Def: **DTIME**, **NTIME**, **DSPACE**, measured on **Multi-tape Turing Machines**.

Th: $\textbf{DTIME}[t(n)] \subseteq \text{RAM-TIME}[t(n)] \subseteq \textbf{DTIME}[(t(n))^3]$

$$\begin{aligned}
\textbf{L} &\equiv \textbf{DSPACE}[\log n] \\
\textbf{P} &\equiv \textbf{DTIME}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \textbf{DTIME}[n^i] \\
\textbf{NP} &\equiv \textbf{NTIME}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \textbf{NTIME}[n^i] \\
\textbf{PSPACE} &\equiv \textbf{DSPACE}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \textbf{DSPACE}[n^i]
\end{aligned}$$

Th: For $t(n) \geq n, s(n) \geq \log n$,

$$\textbf{DTIME}[t(n)] \subseteq \textbf{NTIME}[t(n)] \subseteq \textbf{DSPACE}[t(n)]$$

$$\textbf{DSPACE}[s(n)] \subseteq \textbf{DTIME}[2^{O(s(n))}]$$

Cor: $\textbf{L} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE}$

$$\textbf{NTIME}[t(n)] \equiv \text{probs. accepted by NTMs in time } O(t(n))$$

$$\textbf{NP} \equiv \textbf{NTIME}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \textbf{NTIME}[n^i]$$

Theorem 9.1 *For any function* $t(n)$*,*

$$\textbf{DTIME}[t(n)] \subseteq \textbf{NTIME}[t(n)] \subseteq \textbf{DSPACE}[t(n)]$$

Proof: The first inclusion is obvious. For the second, note that in space $O(t(n))$ we can simulate *all* computations of length $O(t(n))$, so we will find the shortest accepting one if it exists. ♠

Recall: $\textbf{DSPACE}[t(n)] \subseteq \textbf{DTIME}[2^{O(t(n))}]$

Corollary 9.2

$$\textbf{L} \subseteq \textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE}$$

So we can simulate NTM's by DTM's, at the cost of an exponential increase in the running time. It may be possible to improve this simulation, though no essentially better one is known. If the cost could be reduced to polynomial, we would have that $\mathbf{P} = \mathbf{NP}$.

There is probably such a *quantitative* difference between the power of NTM's and DTM's. But note that *qualitatively* there is no difference. If A is the language of some NTM N, it must also be r.e. because there is a DTM that searches through all computations of N on w, first of one step, then of two steps, and so on. If $w \in A$, D will eventually find an accepting computation. If not, it will search forever.

What about an NTM-based definition of "recursive" or "Turing-decidable" sets? This is less clear because NTM's don't decide – they just have a range of possible actions. But one can define "a function computed by an NTM" in a reasonable way, and this leads to the same classes of partial recursive functions, total recursive functions, and recursive sets.

We show that a particular language *is* recursive or r.e. by exhibiting a Turing machine and showing that it decides or accepts the language. (Note that "exhibiting" means "proving the existence of" rather than "giving a state table for", and we may use high-level language to prove that existence.)

We want to show that certain languages *are not* recursive or r.e., which means showing that *no possible* Turing machine can decide or accept them.

The basic process will be to *reduce* an already unsolvable problem to the target problem, showing the latter unsolvable. But clearly we need to start somewhere, by showing *some* problem to be unsolvable in some other way.

Later in this lecture we'll see the standard example of an unsolvable problem, the **halting problem** of determining whether a given TM will halt on a given input. But first (since many of you have seen the halting problem before) we will show a different, related problem to be unsolvable.

