Turing Machines: $M = (Q, \Sigma, \delta, s)$

$$\delta : Q \times \Sigma \rightarrow (Q \cup \{h\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$$

Def: A function f is *recursive* iff it is computed by a TM. f may be total (defined for all inputs) or partial.

Def: A set S is *recursive* or *Turing decidable* iff its characteristic function χ_S is a recursive function.

Recursive is the set of recursive sets.

Def: A set S is *recursively enumberable (r.e.)* or *Turing acceptable* iff its partial characteristic function p_S is a recursive function.

r.e. is the set of r.e. sets.

Theorem: **Recursive = r.e. $\cap$ co-r.e.**

Definition 8.1 A string $w \in \Sigma^\star$ is a *palindrome* iff it is the same as its reversal, i.e., $w = w^R$. ♠

Examples of palindromes:

- 101
- 1101001011
- ABLE WAS I ERE I SAW ELBA
- AMANAPLANACANALPANAMA

Fact 8.2 *The set of PALINDROMES (over a fixed alphabet, Σ) is context-free but not regular.*

Proposition 8.3 *The set of PALINDROMES (over a fixed alphabet, Σ) is a recursive set.*

Proof: We must construct a TM that decides PALINDROMES. From here on, we'll be describing TM's very informally. Given the input:

$\boxed{\triangleright}$	A	B	L	E	E	L	B	A	$\sqcup$

We remember the first letter, delete it, move to the last letter, and either delete it if it matches or return false if it doesn't.

We repeat this process on the undeleted part of the string and return true iff the string becomes empty by deleting its *last* letter.

♠

Fact 8.4 *Time $O(n^2)$ is necessary and sufficient for a one-tape Turing machine to accept the set, PALINDROMES.*

Proof: The procedure above used time $O(n^2)$ (do you see this?). The **lower bound** is harder to prove and depends strongly on the fact that we are working with a *one-tape* Turing machine. One way to see the lower bound is to do problems 2.8.4 and 2.8.5 from [P]. ♠

Definition 8.5 A k-tape Turing machine, $M = (Q, \Sigma, \delta, s)$

Q: finite set of states; $s \in Q$

Σ: finite set of symbols;

δ:$Q \times \Sigma^k \;\to\; (Q \cup \{h\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$

♠

A multitape TM *reads* one cell on each tape (the cell under that tape's head) on each time step. Based on those cells' contents and its state, it *writes* into the current cell on each tape and *moves* up to one cell left or right on each tape.

