

Undecidability

Everything is an Integer
Countable and Uncountable Sets
Turing Machines
Recursive and Recursively
Enumerable Languages

Integers, Strings, and Other Things

- ◆ Data types have become very important as a programming tool.
- ◆ But at another level, there is only one type, which you may think of as integers or strings.
- ◆ **Key point:** Strings that are programs are just another way to think about the same one data type.

Example: Text

- ◆ Strings of ASCII or Unicode characters can be thought of as binary strings, with 8 or 16 bits/character.
- ◆ Binary strings can be thought of as integers.
- ◆ It makes sense to talk about “the i -th string.”

Binary Strings to Integers

- ◆ There's a small glitch:
 - ◆ If you think simply of binary integers, then strings like 101, 0101, 00101,... all appear to be "the fifth string."
- ◆ Fix by prepending a "1" to the string before converting to an integer.
 - ◆ Thus, 101, 0101, and 00101 are the 13th, 21st, and 37th strings, respectively.

Example: Images

- ◆ Represent an image in (say) GIF.
- ◆ The GIF file is an ASCII string.
- ◆ Convert string to binary.
- ◆ Convert binary string to integer.
- ◆ Now we have a notion of “the i -th image.”

Example: Proofs

- ◆ A formal proof is a sequence of logical expressions, each of which follows from the ones before it.
- ◆ Encode mathematical expressions of any kind in Unicode.
- ◆ Convert expression to a binary string and then an integer.

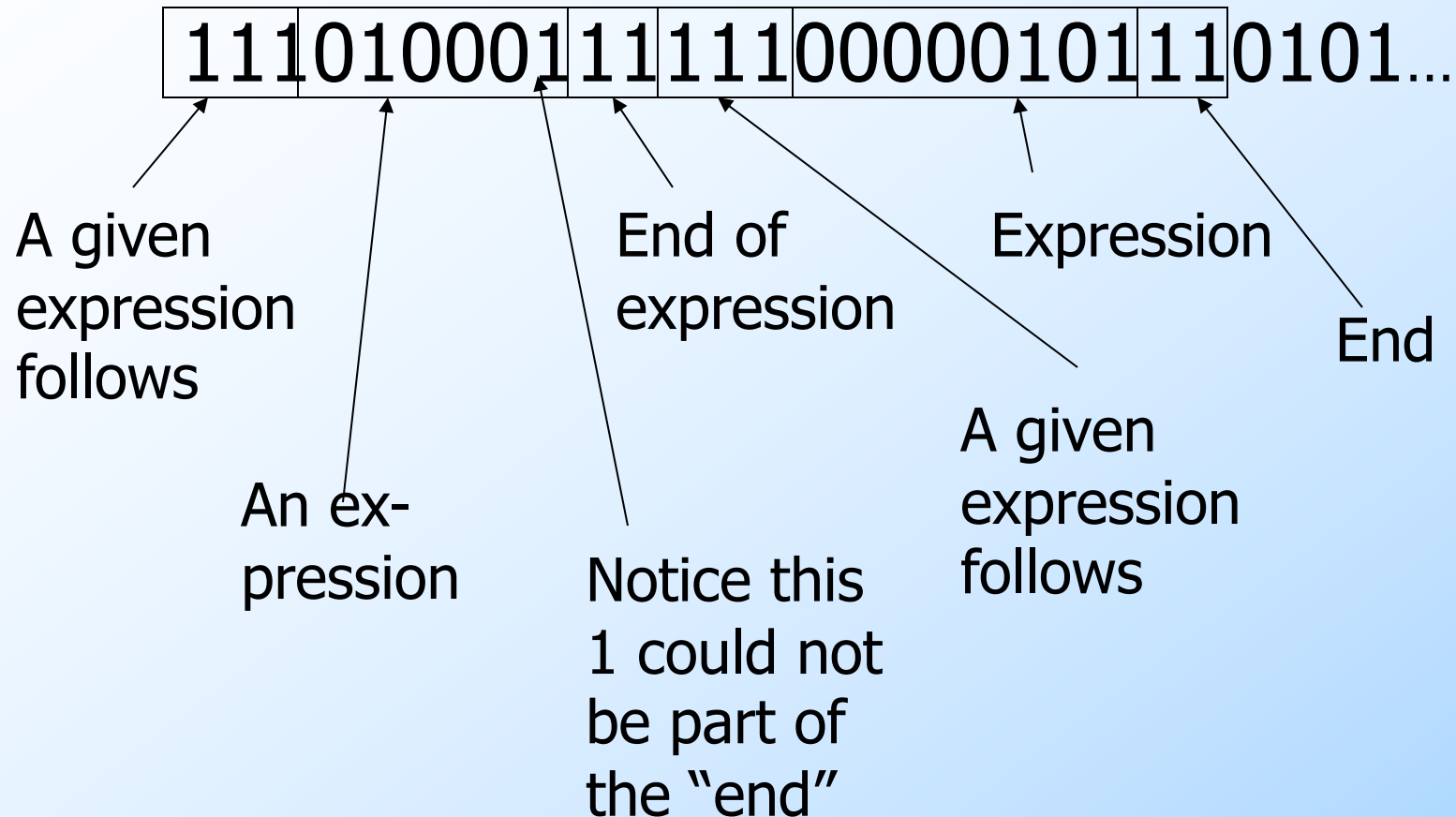
Proofs – (2)