Definition 9.3 Suppose we start an n-state TM, with tape alphabet $\{0, 1\}$, on a tape with all zeroes. We define the **busy beaver function** $\sigma(n)$ to be the maximum number of ones left on the tape by any of the n-state TM's that halt in this situation. (Note that to fit our definitions, "0" is now the "blank character".) ♠

Note that $\sigma(n)$ is well defined:

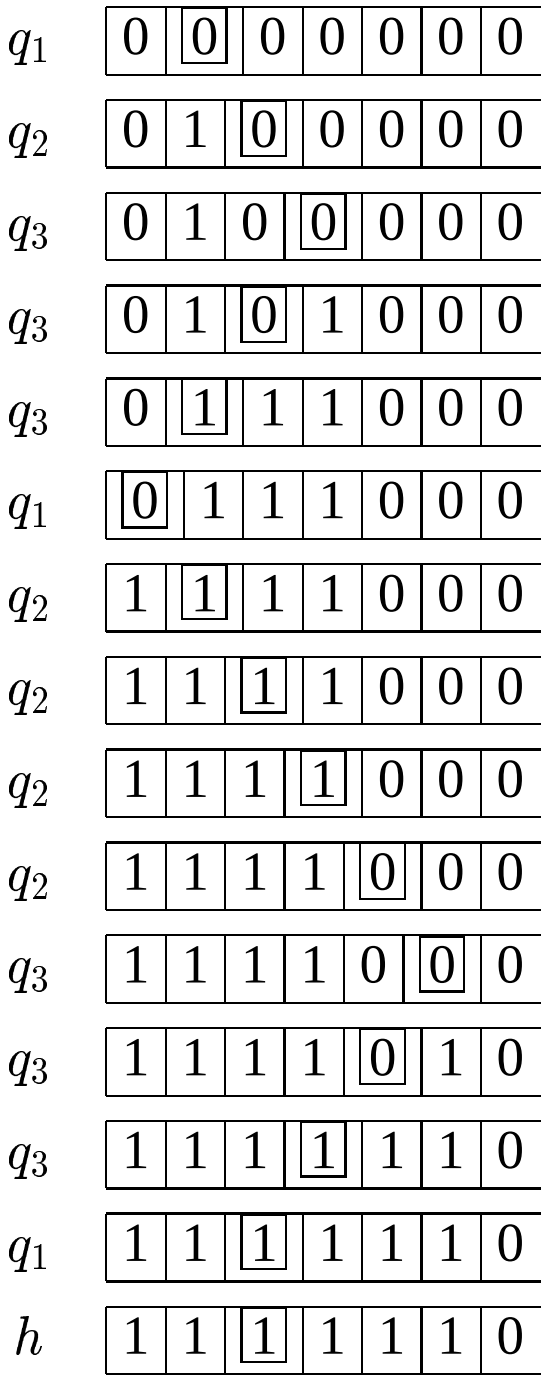
There are only finitely many n-state TMs, with $\Sigma = \{0, 1\}$.

Some finite subset, F_n, of these eventually halt on input 0.

Some element of F_n prints the maximum number of 1's on the tape, and this number is $\sigma(n)$.

	q_1	q_2	q_3
0	$q_2, 1, \rightarrow$	$q_3, 0, \rightarrow$	$q_3, 1, \leftarrow$
1	$h, 1, -$	$q_2, 1, \rightarrow$	$q_1, 1, \leftarrow$

$\sigma(3) \geq 6$



How quickly does $\sigma(n)$ grow as n gets large?

$$\begin{aligned} \text{Is } \sigma(n) \in \quad & O(n^2) \quad ? \\ & O(n^3) \quad ? \\ & O(2^n) \quad ? \\ & O(n!) \quad ? \\ & O(2^{2^n}) \quad ? \\ & O(\exp^*(n)) \quad ? \\ & O(\exp^*(\exp^*(n))) \quad ? \end{aligned}$$

$$\exp^*(n) \quad = \quad 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} \Bigg\}n$$

States	**Max # of 1's**	**Lower Bound for $\sigma(n)$**
3	$\sigma(3)$	6
4	$\sigma(4)$	13
5	$\sigma(5)$	≥ 4098
6	$\sigma(6)$	$> 10^{865}$

See the web pages of Penousal Machado (`eden.dei.uc.pt/~machado`) and Heiner Marxen (`www.drb.insel.de/~heiner/BB`) for more on this problem and its variants.