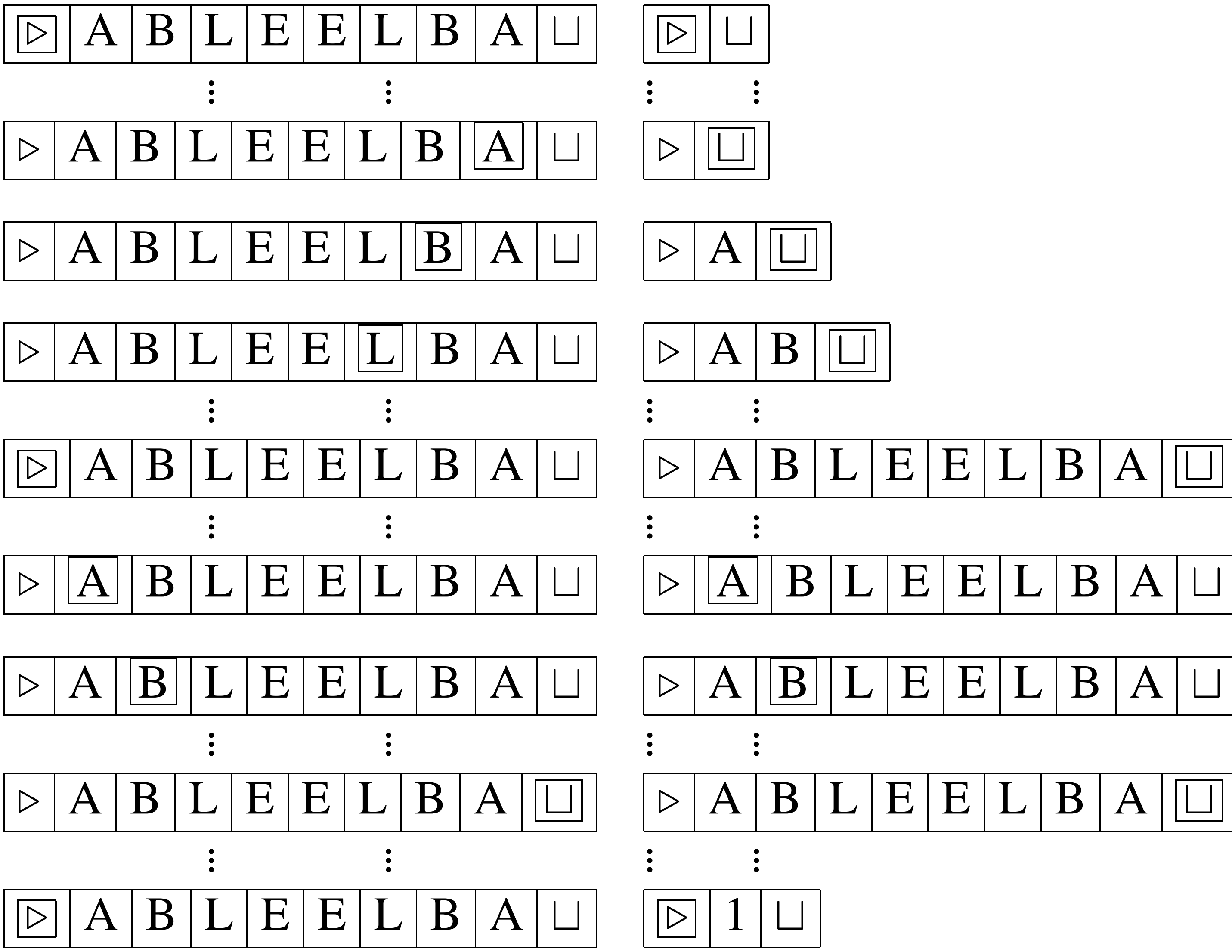
Theorem 8.6 *For any k, any k-tape TM may be simulated by a one-tape TM.*

Proof: See [P] or [S] for a detailed proof. The idea is to store the k tapes on a single tape, with the contents either in series or in parallel. To simulate one step of the k-tape machine, the one-tape machine must find the contents of the current cells and then perform the proper actions on the part of its tape corresponding to each current cell.

♠

Proposition 8.7 *PALINDROMES can be accepted in* **DTIME**$[n]$ *on a 2-tape TM.*

Proof: (that PALINDROMES $\in$ **DTIME**$[n]$)



♠

Definition 8.8 A set $A \subseteq \Sigma^\star$ is in **DTIME**$[t(n)]$ iff there exists a deterministic, multi-tape TM, M, and a constant c, such that,

1. $A = \mathcal{L}(M) \equiv \{w \in \Sigma^\star \mid M(w) = 1\}$, and
2. $\forall w \in \Sigma^\star$, $M(w)$ halts within $c(1 + t(|w|))$ steps.

$\spadesuit$

Why "$c(1+t(|(|w))$ steps"? If the input is size n, we normally want the running time bound to be $O(t(n))$ steps. But for convenience with functions t where $t(n)$ might be less than one, we modify this to $O(1+t(n))$ steps, the max of $O(1)$ and $O(t(n))$.

Definition 8.9 A set $A \subseteq \Sigma^\star$ is in **DSPACE**$[s(n)]$ iff there exists a deterministic, multi-tape TM, M, and a constant c, such that,

1. $A \quad = \quad \mathcal{L}(M), \quad$ and
2. $\forall w \in \Sigma^\star$, $M(w)$ uses at most $c(1 + s(|w|))$ work-tape cells.

(Note: The input tape is **read-only** and **not counted as space used**. Otherwise space bounds below n would rarely be useful. But in the real world we often want to limit space and work with read-only input. Consider, for example, a problem where the "input" is the entire World-Wide Web.) ♠

Example: We have just shown that PALINDROMES is in both **DTIME**$[n]$ and **DSPACE**$[n]$.

In fact, we'll show a little later that PALINDROMES $\in$ **DSPACE**$[\log n]$.

