91.304 Foundations of (Theoretical) Computer Science

Chapter 3 Lecture Notes (Section 3.2: Variants of Turing Machines)

David Martin
dm@cs.uml.edu

With some modifications by Prof. Karen Daniels, Fall 2012

Variants of Turing Machines

- *Robustness*: Invariance under certain changes
- What kinds of changes?
  - Stay put!
  - Multiple tapes
  - Nondeterminism
  - Enumerators
- (Abbreviate Turing Machine by TM.)

Stay Put!

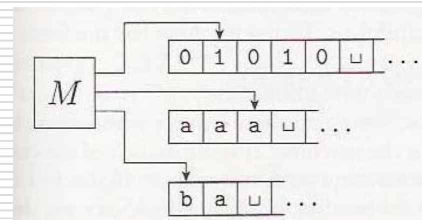
- Transition function of the form:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$$

- Does this really provide additional computational power?
- No! Can convert TM with "stay put" feature to one without it. How?
- Theme: Show 2 models are equivalent by showing they can *simulate each other*.

Multi-Tape Turing Machines

- Ordinary TM with several tapes.
  - Each tape has its own head for reading and writing.
    - Initially the input is on tape 1, with the other tapes blank.
- Transition function of the form:



$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$

  - (k = number of tapes)

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{L}, \mathrm{R}, \ldots \mathrm{L})$$

  - When TM is in state q_i and heads 1 through k are reading symbols a_1 through a_k, TM goes to state q_j, writes symbols b_1 through b_k, and moves associated tape heads L, R, or S.

Note: k tapes (each with own alphabet) but only 1 common set of states!

Source: Sipser textbook

Multi-Tape Turing Machines

- Multi-tape Turing machines are of equal computational power with ordinary Turing machines!
  - Corollary 3.15: A language is Turing-recognizable if and only if some multi-tape Turing machine recognizes it.
    - One direction is easy (how?)
    - The other direction takes more thought…
      - Theorem 3.13: Every multi-tape Turing machine has an equivalent single-tape Turing machine.
      - Proof idea: see next slide…

Source: Sipser textbook

Theorem 3.13: Simulating Multi-Tape Turing Machine with Single Tape

- Proof Ideas:
  - Simulate k-tape TM M's operation using single-tape TM S.
  - Create "virtual" tapes and heads.
    - \# is a delimiter separating contents of one tape from another tape's contents.
    - "Dotted" symbols represent head positions
      - add to tape alphabets.

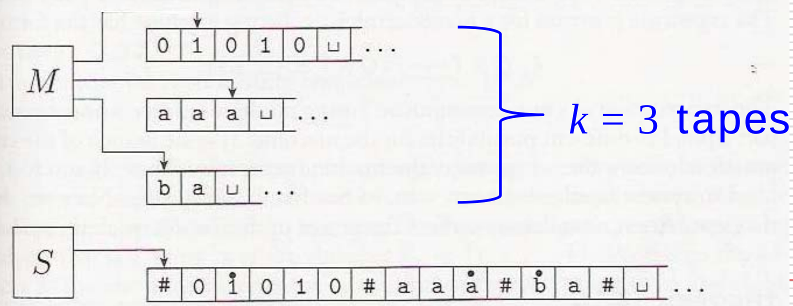


FIGURE 3.14
Representing three tapes with one

Source: Sipser textbook

<u>Theorem 3.13</u>: Simulating Multi-Tape Turing Machine with Single Tape (cont.)

- Processing input: $w = w_1 \cdots w_n$
  - Format S's tape (different blank symbol v for presentation purposes):

$$\# \dot{w}_1 w_2 \cdots w_n \# \dot{\vee} \# \dot{\vee} \# \cdots \#$$

  - Simulate single move:
    - Scan rightwards to find symbols under virtual heads.
    - Update tapes according to M's transition function.
  - Caveat: hitting right end (#) of a virtual tape:
    - rightward shift of S's tape by 1 unit and insert blank, then continue simulation

Why?

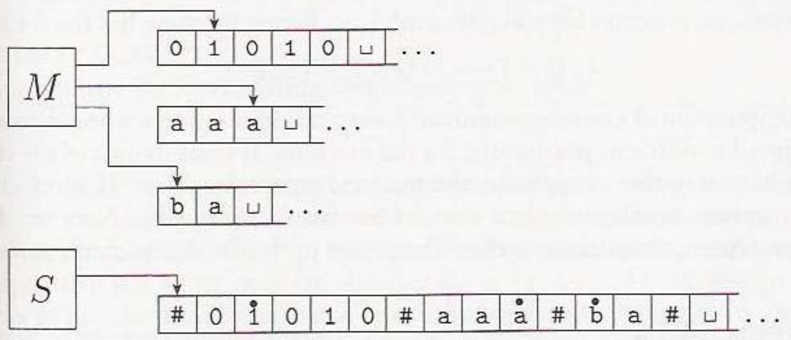


FIGURE 3.14
Representing three tapes with one

Source: Sipser textbook

Nondeterministic Turing Machines

- Transition function: $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
- Computation is a tree whose branches correspond to different possibilities. Example: board work
  - If some branch leads to an accept state, machine accepts.
- Nondeterminism does not affect power of Turing machine!
- <u>**Theorem 3.16**</u>: Every nondeterministic Turing machine (N) has an equivalent deterministic Turing machine (D).
  - <u>Proof Idea:</u> Simulate, simulate!

