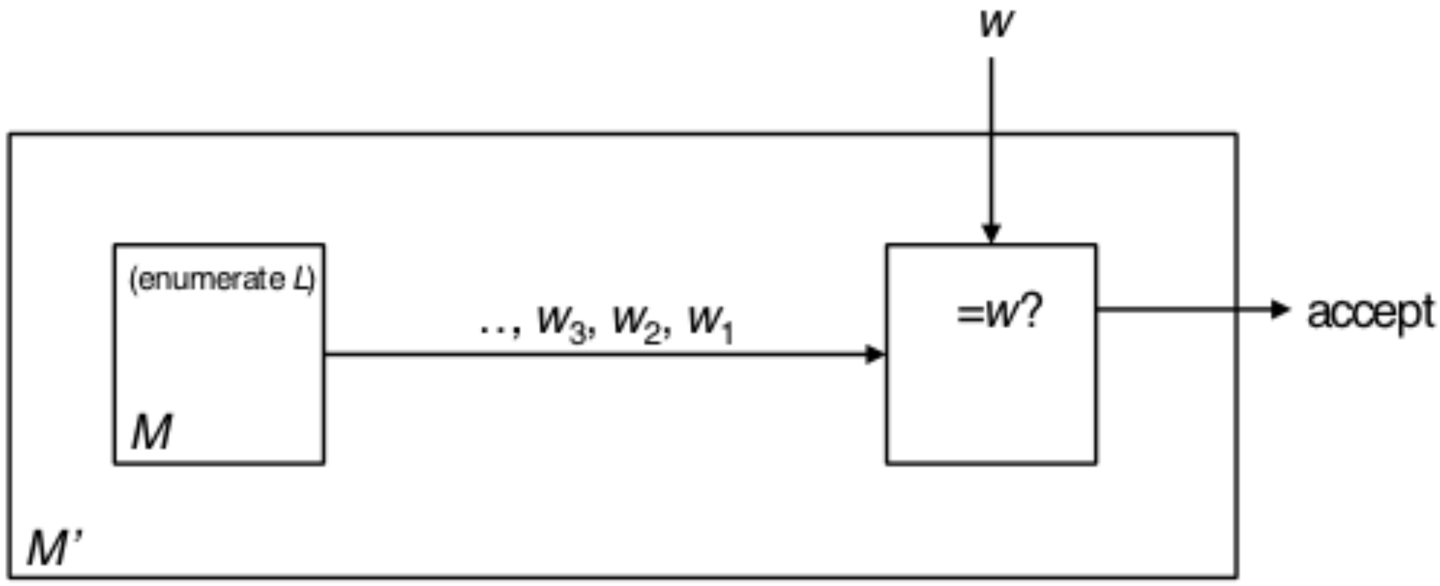
$> P\mathrm{a}P\square\mathrm{R}\mathrm{a}\mathrm{R}\mathrm{a}\mathrm{R}\mathrm{a}P\square P$

SD and Turing Enumerable

Theorem: A language is in 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.
2. Begin enumerating L. Each time an element of L is enumerated, compare it to w. If they match, accept.



The Other Way

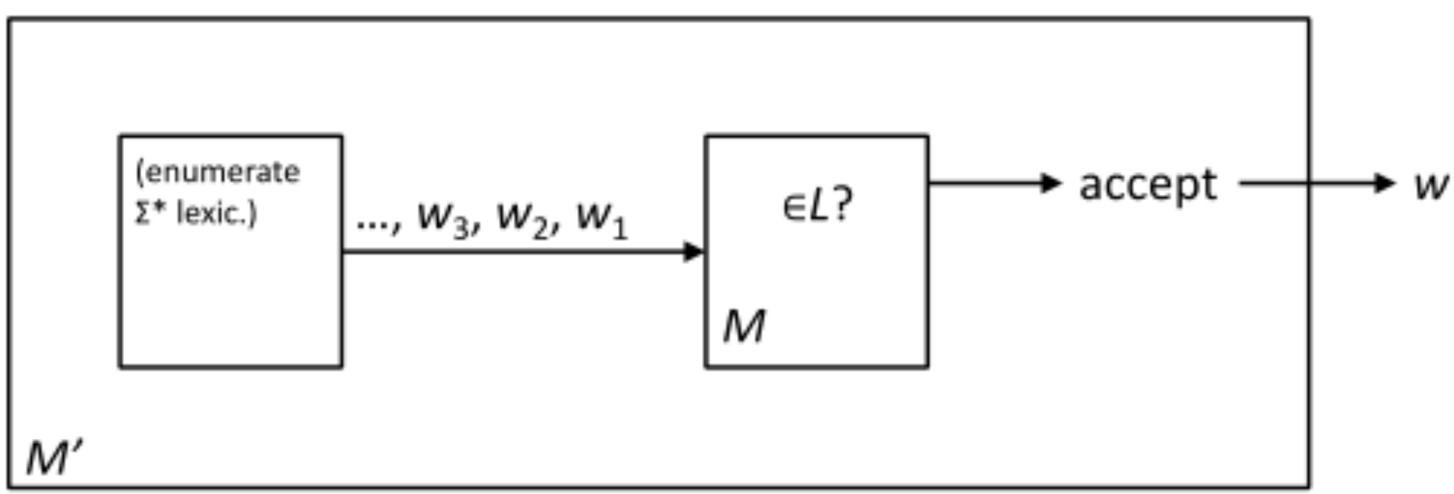
Proof that SD implies Turing-enumerable:

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.