Definition 8.10 $f : \Sigma^\star \to \Sigma^\star$ is in $F(\mathbf{DTIME}[t(n)])$ iff there exists a deterministic, multi-tape TM M, and a constant c, such that:

1. $f \quad = \quad M(\cdot)$, i.e., $\forall w : f(w) = M(w)$;
2. $\forall w \in \Sigma^\star$, $M(w)$ halts within $c(1 + t(|w|))$ steps;
3. $|f(w)| \leq |w|^{O(1)}$, i.e., f is polynomially bounded.

♠

The last condition is a technical one as functions that return enormously bigger strings are convenient to rule out. Note that if $f(n)$ is itself $n^{O(1)}$, as is usual for practically feasible algorithms, the last condition follows automatically.

Definition 8.11 $f : \Sigma^\star \to \Sigma^\star$ is in $F(\mathbf{DSPACE}[s(n)])$ iff there exists a deterministic, multi-tape TM M, and a constant c, such that:

1. $f = M(\cdot)$, i.e., $\forall w : f(w) = M(w)$;
2. $\forall w \in \Sigma^\star$, $M(w)$ uses at most $c(1 + s(|w|))$ work-tape cells;
3. $|f(w)| \leq |w|^{O(1)}$, i.e., f is polynomially bounded.

(Recall that the input tape is *read-only* and that the output tape is *write-only*, and that neither is counted against the space bound.) ♠

Examples: The function Plus is in $F(\mathbf{DTIME}[n])$ and $F(\mathbf{DSPACE}(1)$. As you'll prove on the revised HW#2, the function Times is in $F(\mathbf{DTIME}[n^2])$ and (for extra credit) $F(\mathbf{DSPACE}[\log n])$.

We are now ready to define three important **complexity classes**:

$$\mathbf{L} \equiv \mathbf{DSPACE}[\log n]$$

$$\mathbf{P} \equiv \mathbf{DTIME}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \mathbf{DTIME}[n^i]$$

$$\mathbf{PSPACE} \equiv \mathbf{DSPACE}[n^{O(1)}] \equiv \bigcup_{i=1}^{\infty} \mathbf{DSPACE}[n^i]$$

Theorem 8.12 P *is also the set of languages decidable in* $n^{O(1)}$ *time on a* one-tape *TM.*

Proof: **P** clearly contains all these languages. For the other direction, we look more closely at the simulation of a k-tape TM by a one-tape TM. Simulating one step of the former requires two passes over the entire tape of the latter, which can be done in $O(s(n)) = O(t(n))$ time. So $O(t^2(n))$ time on the one-tape TM suffices to simulate $O(t(n))$ time on the k-tape TM. ♠

Theorem 8.13 *For any functions $t(n) \geq n, s(n) \geq \log n$, we have*

$$\begin{aligned}\mathbf{DTIME}[t(n)] &\subseteq \mathbf{DSPACE}[t(n)] \\ \mathbf{DSPACE}[s(n)] &\subseteq \mathbf{DTIME}[2^{O(s(n))}]\end{aligned}$$

Proof: The first statement is obvious.

To prove the second statement, let M be a **DSPACE**$[s(n)]$ TM, let $w \in \Sigma^\star$, and let $n = |w|$.

M has k tapes and when it runs on w it uses at most $cs(n)$ work-tape cells. During this computation M thus has at most

