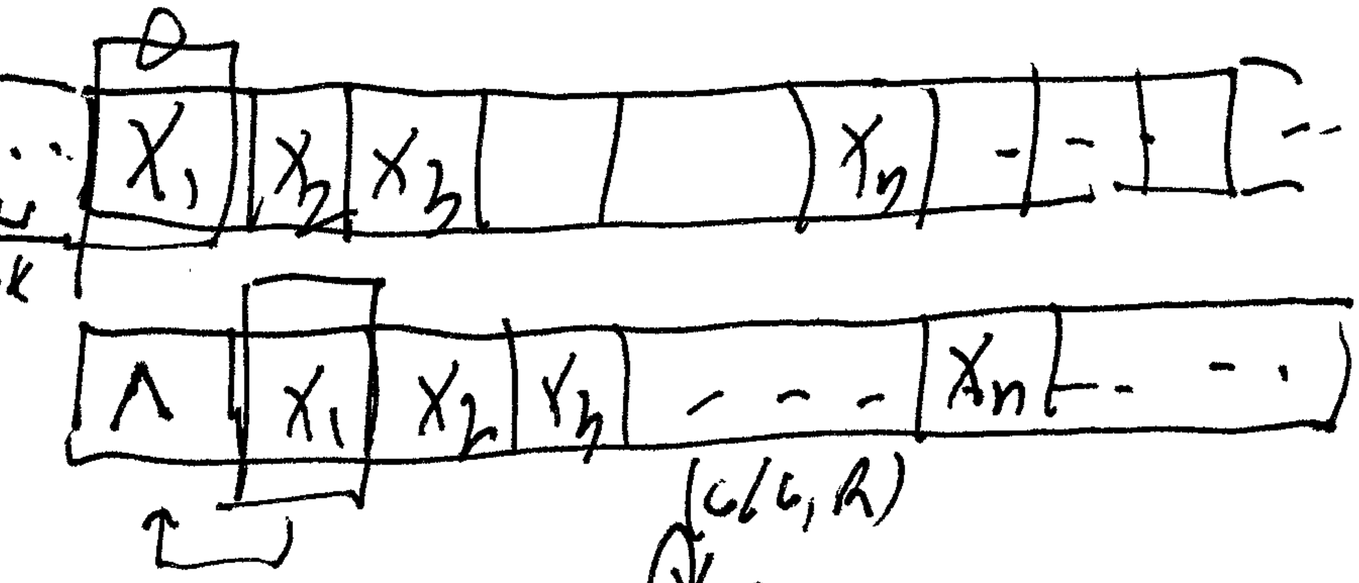
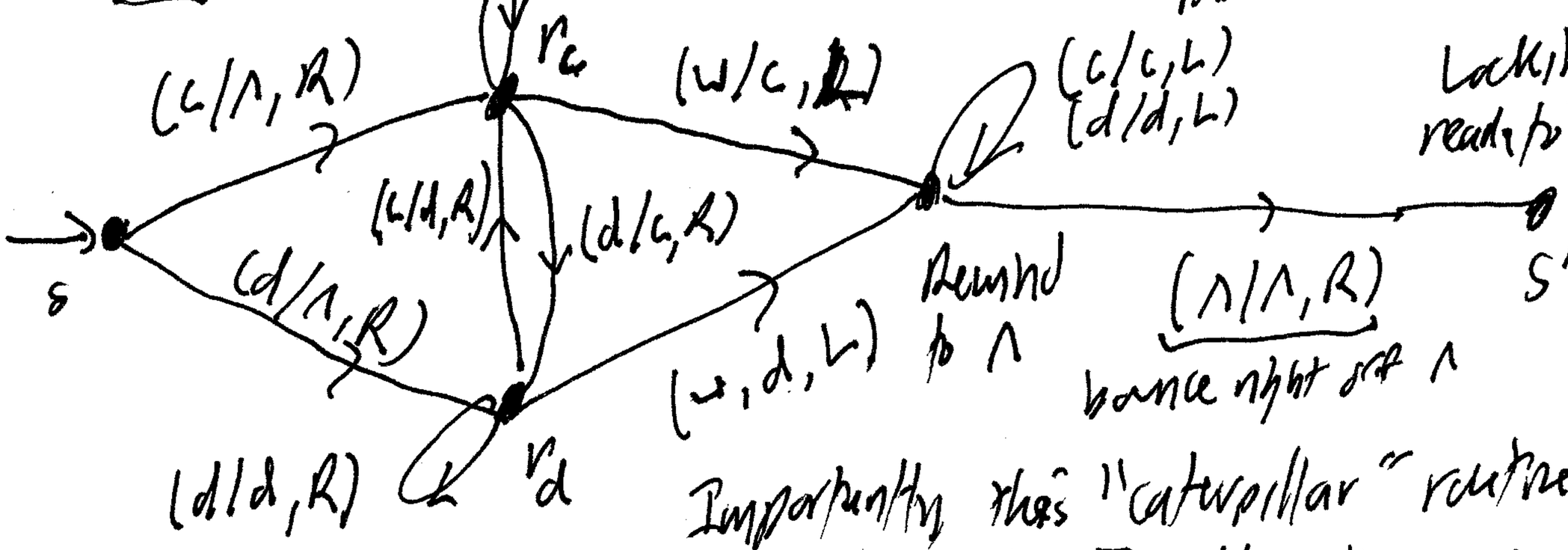
Lecture Thu 4/16 General Capabilities of Turing Machines