- ◆ But a proof is a sequence of expressions, so we need a way to separate them.
- ◆ Also, we need to indicate which expressions are given and which follow from previous expressions.

Proofs – (3)

- ◆ Quick-and-dirty way to introduce new symbols into binary strings:
 1. Given a binary string, precede each bit by 0.
 - ◆ **Example:** 101 becomes 010001.
 2. Use strings of two or more 1's as the special symbols.
 - ◆ **Example:** 111 = "the following expression is given"; 11 = "end of expression."

Example: Encoding Proofs



Example: Programs

- ◆ Programs are just another kind of data.
- ◆ Represent a program in ASCII.
- ◆ Convert to a binary string, then to an integer.
- ◆ Thus, it makes sense to talk about “the i -th program.”
- ◆ Hmm...There aren't all that many programs.

Finite Sets

- ◆ A *finite set* has a particular integer that is the count of the number of members.
- ◆ **Example:** $\{a, b, c\}$ is a finite set; its *cardinality* is 3.
- ◆ It is impossible to find a 1-1 mapping between a finite set and a proper subset of itself.

Infinite Sets

- ◆ Formally, an *infinite set* is a set for which there is a 1-1 correspondence between itself and a proper subset of itself.
- ◆ **Example:** the positive integers $\{1, 2, 3, \dots\}$ is an infinite set.
 - ◆ There is a 1-1 correspondence $1 \leftrightarrow 2, 2 \leftrightarrow 4, 3 \leftrightarrow 6, \dots$ between this set and a proper subset (the set of even integers).

Countable Sets

- ◆ A *countable set* is a set with a 1-1 correspondence with the positive integers.
 - ◆ Hence, all countable sets are infinite.
- ◆ **Example:** All integers.
 - ◆ $0 \leftrightarrow 1; -i \leftrightarrow 2i; +i \leftrightarrow 2i+1.$
 - ◆ Thus, order is 0, -1, 1, -2, 2, -3, 3,...
- ◆ **Examples:** set of binary strings, set of Java programs.

Example: Pairs of Integers

- ◆ Order the pairs of positive integers first by sum, then by first component:
- ◆ $[1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2], \dots, [1,4], [5,1], \dots$
- ◆ **Interesting exercise:** figure out the function $f(i,j)$ such that the pair $[i,j]$ corresponds to the integer $f(i,j)$ in this order.

Enumerations

- ◆ An *enumeration* of a set is a 1-1 correspondence between the set and the positive integers.
- ◆ Thus, we have seen enumerations for strings, programs, proofs, and pairs of integers.