Theorem 9.4 *Let* $f : \mathbf{N} \to \mathbf{N}$ *be any total, recursive function. Then:*

$$\lim_{n\to\infty} \left(\frac{f(n)}{\sigma(n)}\right) \quad = \quad 0$$

That is, $f(n) = o(\sigma(n))$.

Proof:

Let

$$g(n) \quad = \quad n \cdot \left(1 + \sum_{i=0}^{n} f(i)\right)$$

Note:

$$\lim_{n\to\infty} \left(\frac{f(n)}{g(n)}\right) \quad = \quad 0$$

We will show that for all sufficiently large n,

$$\sigma(n) \quad \geq \quad g(n)$$

First note that since f is total recursive, and g depends on f in a simple way, it is easy to design a TM that computes $g(n)$. Let k be the number of states in this TM.

For any n, define the TM

$$C_n \;=\; \underbrace{\boxed{\text{print}\quad n}}_{\lceil \log n \rceil}\underbrace{\boxed{\text{compute } g}}_{k}\underbrace{\boxed{\begin{array}{c}\text{binary}\\ \text{to unary}\end{array}}}_{17}$$

C_n has $\lceil \log n \rceil + k + 17$ states.

C_n prints $g(n)$ 1's.

Once n is big enough that $n \geq \lceil \log n \rceil + k + 17$,

$$\sigma(n) \;\geq\; \sigma(\lceil \log n \rceil + k + 17) \;\geq\; g(n)$$

♠

We have mentioned **type casting** of the input or output of Turing machines. For example, we want to think of numbers as strings or strings as numbers, so we have 1-1, onto functions to convert one to another.

Very often we want to think of **pairs** or **sequences** of numbers as single numbers. So we need a function to convert a pair to a single number, and functions to take a number and find the left or right element of the pair it represents.

$$P : \mathbf{N} \times \mathbf{N} \overset{1:1}{\underset{onto}{\rightarrow}} \mathbf{N}$$

$$\begin{aligned} P(L(w), R(w)) &= w \\ L(P(i, j)) &= i \\ R(P(i, j)) &= j \end{aligned}$$

Thus, the input to a Turing machine is a single binary string which may be thought of as a natural number, a pair of natural numbers, a triple of natural numbers, and so forth. (Later we will worry about the *complexity* of the pairing and string-conversion functions – do you think they are in **L**)?

Some years ago a CMPSCI 601 student wrote a Java applet that calculates these functions. You are welcome to play with the applet at:

`.../~immerman/cs601/chandler.html`.

Turing machines can be encoded as **character strings** which can be encoded as **binary strings** which can be encoded as **natural numbers**.

TM_n	1	2	3	4
0	$1, 0, \rightarrow$	$3, \sqcup, \rightarrow$	$0, 0, -$	$0, 0, -$
1	$1, 1, \rightarrow$	$4, \sqcup, \rightarrow$	$0, 1, -$	$0, 1, -$
$\sqcup$	$2, \sqcup, \leftarrow$	$0, \sqcup, -$	$1, 0, \leftarrow$	$1, 1, \leftarrow$
$\triangleright$	$1, \triangleright, \rightarrow$	$0, \triangleright, -$	$0, \triangleright, -$	$0, \triangleright, -$

ASCII: $1, 0, \rightarrow; 1, 1, \rightarrow; 2, \sqcup, \leftarrow; 1, \triangleright, \rightarrow; ; \ \cdots \ 0, \triangleright, -$

$\{0,1\}^\star$: w

N : n

There is a simple, countable listing of all TM's:

$$M_0, M_1, M_2, \cdots$$

Theorem 9.5 *There is a* Universal Turing Machine U *such that,*

$$U(\langle n, m \rangle) \quad = \quad M_n(m)$$

Proof: Given $\langle n, m \rangle$, compute n and m. n is a binary string encoding the state table of TM M_n. We can simulate M_n on input m by keeping track of its state, its tape, and looking at its state table, n, at each simulated step. (Of course we may use multiple tapes to do this.)