Suppose tape has a hard left-end. We want to make a "caterpillaring" the tape.

Suppose $\Sigma = \{c, d\}$



Locking at X_1, ready to "start again"

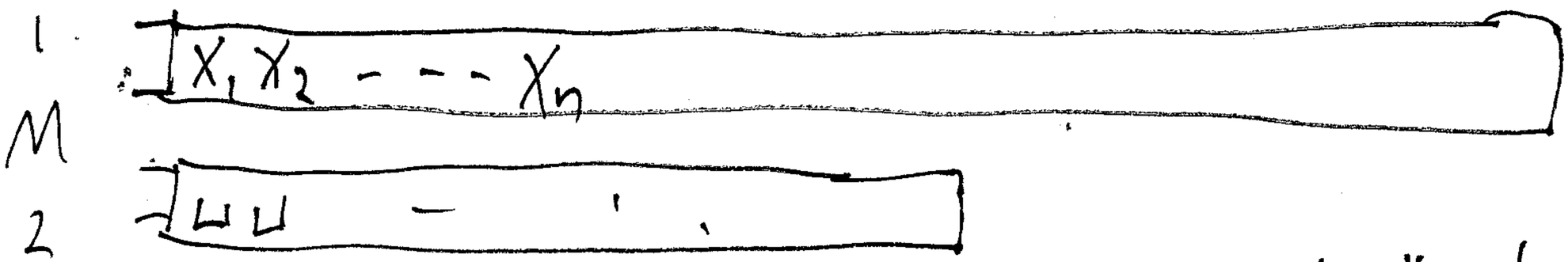
Importantly this "caterpillar" routine - extended to all chars in Γ other than Λ (or #, say) can be used as a "daemon" to make more room.

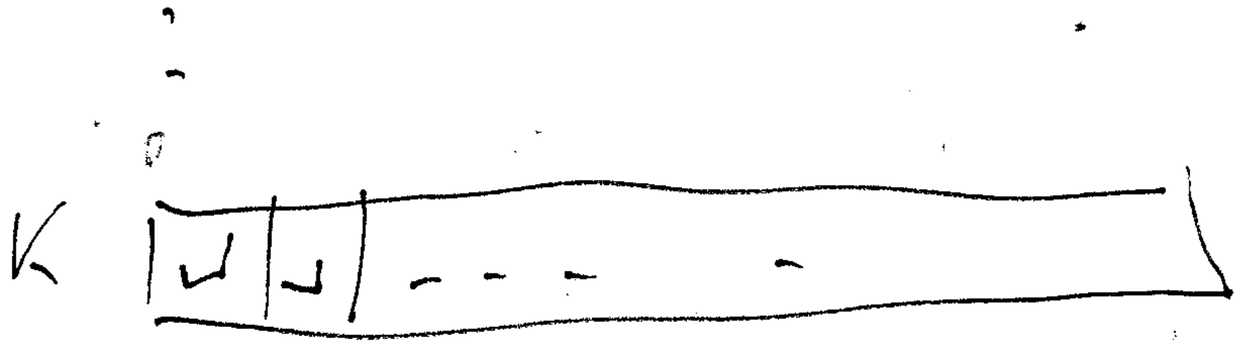
Theorem (alluded to but not formally stated in text): For every TM M with a 2-way infinite tape, we can build a TM M' that has only a one-way infinite tape that simulates M step-for-step. (standard in text? Not?)