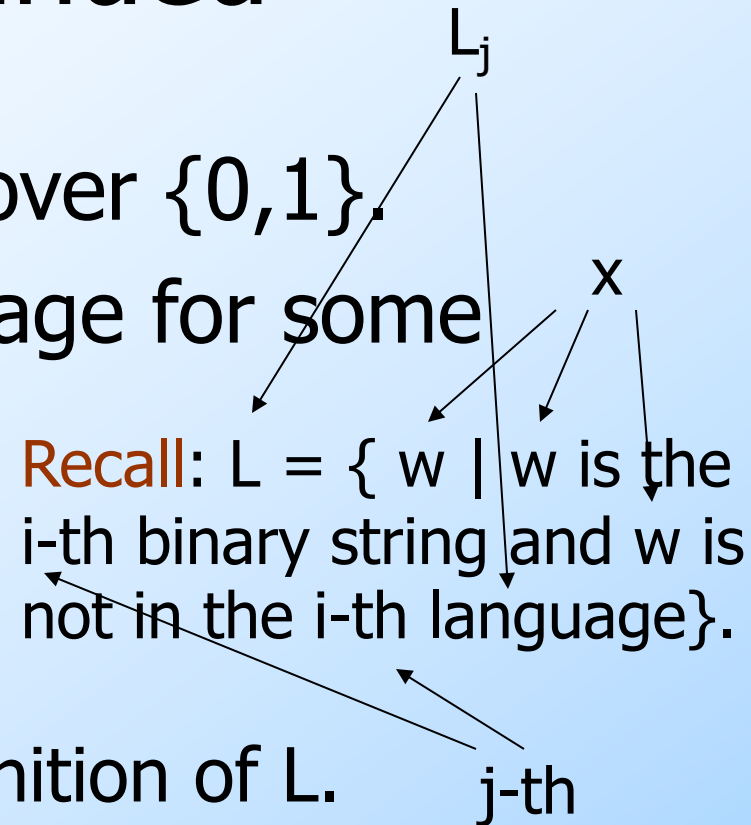
How Many Languages?

- ◆ Are the languages over $\{0,1\}$ countable?
- ◆ No; here's a **proof**.
- ◆ Suppose we could enumerate all languages over $\{0,1\}$ and talk about “the i -th language.”
- ◆ Consider the language $L = \{ w \mid w \text{ is the } i\text{-th binary string and } w \text{ is not in the } i\text{-th language} \}$.

Proof – Continued

- ◆ Clearly, L is a language over $\{0,1\}$.
- ◆ Thus, it is the j -th language for some particular j .
- ◆ Let x be the j -th string.
- ◆ Is x in L ?

- ▶ If so, x is not in L by definition of L .
- ▶ If not, then x is in L by definition of L .



Proof – Concluded

- ◆ We have a contradiction: x is neither in L nor not in L , so our sole assumption (that there was an enumeration of the languages) is wrong.
- ◆ **Comment:** This is really bad; there are more languages than programs.
- ◆ E.g., there are languages with no membership algorithm.

Diagonalization Picture

		Strings					
		1	2	3	4	5	...
Languages	1	1	0	1	1	0	...
	2		1				
	3			0			
	4				0		
	5					1	

Diagonalization Picture

Flip each
diagonal
entry

Languages

Strings						
	1	2	3	4	5	...
1	0	0	1	1	0	..
2		0				
3			1			
4				1		
5					0	
...						...

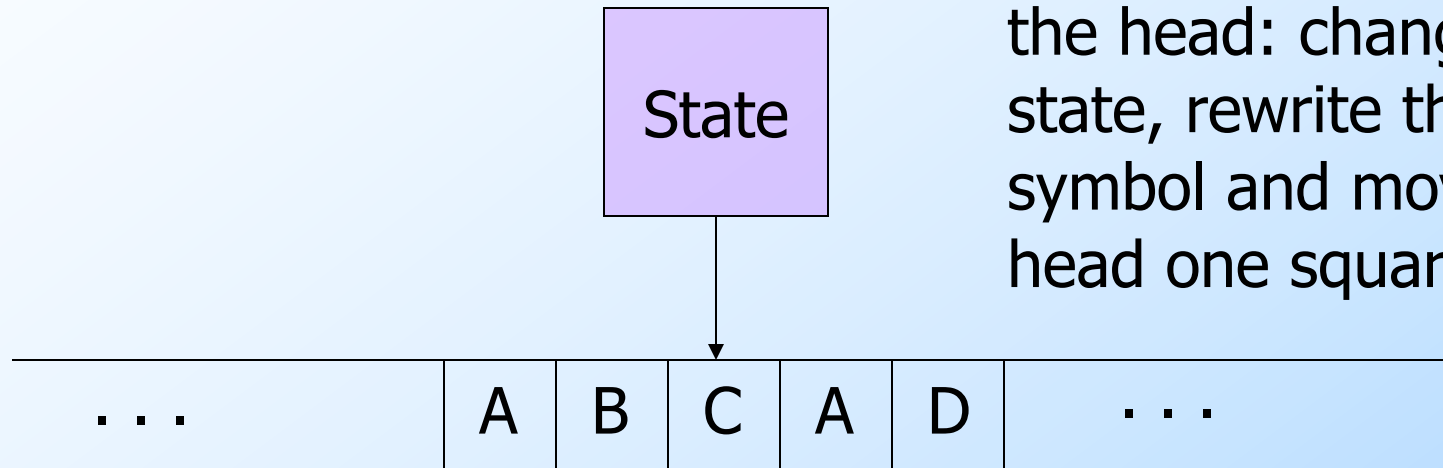
Can't be
a row –
it disagrees
in an entry
of each row.