$$|Q| \cdot (n + cs(n) + 2)^k \cdot |\Sigma|^{cs(n)} \quad < \quad 2^{k's(n)}$$

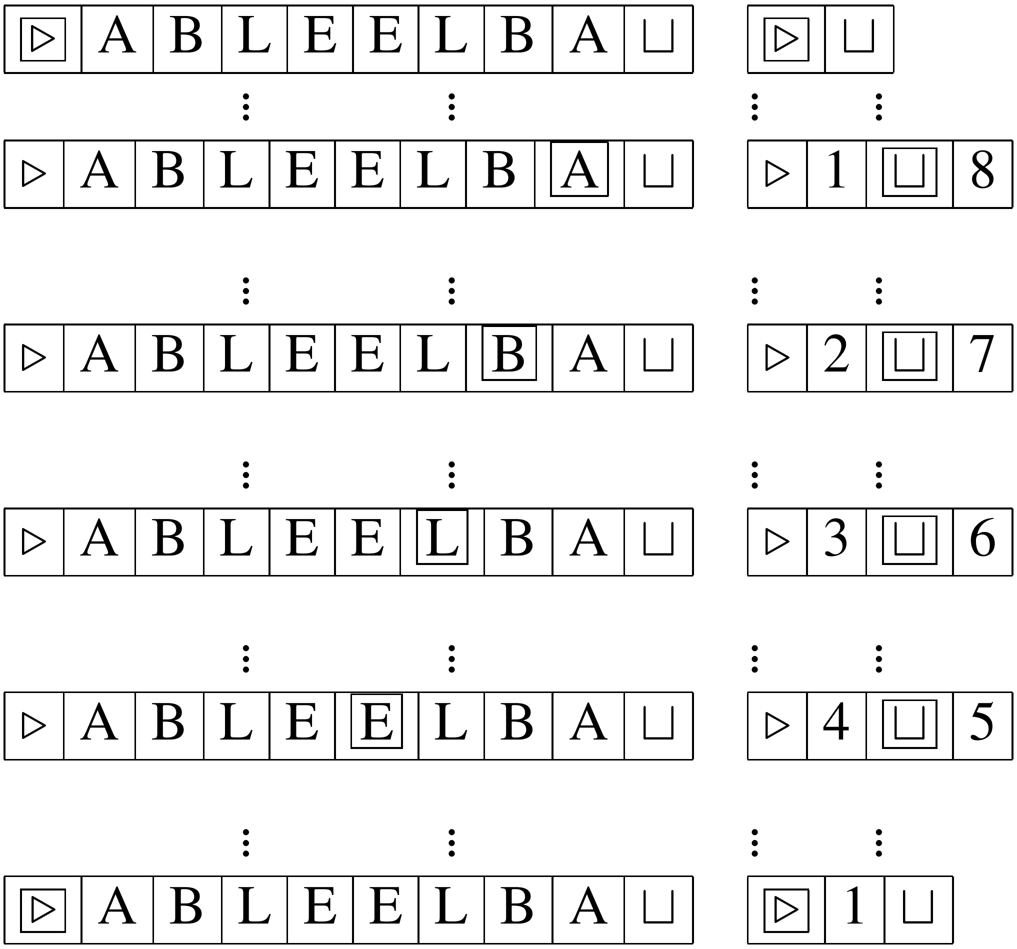
possible configurations.

Thus, after $2^{k's(n)}$ steps, $M(w)$ must be in an infinite loop.

So either it halts *before* using that many steps, or it never halts at all. And for it not to halt is a contradiction, as M is supposed to halt on all inputs. ♠

Corollary 8.14 $\mathbf{L} \subseteq \mathbf{P} \subseteq \mathbf{PSPACE}$

PALINDROMES $\in$ L



The machine must keep track of and manipulate two pointers into the input. Since each of these pointers holds a number from 1 through n, where n is the length of the input, the worktape space used for each pointer is $O(\log n)$ cells, and the total space is $O(\log n)$.

RAM = Random Access Machine

Memory:

κ	r_0	r_1	r_2	r_3	r_4	$\cdots$	r_i	$\cdots$

κ = program counter; r_0 = accumulator

Instruction	**Operand**	**Semantics**
READ	$j \mid \uparrow j \mid \ = j$	$r_0 := (r_j \mid r_{r_j} \mid j)$
STORE	$j \mid \uparrow j$	$(r_j \mid r_{r_j}) := r_0$
ADD	$j \mid \uparrow j \mid \ = j$	$r_0 := r_0 + (r_j \mid r_{r_j} \mid j)$
SUB	$j \mid \uparrow j \mid \ = j$	$r_0 := r_0 - (r_j \mid r_{r_j} \mid j)$
HALF		$r_0 := \lfloor r_0/2 \rfloor$
JUMP	j	$\kappa := j$
JPOS	j	**if** $(r_0 > 0)$ **then** $\kappa := j$
JZERO	j	**if** $(r_0 = 0)$ **then** $\kappa := j$
HALT		$\kappa := 0$

This is a reasonable model of an actual assembly-language program (an extreme version of a RISC).

Theorem 8.15

$$\mathbf{DTIME}[t(n)] \;\subseteq\; \text{RAM-TIME}[t(n)] \;\subseteq\; \mathbf{DTIME}[(t(n))^3]$$

Proof: The first inclusion is obvious. For the other, we must simulate the RAM with a TM. First, we memorize the program in the TM's finite control.