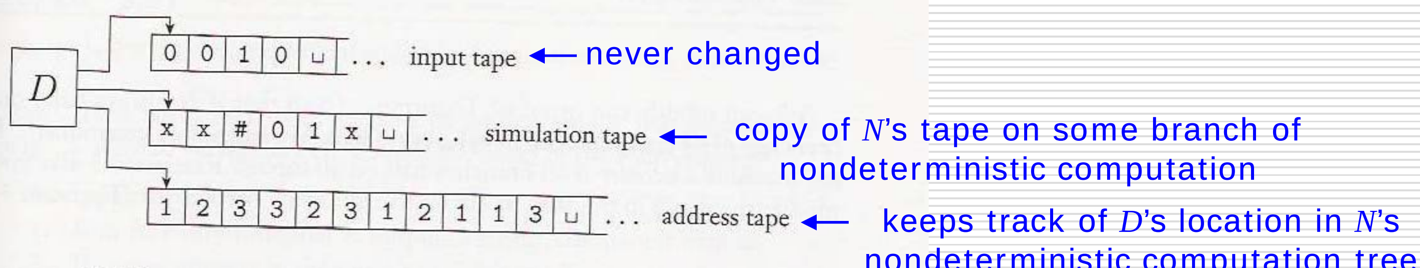


FIGURE 3.17
Deterministic TM D simulating nondeterministic TM N

Source: Sipser textbook

Theorem 3.16 Proof (cont.)

- Proof Idea (continued)
  - View *N*'s computation on input as a tree.
    - Each node is a configuration.
    - Search for an accepting configuration.
    - Important caveat: searching order matters
      - DFS vs. BFS (which is better and why?)
    - Encoding location on address tape:
      - Assume fan-out is at most *b* (what does this correspond to?)
      - Each node has address that is a string over alphabet: Σ_b = {1... *b*}

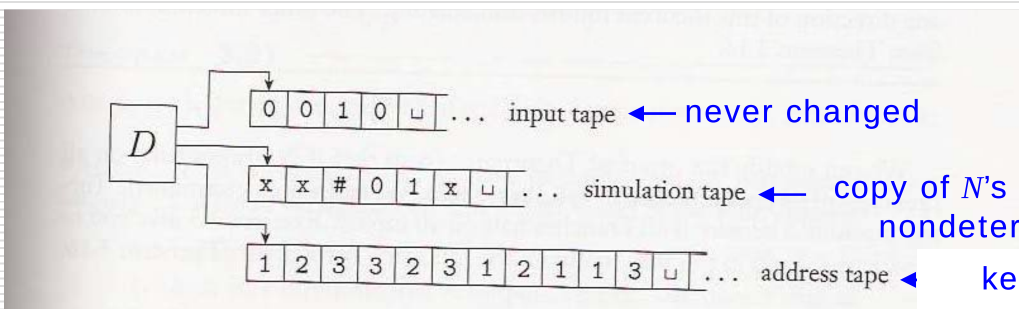


FIGURE 3.17
Deterministic TM *D* simulating nondeterministic TM *N*

never changed

copy of *N*'s tape on some branch of nondeterministic computation

keeps track of *D*'s location in *N*'s nondeterministic computation tree

Source: Sipser textbook

Theorem 3.16 Proof (cont.)

- Operation of deterministic TM D:
  1. Put input w onto tape 1. Tapes 2 and 3 are empty.
  2. Copy tape 1 to tape 2.
  3. Use tape 2 to simulate N with input w on one branch.
     1. Before each step of N, consult tape 3 (why?)
  4. Replace string on tape 3 with lexicographically next string. Simulate next branch of N's computation by going back to step 2.

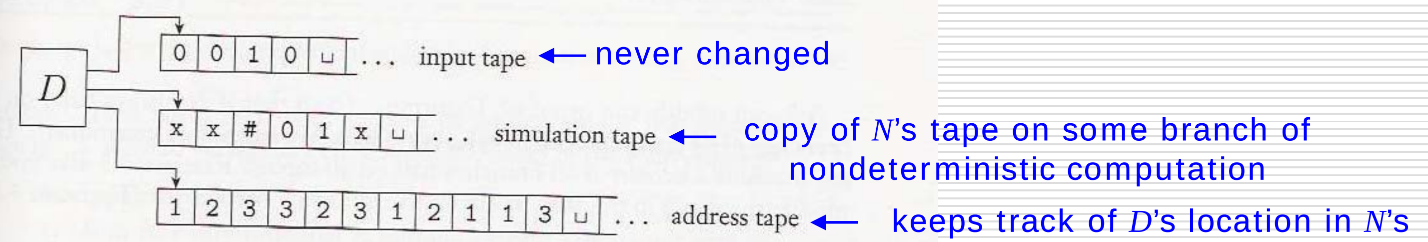


FIGURE 3.17
Deterministic TM D simulating nondeterministic TM N

Source: Sipser textbook

Consequences of Theorem 3.16

- Corollary 3.18:
  - A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.
    - Proof Idea:
      - One direction is easy (how?)
      - Other direction comes from Theorem 3.16.
- Corollary 3.19:
  - A language is decidable if and only if some nondeterministic Turing machine decides it.
    - Proof Idea:
      - Modify proof of Theorem 3.16 (how?)

