§4.3. **Extensions to Turing machines**

- The extensions to Turing machines don't change the model's power
- These extensions make certain results easier

Why add multiple tapes to a Turing machine?

- It's interesting that it adds no power
- Sometimes it's easier to use
  - Theorem 4.4.2, reducing a RAM to a $(k+3)$-tape machine
  - Theorem 4.5.1, reducing a nondeterministic Turing machine to a 3-tape machine

Theorem 4.3.1. What can be done with k tapes can be done with one tape, and the one-tape machine is (no worse than) $O(t(|x| + t))$.

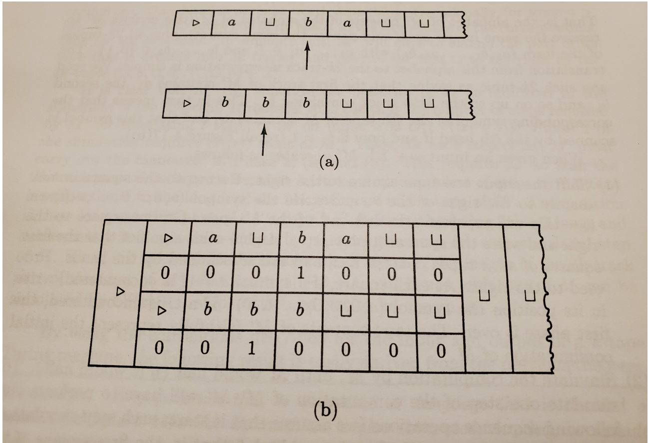


Figure 4-17

- Make the alphabet of the constructed one-tape machine to be $2k$-tuples
- For each single step in the original, there are two phases
  - Scan to plan
  - Act

§4.4. **Random access Turing machines (RATMs or RAMs)**

Definition 4.4.1: A RATM is a pair $M = (k, \Pi)$, where k is the number of registers and Π is a list of instructions.

A **configuration** is a $k+2$ tuple, $(\kappa, R_0, R_1, \ldots R_{k-1}, T)$. Note that κ, the program counter, is the Greek kappa.

Theorem 4.4.1: Any recursive or recursively enumerable language, and any recursive function, can be decided, semidecided, and computed, respectively, by a random access Turing machine.

Theorem 4.4.2: Any language decided or semidecided by a random access Turing machine, and any function computable by a random access Turing machine, can be decided, semidecided, and computed, respectively, by a standard Turing machine in polynomial steps.

Theorem 4.4.2: Any language decided or semidecided by a random access Turing machine, and any function computable by a random access Turing machine, can be decided, semidecided, and computed, respectively, by a standard Turing machine in polynomial steps.

Proof sketch.

- *One tape for input*
- *One tape for the store*
- k *tapes for registers*
- *One tape as "scratch space"*
- *One tape to rule them all, one tape to find them, one tape to bring them all and in the darkness bind them.*
  In the land of Mordor, where the Turing machine lies.

§4.5. **Nondeterministic Turing machines**

Three ways to think of nondeterminism: Oracular knowledge, Searching with back-tracking, and bifurcation.

Decision and semidecision for nondeterministic Turing machines

Decide	For all computations	the machine *must* halt (For w there exists a finite bound N on the length of any computation)
	For all $w \in L$	there exists a computation that halts y (Some may halt n)
	For all $w \notin L$	*no* computations halt y (*All* computations halt n)
Semidecide	Some computations may not halt	
	For all $w \in L$	there exists a computation that halts
	For all $w \notin L$	*no* computations halt

Prob 4.5.1.a

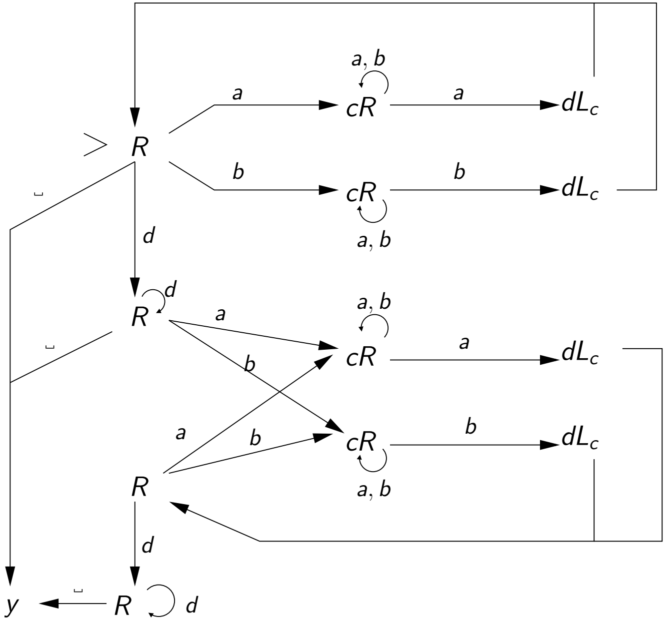
Consider the regular expression to be broken up in the following phases:

$$\underbrace{a*}_{1}\underbrace{a}_{2}\underbrace{b}_{3}\underbrace{b*}_{4}\underbrace{b}_{5}\underbrace{a}_{6}\underbrace{a*}_{7}$$

Consider those phases states. Then we can make the Turing machine as

$$\begin{array}{cc}
(1, a, 1, \rightarrow) & (1, a, 2, \rightarrow) \\
(2, a, 3, \rightarrow) & (3, b, 4, \rightarrow) \\
(4, b, 4, \rightarrow) & (4, b, 5, \rightarrow) \\
(5, b, 6, \rightarrow) & (6, a, 7, \rightarrow) \\
(7, a, 7, \rightarrow) & (7, \sqcup, h,)
\end{array}$$

Prob 4.5.1.b



Theorem 4.5.1: If a nondeterministic Turing machine M semidecides or decides a language, or computes a function, then there is a standard Turing machine M' that semidecides or decides the language or computes the function, respectively.

Proof sketch. *(Main idea: simulate all computations until you get a halt, if ever.)*

At a given step, there are a fine number of steps the machine can make next. Suppose the configuration is $(q, u\underline{a}v)$. Then the next step considers only q and a*, but is drawn from $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$, of which there are at most $|K|(|\Sigma| + 2)$ (call this r) possibilities.*

In the worst case, each state/symbol pair has exactly r possibilities, of which we arbitrarily pick one.

Define machine M' with three tapes: the original input, the simulated M input, and the hint tape.

Algorithm: *Copy input onto the simulated tape*
Put 1 onto the hint tape
L: *Operate like M_d*
If you ever halt, then great!
If you run out of hints, then
  Copy the original input back to the simulated tape
  Put the "lexicographically next" hint on the hint tape
  Go to L