Store all registers on one tape:

$\triangleright$	1	1	,	0	:	1	0	1	,	1	0	1	:	0	,	1	0	1	1	:	1	0	$\sqcup$
	κ					r_0						r_5							r_{11}				

Store workspace for calculations on second tape:

$\triangleright$	1	0	0	,	1	0	1	1	$\sqcup$
		κ'					A		

Use the third tape for moving over sections of the first tape.

$\triangleright$	0	:	1	0	1	,	1	0	1	:	0	,	1	0	1	1	:	1	0	$\sqcup$
			r_0						r_5							r_{11}				

Each register contains at most $n + t(n)$ bits, because our instructions allow us to at most double the largest number each time.

The total number of tape cells used is at most

$$2t(n)(n + t(n)) = O((t(n))^2)$$

Each step takes at most $O((t(n))^2)$ steps to simulate. ♠

What if we added a MULT instruction to our RAM? Then we could *square* the largest number each time, generating numbers of $2^{O(t(n))}$ bits in time $t(n)$. These numbers can't be processed in polynomial time by a TM!

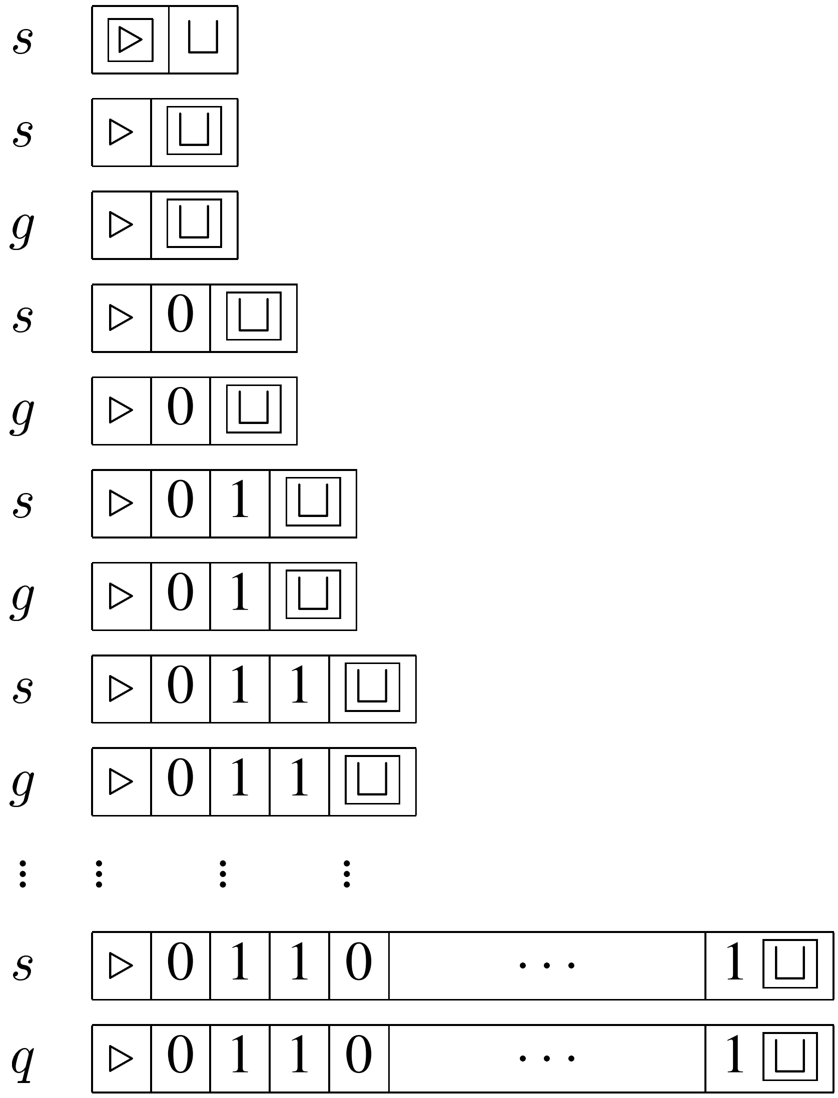
But real computers have a fixed word size rather than registers that can hold arbitrary integers. To model this, the algorithms books use a *log-cost RAM* as their basic model, so that touching a register with a number that is b bits costs you $O(b)$ time instead of $O(1)$. The distinction between ordinary (*unit-cost*) and log-cost RAMs is only important when dealing with very large numbers.

