Caveat lector: This is the zeroth (draft) edition of this lecture note. In particular, some topics still need to be written. Please send bug reports and suggestions to jeffe@illinois.edu.

If first you don't succeed, then try and try again.
And if you don't succeed again, just try and try and try.

— Marc Blitzstein, "Useless Song", *The Three Penny Opera* (1954)
Adaptation of Bertold Brecht, "Das Lied von der Unzulänglichkeit menschlichen Strebens" *Die Dreigroschenoper* (1928)

Children need encouragement.
If a kid gets an answer right, tell him it was a lucky guess.
That way he develops a good, lucky feeling.

— Jack Handey, "Deep Thoughts", *Saturday Night Live* (March 21, 1992)

38 Nondeterministic Turing Machines

38.1 Definitions

In his seminal 1936 paper, Turing also defined an extension of his "automatic machines" that he called ***choice machines***, which are now more commonly known as ***nondeterministic Turing machines***. The execution of a nondeterministic Turing machine is *not determined* entirely by its input and its transition function; rather, at each step of its execution, the machine can *choose* from a set of possible transitions. The distinction between deterministic and nondeterministic Turing machines exactly parallels the distinction between deterministic and nondeterministic finite-state automata.

Formally, a nondeterministic Turing machine has all the components of a standard deterministic Turing machine—a finite tape alphabet Γ that contains the input alphabet Σ and a blank symbol $\square$; a finite set Q of internal states with special start, accept, and reject states; and a transition function δ. However, the transition function now has the signature

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{-1,+1\}}.$$

That is, for each state p and tape symbol a, the output $\delta(p, a)$ of the transition function is a *set* of triples of the form $(q, b, \Delta) \in Q \times \Gamma \times \{-1, +1\}$. Whenever the machine finds itself in state p reading symbol a, the machine *chooses* an arbitrary triple $(q, b, \Delta) \in \delta(p, a)$, and then changes its state to q, writes b to the tape, and moves the head by Δ. If the set $\delta(p, a)$ is empty, the machine moves to the reject state and halts.

The set of all possible transition sequences of a nondeterministic Turing machine N on a given input string w define a rooted tree, called a ***computation tree***. The initial configuration $(\text{start}, w, 0)$ is the root of the computation tree, and the children of any configuration (q, x, i) are the configurations that can be reached from (q, x, i) in one transition. In particular, any configuration whose state is accept or reject is a leaf. For deterministic Turing machines, this computation tree is just a single path, since there is at most one valid transition from every configuration.

38.2 Acceptance and Rejection

Unlike deterministic Turing machines, there is a fundamental asymmetry between the acceptance and rejection criteria for nondeterministic Turing machines. Let N be any nondeterministic Turing machine, and let w be any string.

- ***N accepts w*** if and only if there is *at least one* sequence of valid transitions from the initial configuration (start, w, 0) that leads to the accept state. Equivalently, N accepts w if the computation tree contains at least one accept leaf.
- ***N rejects w*** if and only if *every* sequence of valid transitions from the initial configuration (start, w, 0) leads to the reject state. Equivalently, N rejects w if every path through the computation tree ends with a reject leaf.

In particular, N can accept w even when there are choices that allow the machine to run forever, but rejection requires N to halt after only a finite number of transitions, no matter what choices it makes along the way. Just as for deterministic Turing machines, it is possible that N neither accepts nor rejects w.

Acceptance and rejection of *languages* are defined exactly as they are for deterministic machines. A non-deterministic Turing machine N ***accepts*** a language $L \subseteq \Sigma^*$ if M accepts all strings in L and nothing else; N ***rejects*** L if M rejects every string in L and nothing else; and finally, N ***decides*** L if M accepts L and rejects $\Sigma^* \setminus L$.

38.3 Time and Space Complexity

★★★

- Define "time" and "space".
- TIME($f(n)$) is the class of languages that can be decided by a deterministic multi-tape Turing machine in $O(f(n))$ time.
- NTIME($f(n)$) is the class of languages that can be decided by a nondeterministic multi-tape Turing machine in $O(f(n))$ time.
- SPACE($f(n)$) is the class of languages that can be decided by deterministic multi-tape Turing machine in $O(f(n))$ space.
- NSPACE($f(n)$) is the class of languages that can be decided by a nondeterministic multi-tape Turing machine in $O(f(n))$ space.
- Why multi-tape TMs? Because t steps on any k-tape Turing machine can be simulated in $O(t \log t)$ steps on a two-tape machine [Hennie and Stearns 1966, essentially using lazy counters and amortization], and in $O(t^2)$ steps on a single-tape machine [Hartmanis and Stearns 1965; realign multiple tracks at every simulation step]. Moreover, the latter quadratic bound is tight [Hennie 1965 (palindromes, via communication complexity)].