Does this work?

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 wi is enumerated:
    1. Start up a copy of M with wi as its input.
    2. Execute one step of each Mi initiated so far, excluding only those that have previously halted.
    3. Whenever an Mi accepts, output wi.

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.

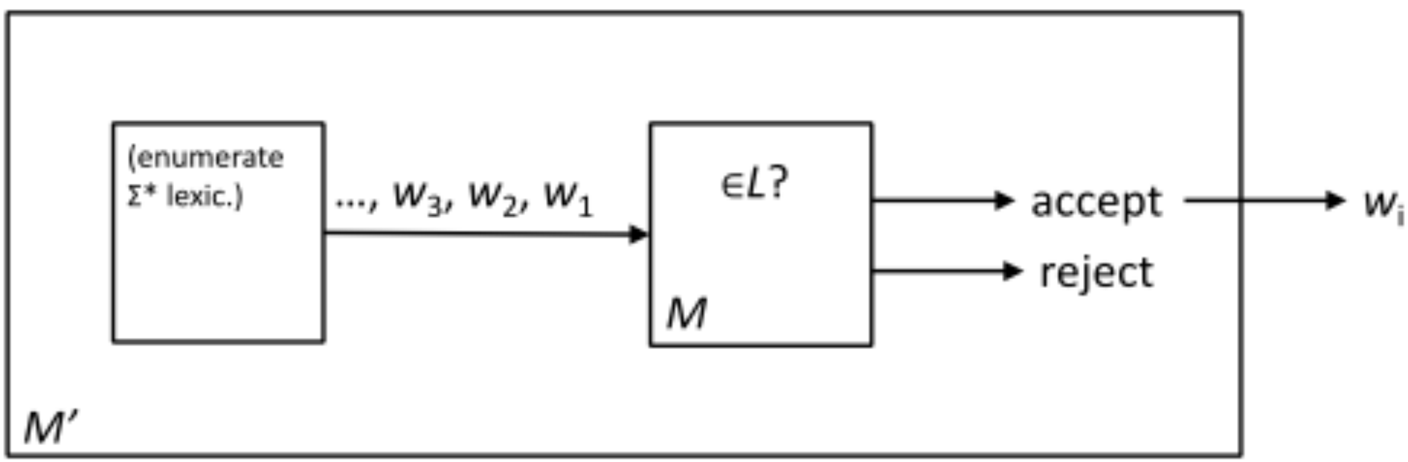
Example: AnBnCn = $\{a^n b^n c^n : n \geq 0\}$

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*.

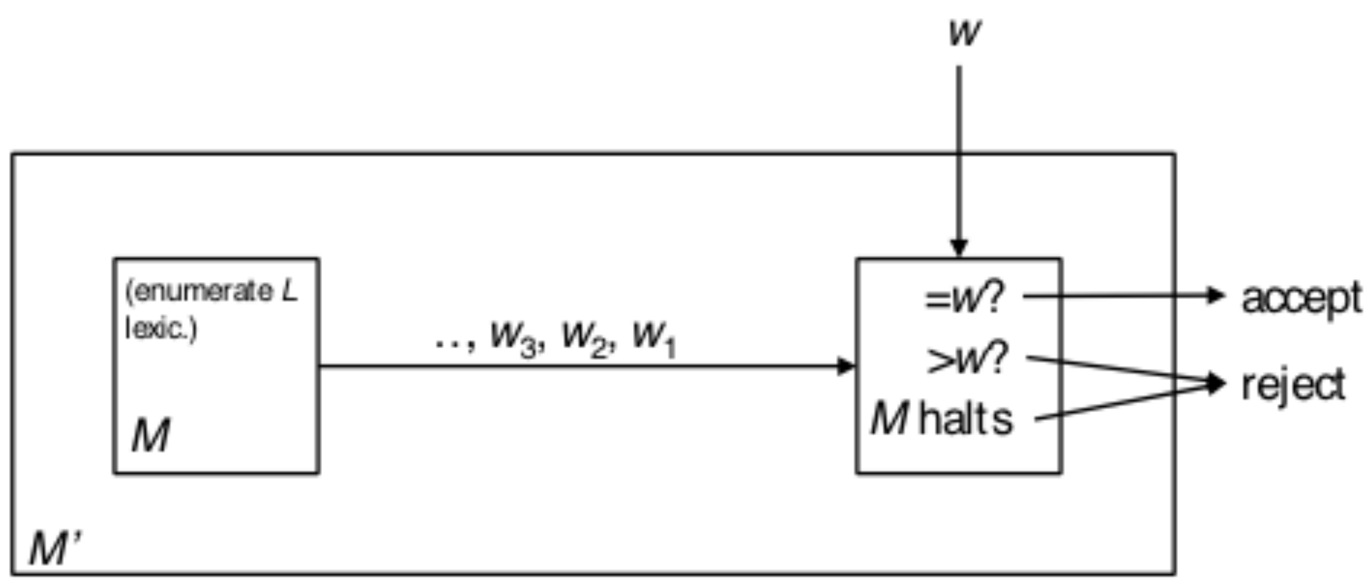


Proof, Continued

Proof that lexicographically TE 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 | **SD** | **OUT**

Semideciding TM | H | Reduction

Enumerable

D

Deciding TM | AnBnCn | Diagonalize

Lexic. enum | Reduction

L and ¬*L* in SD

Context-Free

CF grammar | AnBn | Pumping

PDA | Closure

Closure

Regular

Regular Expression | a*b* | Pumping

FSM | Closure