Turing-Machine Theory

- ◆ The purpose of the theory of Turing machines is to prove that certain specific languages have no algorithm.
- ◆ Start with a language about Turing machines themselves.
- ◆ Reductions are used to prove more common questions undecidable.

Picture of a Turing Machine

Action: based on the state and the tape symbol under the head: change state, rewrite the symbol and move the head one square.



Infinite tape with squares containing tape symbols chosen from a finite alphabet

Why Turing Machines?

- ◆ Why not deal with C programs or something like that?
- ◆ **Answer:** You can, but it is easier to prove things about TM's, because they are so simple.
 - ◆ And yet they are as powerful as any computer.
 - More so, in fact, since they have infinite memory.

Turing-Machine Formalism

- ◆ A TM is described by:
 1. A finite set of *states* (Q , typically).
 2. An *input alphabet* (Σ , typically).
 3. A *tape alphabet* (Γ , typically; contains Σ).
 4. A *transition function* (δ , typically).
 5. A *start state* (q_0 , in Q , typically).
 6. A *blank symbol* (B , in $\Gamma - \Sigma$, typically).
 - ◆ All tape except for the input is blank initially.
 7. A set of *final states* ($F \subseteq Q$, typically).

Conventions

- ◆ a, b, \dots are input symbols.
- ◆ \dots, X, Y, Z are tape symbols.
- ◆ \dots, w, x, y, z are strings of input symbols.
- ◆ α, β, \dots are strings of tape symbols.

The Transition Function

- ◆ Takes two arguments:
 1. A state, in Q .
 2. A tape symbol in Γ .
- ◆ $\delta(q, Z)$ is either undefined or a triple of the form (p, Y, D) .
 - ◆ p is a state.
 - ◆ Y is the new tape symbol.
 - ◆ D is a *direction*, L or R.

Example: Turing Machine

- ◆ This TM scans its input right, looking for a 1.
- ◆ If it finds one, it changes it to a 0, goes to final state f , and halts.
- ◆ If it reaches a blank, it changes it to a 1 and moves left.

Example: Turing Machine – (2)

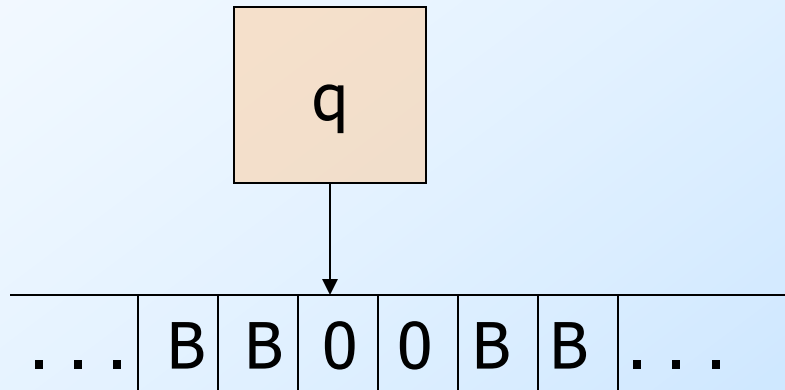
- ◆ States = $\{q \text{ (start), } f \text{ (final)}\}$.
- ◆ Input symbols = $\{0, 1\}$.
- ◆ Tape symbols = $\{0, 1, B\}$.
- ◆ $\delta(q, 0) = (q, 0, R)$.
- ◆ $\delta(q, 1) = (f, 0, R)$.
- ◆ $\delta(q, B) = (q, 1, L)$.

Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

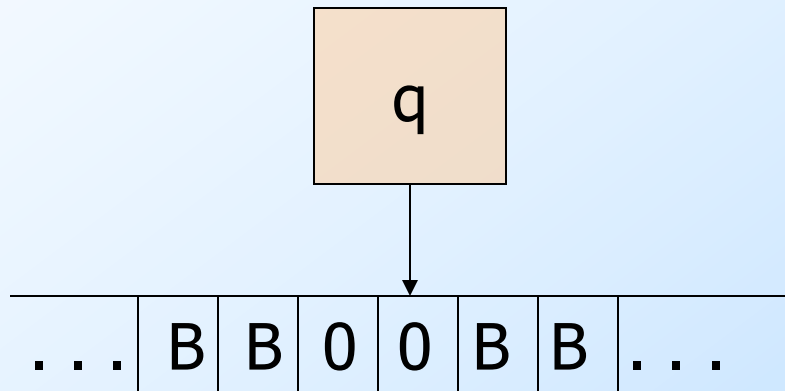


Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

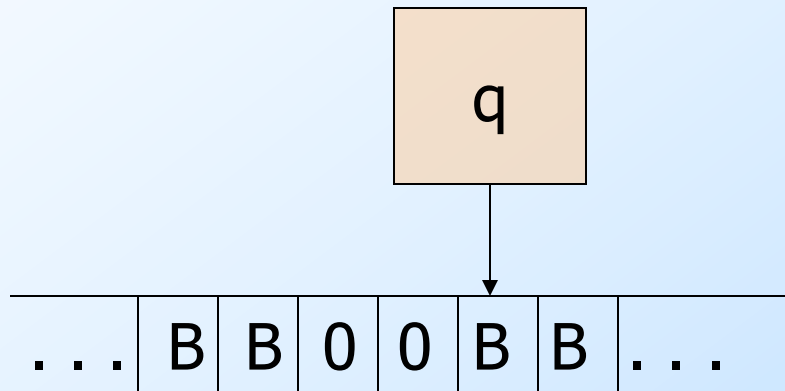


Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

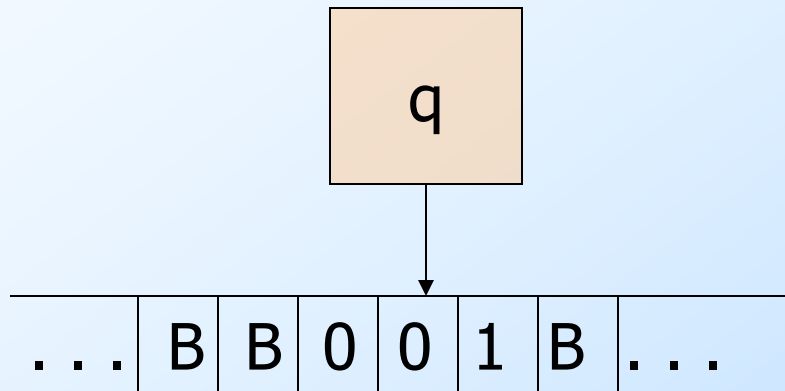


Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

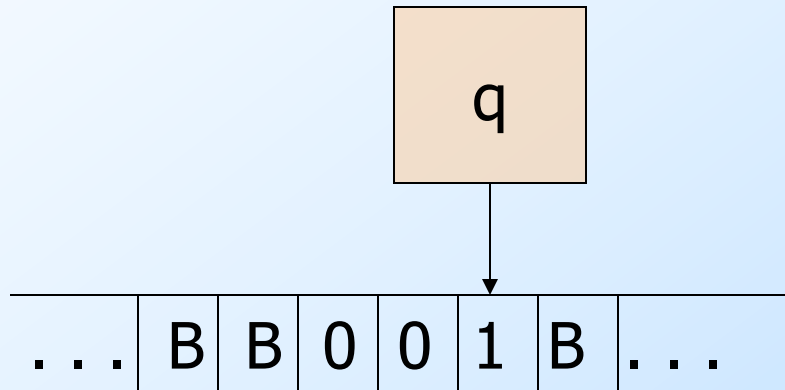


Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

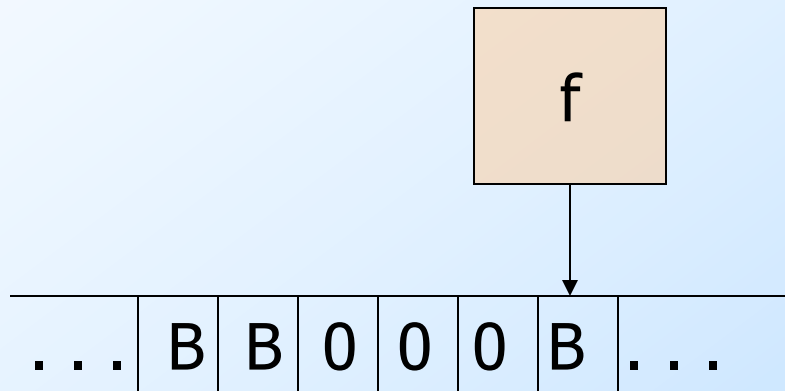


Simulation of TM

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$



No move is possible.
The TM halts and
accepts.

Instantaneous Descriptions of a Turing Machine

- ◆ Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- ◆ The TM is in the start state, and the head is at the leftmost input symbol.

TM ID's – (2)

- ◆ An ID is a string $\alpha q \beta$, where $\alpha \beta$ includes the tape between the leftmost and rightmost nonblanks.
- ◆ The state q is immediately to the left of the tape symbol scanned.
- ◆ If q is at the right end, it is scanning B .
 - ◆ If q is scanning a B at the left end, then consecutive B 's at and to the right of q are part of α .

TM ID's – (3)

- ◆ As for PDA's we may use symbols \vdash and \vdash^* to represent "becomes in one move" and "becomes in zero or more moves," respectively, on ID's.
- ◆ **Example:** The moves of the previous TM are $q_00 \vdash 0q_0 \vdash 00q \vdash 0q_01 \vdash 00q1 \vdash 000f$

Formal Definition of Moves

1. If $\delta(q, Z) = (p, Y, R)$, then
 - ◆ $\alpha q Z \beta \vdash \alpha Y p \beta$
 - ◆ If Z is the blank B , then also $\alpha q \vdash \alpha Y p$
2. If $\delta(q, Z) = (p, Y, L)$, then
 - ◆ For any X , $\alpha X q Z \beta \vdash \alpha p X Y \beta$
 - ◆ In addition, $q Z \beta \vdash p B Y \beta$

Languages of a TM

- ◆ A TM defines a language by final state, as usual.