> Key: M' sees Λ exactly when M moves into a previously unvisited blank cell on the left.

M' then overlays, at every state r of M, a modified caterpillar routine that is triggered by Λ and inserts ␣.

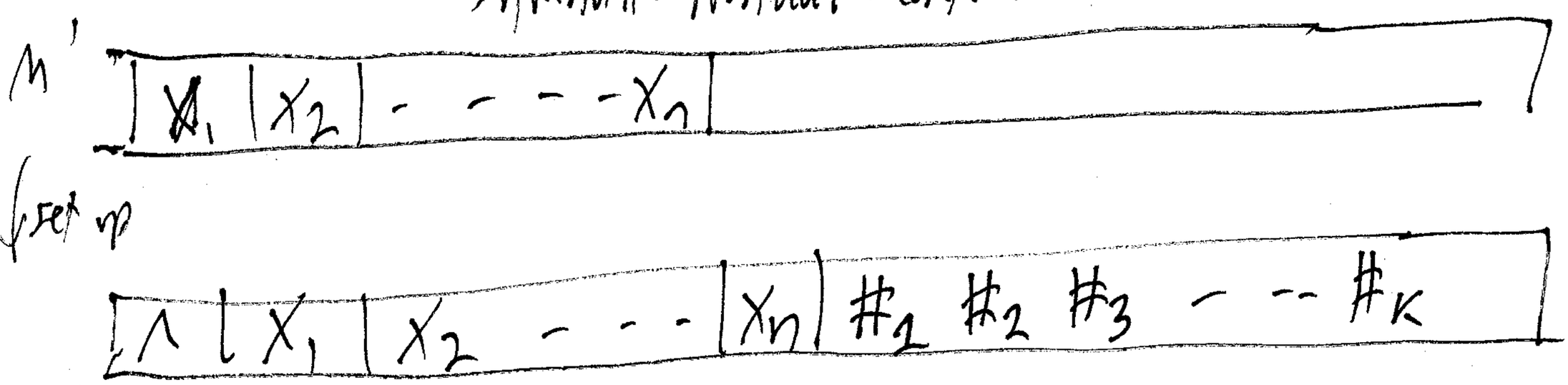
Theorem (in §3.2): Given any k-tape TM M, we can build a 5-style-tape TM M' that simulates M step-for-5 steps.