♠

Brookshear's 1979 textbook has a complete diagram for a universal Turing machine on two pages. Lewis and Papadimitriou's 1981 book has a pretty complete description of one.

Let's now look at $L(U)$, the set of numbers $P(n, m)$ such that the Turing machine M_n eventually halts on input n. We'll call this language HALT. The existence of U proves that HALT is r.e., and now we will prove that it's not recursive.

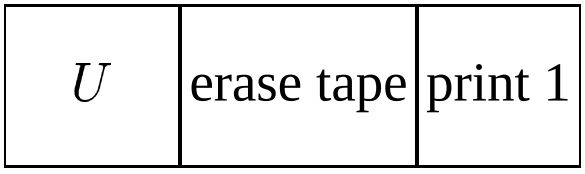
$$\text{HALT} \quad = \quad \{P(n,m) \mid \text{TM } M_n(m) \text{ eventually halts}\}$$

Theorem 9.6 (Unsolvability of the Halting Problem) HALT *is r.e. but not recursive.*

Proof: (First proof, based on busy-beaver result.)

$$\begin{aligned} \text{HALT} &= \{w \mid U(w) \text{ eventually halts}\} \\ &= \{w \mid U'(w) = 1\} \end{aligned}$$

$U' \quad =$



Suppose HALT were recursive. Then $\sigma(n)$ would be a total recursive function: Cycle through all n-state TMs, M_i, and if $P(i,0) \in$ HALT, then count the number of 1's in $M_i(0)$. Return the maximum of these. But $\sigma(n)$ *isn't* total recursive, so we have a contradiction.

♠

$$W_i \quad = \quad \{n \mid M_i(n) = 1\}$$

The set of all r.e. sets = $W_0, W_1, W_2, \cdots$ can be arranged in an infinite two-dimensional array:

n	0	1	2	3	4	5	6	7	8	$\cdots$	W_n
0	**0**	0	0	0	0	0	0	0	0	$\cdots$	W_0
1	1	**1**	1	1	1	1	1	1	1	$\cdots$	W_1
2	1	0	**1**	0	1	0	1	0	1	$\cdots$	W_2
3	0	1	0	**1**	0	1	0	1	0	$\cdots$	W_3
4	1	0	0	0	**0**	0	0	0	0	$\cdots$	W_4
5	0	1	1	0	1	**0**	0	0	1	$\cdots$	W_5
6	1	0	0	1	0	0	**1**	0	0	$\cdots$	W_6
7	1	1	0	0	0	0	0	**0**	0	$\cdots$	W_7
8	0	1	0	0	0	0	0	0	**0**	$\cdots$	W_8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
	0	1	1	1	0	0	1	0	0	$\cdots$	K
	1	0	0	0	1	1	0	1	1	$\cdots$	$\overline{K}$

$$\begin{aligned} K &= \{n \mid M_n(n) = 1\} \\ &= \{n \mid U(P(n,n)) = 1\} \\ &= \{n \mid n \in W_n\} \end{aligned}$$

Theorem 9.7 *$\overline{K}$ is not r.e.*

Proof: $\overline{K} = \{n \mid n \notin W_n\}$

Suppose $\overline{K}$ were r.e. Then for some c,

$$\overline{K} = W_c = \{n \mid M_c(n) = 1\}$$

$$c \in K \Leftrightarrow M_c(c) = 1 \Leftrightarrow c \in W_c \Leftrightarrow c \in \overline{K}$$

♠

Corollary 9.8 $K \in$ **r.e.** $-$ **Recursive**

Theorem 9.9 HALT *is still not recursive.*

Proof: (Second proof, based on diagonalization:)

If HALT were, we recursive, we could use a decider for HALT to build a decider for $K = \{n : n \in W_n\}$. (How?)

But that would make K recursive and thus would make $\overline{K}$ r.e, contradicting the theorem above. So the HALT-decider cannot exist. ♠