A nondeterministic Turing Machine may choose one of two possible moves on each step. Here is an example:

guess.tm	s	g	q
0			
1			
$\sqcup$	$g, \sqcup, - \mid q, \sqcup, -$	$s, 0, \rightarrow \mid s, 1, \rightarrow$	
$\triangleright$	$s, \triangleright, \rightarrow$		
comment	g or q	guess 0 or 1	the rest

- Write down an arbitrary string $g \in \{0, 1\}^\star$, the guess.
- Proceed with the rest of the computation, using g if desired.
- As with an NFA, we "accept" the input iff there exists a computation path leading to acceptance. With this machine this is true iff there exists some guess string that leads to acceptance.

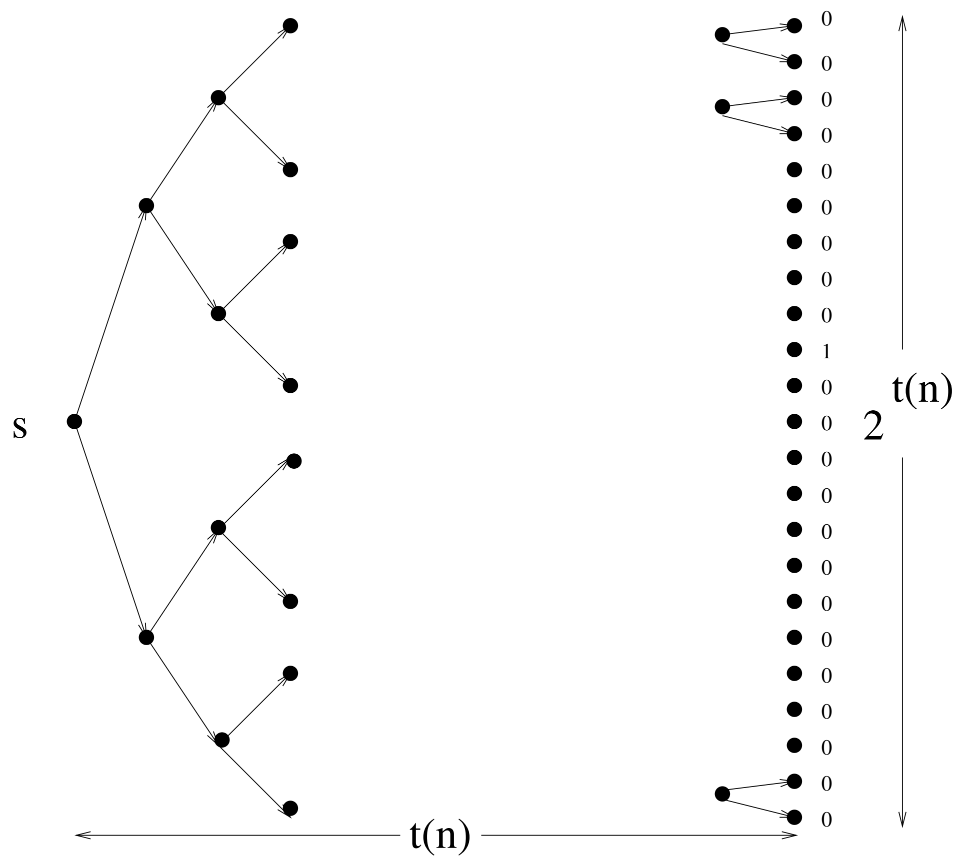
guess.tm	s	g	q
0			
1			
$\sqcup$	$g, \sqcup, - \mid q, \sqcup, -$	$s, 0, \rightarrow \mid s, 1, \rightarrow$	
$\triangleright$	$s, \triangleright, \rightarrow$		
comment	g or q	guess 0 or 1	the rest



Definition 8.16 The set accepted by a NTM, $N: \ \mathcal{L}(N) \equiv$

$$\{w \in \Sigma^\star \mid \text{some run of } N(w) \text{ halts with output "1"}\}$$

We define **the time taken** by N on $w \in \mathcal{L}(N)$ to be the number of steps in the *shortest computation* of $N(w)$ that accepts. ♠