38.4 Deterministic Simulation

Theorem 1. *For any nondeterministic Turing machine N, there is a deterministic Turing machine M that accepts exactly the same strings and N and rejects exactly the same strings as N. Moreover, if every computation path of N on input x halts after at most t steps, then M halts on input x after at most $O(t^2 r^{2t})$ steps, where r is the maximum size of any transition set in N.*

Proof: I'll describe a deterministic machine M that performs a breadth-first search of the computation tree of N. (The depth-first search performed by a standard recursive backtracking algorithm won't work here. If N's computation tree contains an infinite path, a depth-first search would get stuck in that path without exploring the rest of the tree.)

At the beginning of each simulation round, M's tape contains a string of the form

$$\square\square\cdots\square\bullet\bullet y_1 q_1 z_1 \bullet y_2 q_2 z_2 \bullet \cdots \bullet y_k q_k z_k \bullet\bullet$$

where each substring $y_i q_i z_i$ encodes a configuration $(q_i, y_i z_i, |y_i|)$ of some computation path of N, and $\bullet$ is a new symbol not in the tape alphabet of N. The machine M interprets this sequence of encoded configurations as a queue, with new configurations inserted on the right and old configurations removed

from the left. The double-separators $\bullet\bullet$ uniquely identify the start and end of this queue; outside this queue, the tape is entirely blank.

Specifically, in each round, first M appends the encodings of all configurations than N can reach in one transition from the first encoded configuration $(q_1, y_1 z_1, |y_1|)$; then M erases the first encoded configuration.

$$\cdots\square\square\bullet\bullet\underline{y_1 q_1 z_1}\bullet y_2 q_2 z_2 \bullet \cdots \bullet y_r q_r z_r \bullet\bullet\underline{\square\square\cdots}$$
$$\Downarrow \qquad\qquad\qquad \Downarrow$$
$$\cdots\square\square\underline{\square\square\cdots\square}\bullet\bullet y_2 q_2 z_2 \bullet \cdots \bullet y_k q_k z_k \bullet \underline{\tilde{y}_1\tilde{q}_1\tilde{z}_1 \bullet \tilde{y}_2\tilde{q}_2\tilde{z}_2 \bullet \cdots \bullet \tilde{y}_r\tilde{q}_r\tilde{z}_r \bullet\bullet\square\square\cdots}$$

Suppose each transition set $\delta_N(q,a)$ has size at most r. Then after simulating t steps of N, the tape string of M encoding $O(r^t)$ different configurations of N and therefore has length $L = O(tr^t)$ (not counting the initial blanks). If M begins each simulation phase by moving the initial configuration from the beginning to the end of the tape string, which takes $O(t^2 r^t)$ time, the time for the rest of the the simulation phase is negligible. Altogether, simulating all r^t possibilities for the the tth step of N requires $O(t^2 r^{2t})$ time. We conclude that M can simulate the first t steps of every computation path of N in $\boldsymbol{O(t^2 r^{2t})}$ ***time***, as claimed. □

The running time of this simulation is dominated by the time spent reading from one end of the tape string and writing to the other. It is fairly easy to reduce the running time to $O(tr^t)$ by using either two tapes (a "read tape" containing N-configurations at time t and a "write tape" containing N-configurations at time $t+1$) or two independent heads on the same tape (one at each end of the queue).

38.5 Nondeterminism as Advice

Any nondeterministic Turing machine N can also be simulated by a *deterministic* machine M with *two* inputs: the user input string $w \in \Sigma^*$, and a so-called ***advice*** string $x \in \Omega^*$, where Ω is another finite alphabet. Only the first input string w is actually given by the user. At least for now, we assume that the advice string x is given on a separate read-only tape.