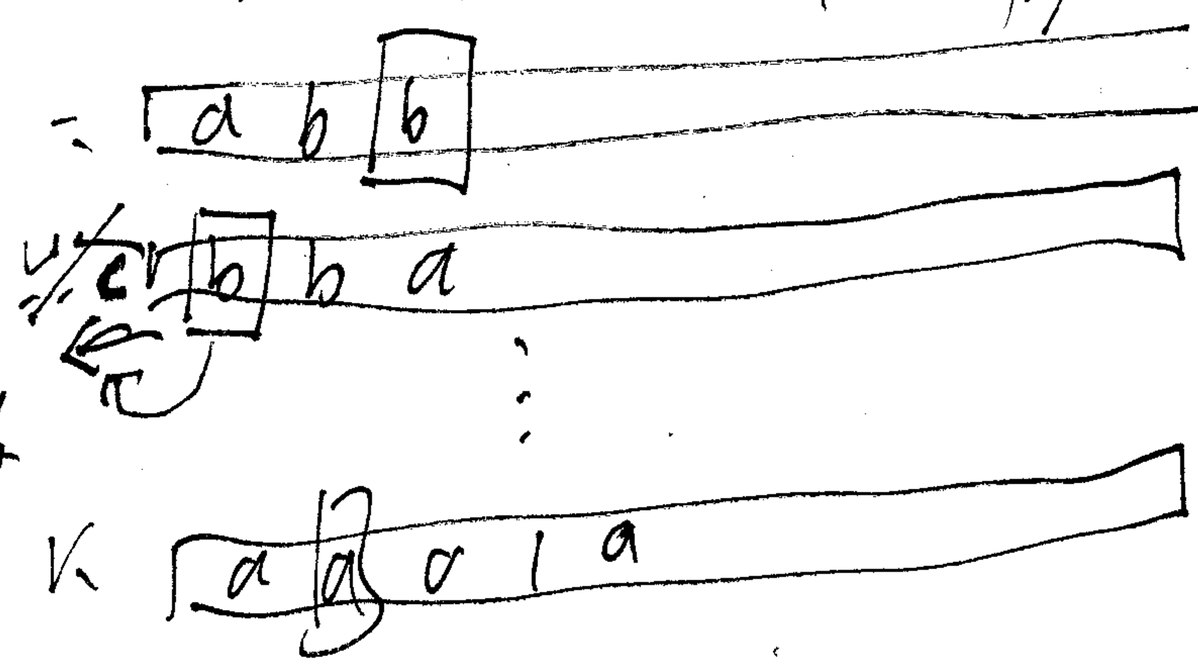


M' allocates a new # char not used by M. If needed or helpful, we can have M' allocate k-many # chars $\#_1, \#_2 \cdots \#_k$ instead.

Invariant: Nonblank contents of Tape j are between $\#_{j-1}$ and $\#_j$.

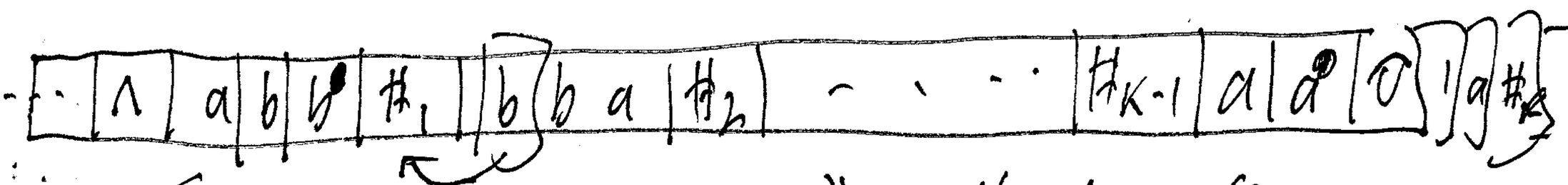


— Now consider any configuration of a computation by M



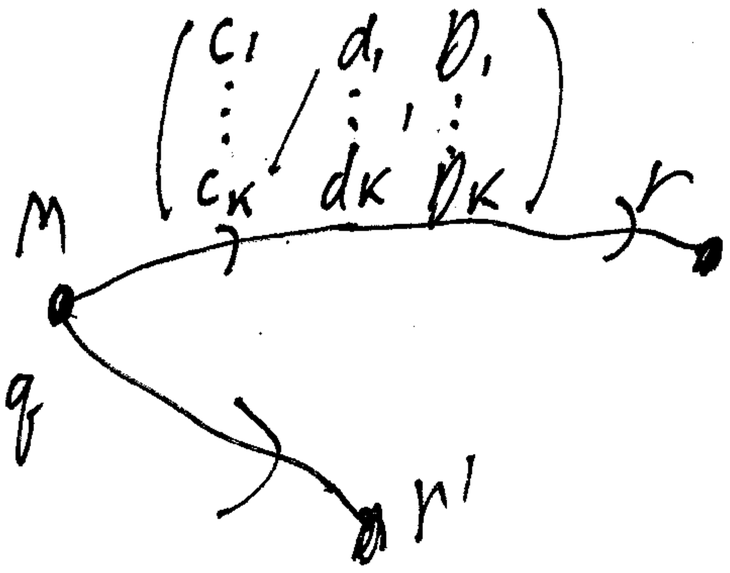
M' also has a "dot alias" $c^\bullet$ for every char c used by M. to mark head locations on other tapes.