Another model

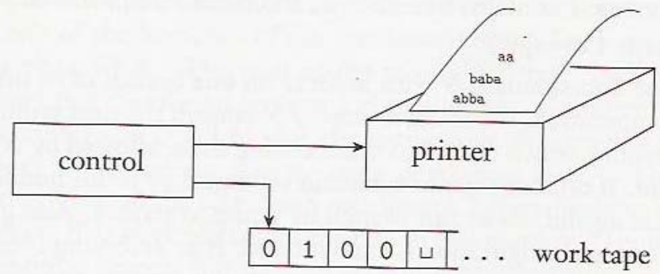


FIGURE 3.20
Schematic of an enumerator

- **Definition** An **enumerator** E is a 2-tape TM with a special state named $\mathbf{q_p}$ ("print") (2nd tape is for printing)
  - The language generated by E is (tape 1) (tape 2)
    $$L(E) = \{ x \in \Sigma^* \mid (q_0 \sqcup, q_0 \sqcup) \vdash^* (u\, q_p\, v, \mathbf{x}\, q_p\, z) \text{ for some } u, v, z \in \Gamma^* \}$$
    - Here the instantaneous description is split into two parts (tape1, tape2)
    - So this says that "x appears to the left of the tape 2 head when E enters the q_p state"
    - Note that E *always* starts with a blank tape and potentially runs forever
    - Basically, E generates the language consisting of all the strings it decides to print
    - And it doesn't matter what's on tape 1 when E prints

Source: Sipser textbook

Theorem 3.21

$L \in \Sigma_1 \Leftrightarrow L=L(E)$ for some enumerator E (in other words, enumerators are equivalent to TMs) (Recall Σ_1 is set of Turing-recognizable languages.)

Proof First we show that $L=L(E) \Rightarrow L\in\Sigma_1$. So assume that L=L(E); we need to produce a TM M such that L=L(M). We define M as a 3-tape TM that works like this:

1. input w (on tape #1)
2. run E on M's tapes #2 and #3
3. whenever E prints out a string x, compare x to w;
   if they are equal, then **accept**
   else goto 2 and continue running E

So, M accepts input strings (via input w) that appear on E's list.

Theorem 3.21 continued

Now we show that $L \in \Sigma_1 \Rightarrow L = L(E)$ for some enumerator E. So assume that $L = L(M)$ for some TM M; we need to produce an enumerator E such that $L = L(E)$. First let s_1, s_2, $\cdots$ be the lexicographical enumeration of Σ^* (strings over M's alphabet). E behaves as follows:

1. for $i := 1$ to ∞
   2. run M on input s_i
   3. if M accepts s_i then *print* string s_i (else continue with next i)

DOES NOT WORK!! WHY??

Theorem 3.21 continued

Now we show that $L \in \Sigma_1 \Rightarrow L = L(E)$ for some enumerator E. So assume that $L=L(M)$ for some TM M; we need to produce an enumerator E such that $L=L(E)$. First let $s_1, s_2, \cdots$ be the lexicographical enumeration of Σ^*. E behaves as follows:

1. for t:=1 to ∞ /* t = time to allow */
   2. for j:=1 to t /* *continue* resumes here */
      3. compute the instantaneous description uqv in M such that $q_0\, s_j \vdash^t uqv$. (If M halts *before* t steps, then *continue*)
      4. if $q = q_{acc}$ then *print* string s_j (else *continue*)

exactly t steps of the $\vdash$ relation

Theorem 3.21 continued

- First, E never prints out a string s_j that is not accepted by M
- Suppose that $q_0\ s_5 \vdash^{27} u\ q_{acc}\ v$ (in other words, M accepts s_5 after exactly 27 steps)
  - Then E prints out s_5 in iteration t=27, j=5
- Since every string s_j that is accepted by M is accepted in some number of steps t_j, E will print out s_j in iteration $t=t_j$ and in no other iteration
  - This is a slightly different construction than the textbook, which prints out each accepted string s_j infinitely many times

Summary

- *Remarkably*, the presented variants of the Turing machine model are all equivalent in power!
  - Essential feature:
    - Unrestricted access to unlimited memory
    - More powerful than DFA, NFA, PDA…
    - Caveat: satisfy "reasonable requirements"
      - e.g. perform only *finite* work in a single step.