The deterministic machine M simulates N step-by-step, but whenever N has a choice of how to transition, M reads a new symbol from the advice string, and that symbol determines the choice. In fact, without loss of generality, we can assume that M reads a new symbol from the advice string and moves the advice-tape's head to the right on *every* transition. Thus, M's transition function has the form $\delta_M : Q \times \Gamma \times \Omega \to Q \times \Gamma \times \{-1, +1\}$, and we require that

$$\delta_N(q,a) = \{\delta_M(q,a,\omega) \mid \omega \in \Omega\}$$

For example, if N has a binary choice

$$\delta_N(\mathsf{branch}, \mathsf{?}) = \big\{(\mathsf{left}, \mathsf{L}, -1), (\mathsf{right}, \mathsf{R}, +1)\big\},$$

then M might determine this choice by defining

$$\delta_M(\mathsf{branch}, \mathsf{?}, \mathtt{0}) = (\mathsf{left}, \mathsf{L}, -1) \qquad \text{and} \qquad \delta_M(\mathsf{branch}, \mathsf{?}, \mathtt{1}) = (\mathsf{right}, \mathsf{R}, +1)$$

More generally, if every set $\delta_N(p,a)$ has size r, then we let $\Omega = \{1, 2, \ldots, r\}$ and define $\delta_M(q,a,i)$ to be the ith element of $\delta_N(q,a)$ in some canonical order.

Now observe that N accepts a string w if and only if M accepts the pair (w,x) for *some* string $x \in \Omega^*$, and N rejects w if and only if M rejects the pair (w,x) for *all* strings $x \in \Omega^*$.

The "advice" formulation of nondeterminism allows a different strategy for simulation by a standard deterministic Turing machine, which is often called ***dovetailing***. Consider all possible advice strings x, in increasing order of length; listing these advice strings is equivalent to repeatedly incrementing a base-r counter. For each advice string x, simulate M on input (w, x) for exactly $|x|$ transitions.

```
DOVETAIL_M(w):
  for t ← 1 to ∞
      done ← TRUE
      for all strings x ∈ Ω^t
          if M accepts (w, x) in at most t steps
              accept
          if M(w, x) does not halt in at most t steps
              done ← FALSE
      if done
          reject
```

The most straightforward Turing-machine implementation of this algorithm requires three tapes: A read-only input tape containing w, an advice tape containing x (which is also used as a timer for the simulation), and the work tape. This simulation requires $O(tr^t)$ time to simulate all possibilities for t steps of the original non-deterministic machine N.

If we insist on using a standard Turing machine with a single tape and a single head, the simulation becomes slightly more complex, but (unlike our earlier queue-based strategy) not significantly slower. This standard machine S maintains a string of the form $\bullet w \bullet x \bullet z$, where z is the current work-tape string of M (or equivalently, of N), with marks (on a second track) indicating the current positions of the heads on M's work tape and M's advice tape. Simulating a single transition of M now requires $O(|x|)$ steps, because S needs to shuttle its single head between these two marks. Thus, S requires $\boldsymbol{O(t^2 r^t)}$ ***time*** to simulate all possibilities for t steps of the original non-deterministic machine N. This is significantly faster than the queue-based simulation, because we don't record (and therefore don't have to repeatedly scan over) intermediate configurations; recomputing everything from scratch is actually cheaper!

38.6 The Cook-Levin Theorem

★★★

Define SAT and CIRCUITSAT. Non-determinism is fundamentally different from other Turing machine extensions, in that it seems to provide an exponential speedup for some problems, just like NFAs can use exponentially fewer states than DFAs for the same language.

The Cook-Levin Theorem. *If SAT ∈ P, then P=NP.*

Proof: Let $L \subseteq \Sigma^*$ be an arbitrary language in NP, over some fixed alphabet Σ. There must be an integer k and Turing machine M that satisfies the following conditions:

- For all strings $w \in L$, there is at least one string $x \in \Sigma^*$ such that M accepts the string $w\square x$.
- For all strings $w \notin L$ and $x \in \Sigma^*$, M rejects the string $w\square x$.
- For all strings $w, x \in \Sigma^*$, M halts on input $w\square x$ after at most $\max\{1, |w|^k\}$ steps.

Now suppose we are given a string $w \in \Sigma^*$. Let $n = |w|$ and let $N = \max\{1, |w|^k\}$. We construct a boolean formula Φ_w that is satisfiable if and only if $w \in L$, by following the execution of M on input $w\square x$ for some unknown advice string x. Without loss of generality, we can assume that $|x| = N - n - 1$ (since we can extend any shorter string x with blanks.) Our formula Φ_w uses the following boolean variables for all symbols $a \in \Gamma$, all states $q \in Q$, and all integers $0 \le t \le N$ and $0 \le i \le N + 1$.