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

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

Theorem 8.17 *For any function $t(n)$,*

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

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

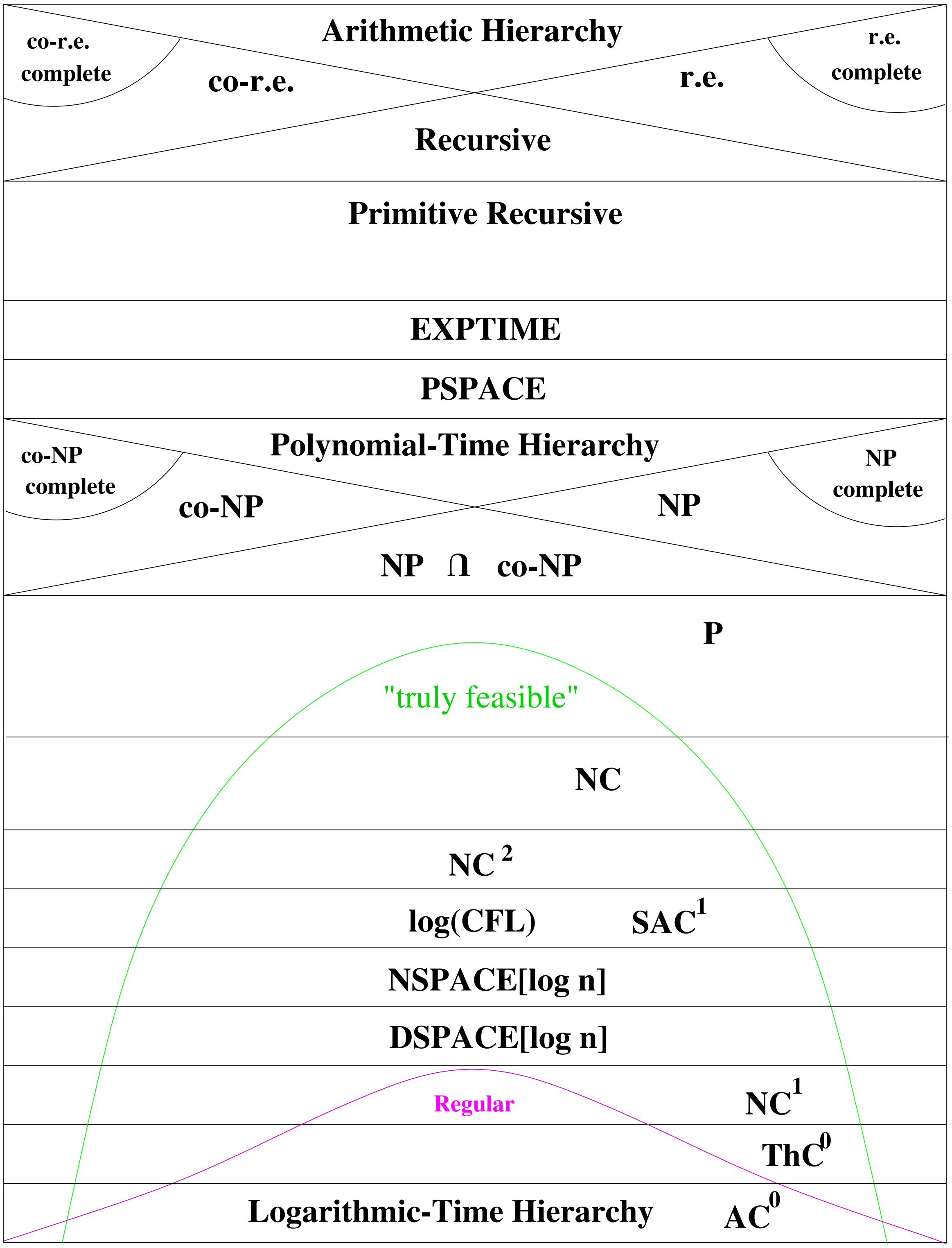
Corollary 8.18

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

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.


Arithmetic Hierarchy
co-r.e. complete
co-r.e.
r.e.
r.e. complete
Recursive
Primitive Recursive
EXPTIME
PSPACE
Polynomial-Time Hierarchy
co-NP complete
co-NP
NP
NP complete
NP ∩ co-NP
P
"truly feasible"
NC
NC 2
log(CFL)
SAC1
NSPACE[log n]
DSPACE[log n]
Regular
NC1
ThC0
Logarithmic-Time Hierarchy
AC0