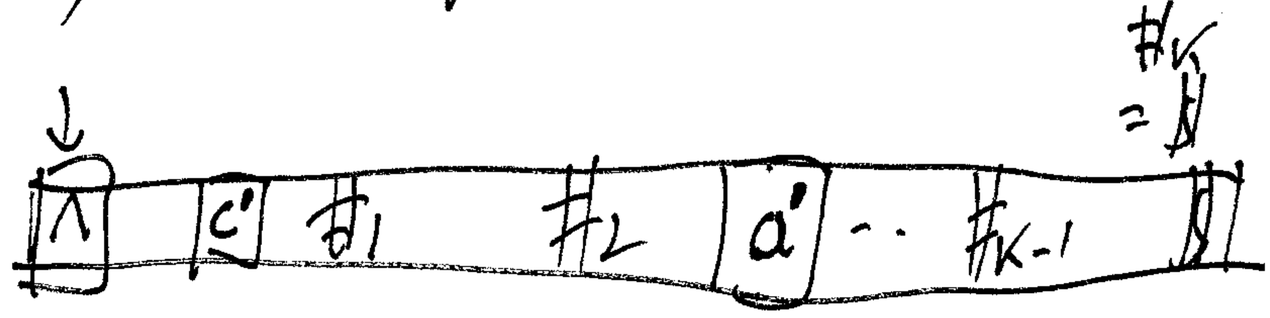
By our INVariant, M' has



$\#_1$, Activate leftward "caterpillar daemon".

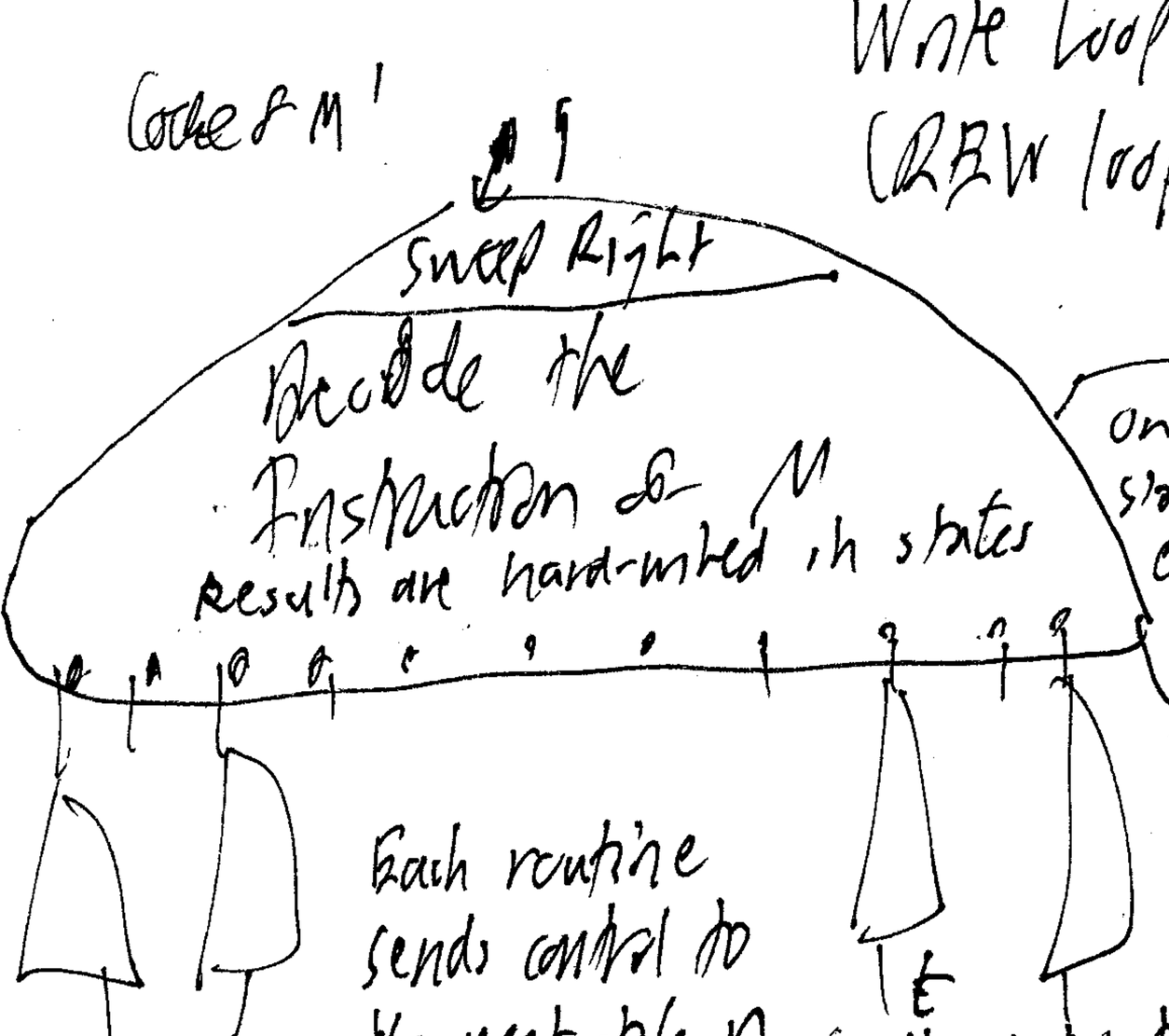
- "Caterpillar" solves the problem of maintaining virtual tapes.
- M' has only one head, but M' gets K inputs at once for its steps