- $Q_{t,i,q}$ — M is in state q with its head at position i after t transitions.
- $T_{t,i,a}$ — The kth cell of M's work tape contains a after t transitions.

The formula Φ_w is the conjunction of the following constraints:

- **Boundaries:** To simplify later constraints, we include artificial boundary variables just past both ends of the tape:

$$\begin{aligned} Q_{t,i,q} = Q_{t,N+1,q} &= \textsc{False} && \text{for all } 0 \le t \le N \text{ and } q \in Q \\ T_{t,0,a} = T_{t,N+1,a} &= \textsc{False} && \text{for all } 0 \le t \le N \text{ and } a \in \Gamma \end{aligned}$$

- **Initialization:** We have the following values for variables with $t = 0$:

$$\begin{aligned} Q_{0,1,\text{start}} &= \textsc{True} \\ Q_{0,1,q} &= \textsc{False} && \text{for all } q \neq \text{start} \\ H_{0,i,q} &= \textsc{False} && \text{for all } i \neq 1 \text{ and } q \in Q \\ T_{0,i,w_i} &= \textsc{True} && \text{for all } 1 \le i \le n \\ T_{0,i,a} &= \textsc{False} && \text{for all } 1 \le i \le n \text{ and } a \neq w_i \\ T_{0,n+1,\square} &= \textsc{True} \\ T_{0,i,a} &= \textsc{False} && \text{for all } a \neq \square \end{aligned}$$

- **Uniqueness:** The variables $T_{0,i,a}$ with $n+2 \le i \le N$ represent the unknown advice string x; these are the "inputs" to Φ_w. We need some additional constraints ensure that for each i, *exactly one* of these variables is $\textsc{True}$:

$$\left(\bigvee_{a \in \Gamma} T_{0,j,a}\right) \wedge \bigwedge_{a \neq b} \left(\overline{T_{0,j,a}} \vee \overline{T_{0,j,b}}\right)$$

- **Transitions:** For all $1 \le t \le N$ and $1 \le i \le N$, the following constraints simulate the transition from time $t-1$ to time t.

$$Q_{t,i,q} = \bigvee_{\delta(p,a)=(q,\cdot,+1)} \left(Q_{t-1,i-1,p} \wedge T_{t,i-1,a}\right) \vee \bigvee_{\delta(p,a)=(q,\cdot,-1)} \left(Q_{t-i,i+1,p} \wedge T_{t,i+1,a}\right)$$

$$T_{t,i,b} = \bigvee_{\delta(p,a)=(\cdot,b,\cdot)} \left(Q_{t-1,i,p} \wedge T_{t-1,i,a}\right) \vee \left(\bigwedge_{q \in Q} \overline{Q_{t-1,i,q}} \wedge T_{t-1,i,b}\right)$$

- **Output:** We have one final constraint that indicates acceptance.

$$z = \bigvee_{t=0}^{N} \bigvee_{i=1}^{N} Q_{t,i,\text{accept}}$$

A straightforward induction argument implies that *without the acceptance constraint*, any assignment of values to the unknown variables $T_{0,i,a}$ that satisfies the uniqueness constraints determines *unique* values for the other variables in Φ_w, which consistently describe the execution of M. Thus, Φ_w is satisfiable if and only if for some input values $T_{0,i,a}$, the resulting , including acceptance. In other words, Φ_w is

satisfiable if and only if there is a string $x \in \Gamma^*$ such that M accepts the input $w\square x$. We conclude that Φ_w is satisfiable if and only if $w \in L$.

For any any string w of length n, the formula Φ_w has $O(N^2)$ variables and $O(N^2)$ constraints (where the hidden constants depend on the machine M). Every constraint except acceptance has constant length, so altogether Φ_w has length $O(N^2)$. Moreover, we can construct Φ_w in $O(N^2) = O(n^{2k})$ time.

In conclusion, if we could decide SAT for formulas of size n in (say) $O(n^c)$ time, then we could decide membership in L in $O(n^{2kc})$ time, which implies that $L \in \mathrm{P}$. □

Exercises

1. Prove that the following problem is NP-hard, *without* using the Cook-Levin Theorem. Given a string $\langle M, w\rangle$ that encodes a non-deterministic Turing machine M and a string w, does M accept w in at most $|w|$ transitions?

★★★ More excerises!