- ◆ $L(M) = \{w \mid q_0 w \vdash^* I, \text{ where } I \text{ is an ID with a final state}\}.$
- ◆ Or, a TM can accept a language by halting.
- ◆ $H(M) = \{w \mid q_0 w \vdash^* I, \text{ and there is no move possible from ID } I\}.$

Equivalence of Accepting and Halting

1. If $L = L(M)$, then there is a TM M' such that $L = H(M')$.
2. If $L = H(M)$, then there is a TM M'' such that $L = L(M'')$.

Proof of 1: Final State -> Halting

- ◆ Modify M to become M' as follows:
 1. For each final state of M , remove any moves, so M' halts in that state.
 2. Avoid having M' accidentally halt.
 - ◆ Introduce a new state s , which runs to the right forever; that is $\delta(s, X) = (s, X, R)$ for all symbols X .
 - ◆ If q is not a final state, and $\delta(q, X)$ is undefined, let $\delta(q, X) = (s, X, R)$.

Proof of 2: Halting \rightarrow Final State

- ◆ Modify M to become M'' as follows:
 1. Introduce a new state f , the only final state of M'' .
 2. f has no moves.
 3. If $\delta(q, X)$ is undefined for any state q and symbol X , define it by $\delta(q, X) = (f, X, R)$.

Recursively Enumerable Languages

- ◆ We now see that the classes of languages defined by TM's using final state and halting are the same.
- ◆ This class of languages is called the *recursively enumerable languages*.
 - Why? The term actually predates the Turing machine and refers to another notion of computation of functions.

Recursive Languages

- ◆ An *algorithm* is a TM, accepting by final state, that is guaranteed to halt whether or not it accepts.
- ◆ If $L = L(M)$ for some TM M that is an algorithm, we say L is a *recursive language*.
 - Why? Again, don't ask; it is a term with a history.

Example: Recursive Languages

- ◆ Every CFL is a recursive language.
 - ◆ Use the CYK algorithm.
- ◆ Almost anything you can think of is recursive.