M' does a Read-Evaluate-Write Loop (RBW loop)

- Read and remember K-many dotted chars, in L to R order.



- Enter δ_M [⊔]

to find the corresponding instruction $(q, \begin{matrix} c_1 \\ \vdots \\ c_K \end{matrix}, \begin{matrix} d_1 \\ \vdots \\ d_K \end{matrix}, \begin{matrix} D_1 \\ \vdots \\ D_K \end{matrix})$ and remember the $d_1 \cdots d_K$ and directions $D_1 \cdots D_K$ that you need to execute on each tape.

- Enter each R-to-L sweep in execute phase, writing $d_1 \cdots d_K$

In this way, M' simulates M, and in particular, enters q_{acc} iff M does.

☒ $\therefore L(M') = L(M)$.

Other things TMs can do:

- Multiply numbers by repeated addition
- copy strings — shifting them if needed.
- compare strings for $=$, also for $\leq$
- add/subtract (binary) numbers, also compare $\leq$
- Maintain virtual registers on the tape or tapes.

☆ Turing Machines can be a Target Machine Language for any Compiler.
1. Compile to a (type-free) assembly language. 2- Simulate assembly by TM.

Significance: Turing Machines are a General Universal Model.

Defn: A language $L \subseteq \Sigma^*$ is { (Turing-) acceptable / (Turing-) recognizable / computable enumerable } if there is a TM M s.t. $L = L(M)$.

If in addition M halts for all inputs, then L is decidable.

Example: $\{a^n b^n c^n : n \geq 0\}$ is decidable but not a CFL.
Because a DFA or a DPDA "Is-A" TM that always halts,
all regular languages, indeed all CFLs, are decidable too.

☆ These definitions have the same effect for programs in any other (known) High-Level Language in place of Turing Machines.

Church-Turing Thesis: This will remain true for any model we can build, or any formulation of human or alien decision making.

Footnote: From now on, given any clear procedure-in pseudo code related to a high-level language or just in clear steps - we can assert that a Turing Machine can do it. This is how "appeals to the C-T Thesis" work in practice.
For a simplifying example, can you imagine precisely how to simulate a Nondeterministic TM N by a high-level program that exhaustively tries all options available to N? Well, by C-T Thesis that program can ipso-facto be converted into a Deterministic TM M such that $L(M) = L(N)$.