Lecture 2: Introduction to Turing Machines

SAT = the set of satisfiable CNF propositional formulae. We start with a formula e and clauses $C_1, C_2, \ldots, C_m$; $x_1, \ldots, x_n$ are Boolean variables. That is, $e = C_1 \wedge C2 \wedge \cdots \wedge C_m$ and $C_i = x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,j(i)}$

k-SAT: Same as SAT, except $j(i) \leq k$, $i \leq j \leq n$ (that is, each clause has at most k Boolean variables)
k-SAT $\leq_P$ (k-1)-SAT for $k \geq 4$ (not $k = 3$)

Turing Machines: See pages 3 and 4.

Facts and definitions about Turing machines and complexity classes:

A non-deterministic TM has same the definition as a deterministic TM, but has multiple (some constant number of) δs. At each step, the TM can use any one of these transition functions. In at most $T(n)$ steps, the machine halts ($n = |x|$, where x is the input). A non-deterministic TM is not "random" – think of it as a tree. Each node represents a choice; each path from root to leaf represents a possible computation.

A TM M "recognizes" language L in $T(n)$ if M runs in time $T(n)$ and $\forall x \in L$, $M(x)$ outputs 1, otherwise 0.

$T : \mathbb{N} \to \mathbb{N}$ is a time-constructible function if $T(n) \geq n$ and there is a TM M that computes the result from an input x in time T. That is, there exists a machine that counts how many steps are taken on an input.

Optional exercise: Write a Turing machine for a counter.

Some important things to remember:

1. If some binary function is computable in time T, and T is time constructible, and M has alphabet Γ, then f is computable in time $4 \log |\Gamma| T$ by a TM that uses the alphabet $\{\triangleright, \Box, 0, 1\}$.

2. If you have a language that you can recognize in time T with k work tapes, then you can also recognize it in time $5kT^2$ with one work tape.

3. If you can recognize a language in time T with a bidirectional machine, then you can do the same using a unidirectional machine in time $4T$ (note: should be $2T$).

4. Since the definition of a Turing machine is finite (it's a program, and a program is finite), we can encode its definition in binary. There exists a universal Turing machine U (see theorem 1.9). For every x and α in $\{0, 1\}^*$, $U(x, \alpha) = M_\alpha(x)$ – the universal TM is running the machine

encoded by α on input x. Moreover, if M_α halts in T steps on input x, then U halts in $CT \log T$ on input (x, α). Note that C is independent of the length of x; it depends on M_α (size of tape alphabet, etc.)

$f : \mathbb{N} \to \mathbb{N}$ is a space constructible function if it is non-decreasing and there exists a Turing machine that on input 1^n outputs the binary representation of $f(n)$ using $O(f(n))$ space. Note that if f is space constructible, then there exists a Turing machine that on input 1^n marks off exactly $f(n)$ cells on its work tapes (say, using a special symbol) without ever exceeding $O(f(n))$ space.

The statement "M recognizes language L in DTIME(T)" = "there exists a TM M that recognizes L in time $O(T)$." Why do we use O? We don?t want to allow different machine architectures to change the meaning of our running time statement.

$P = \bigcup_{c \in \mathbb{N}} DTIME(n^c)$

$L \in NP$ means that $\exists$ a poly-time M (called the verifier) and a poly q such that for every $x \in \{0,1\}^*$, $x \in L$ if there is another string w ($w \in \{0,1\}^{q(|x|)}$) and $M(x, w) = 1$. We call this w a witness for the membership of x in L.

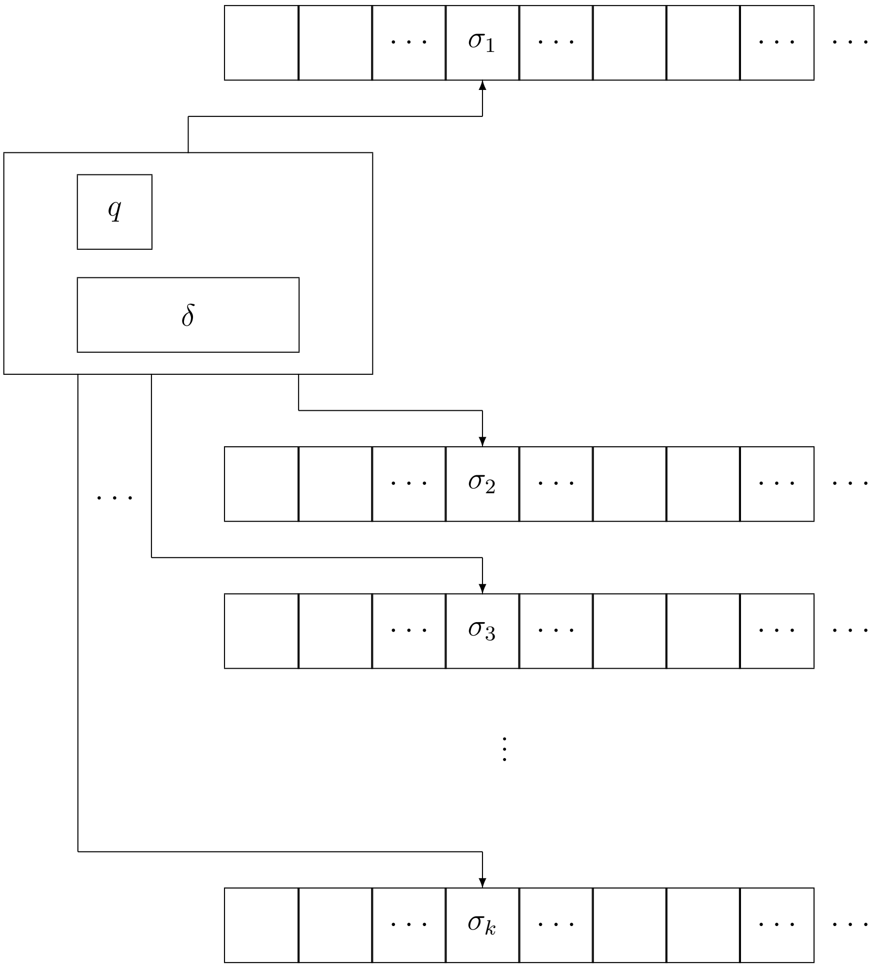
This is different from solving the problem – we are simply verifying a solution, not *finding* one. For example, let the input be a formula that belongs to SAT; w is an assignment that satisfies it – w is thus relatively short.

$L \leq_P L'$: *Many-to-one poly-time reducibility (Karp reducibility)*

If there exists a poly-time computable function f such that $x \in L \Leftrightarrow f(x) \in L'$, we say that L is NP complete if $L \in NP$, $\forall S \in NP$, S is many-to-one poly-time reducible to L.

Turing-Machine model of Computation

Deterministic k-tape Turing machine M.



There is one read-only **input tape** (on top) and $k-1$ read-write **work/output tapes**. M is a triple Γ, Q, δ that is defined as follows:

- Γ is the **tape alphabet**, a finite set of symbols. Assume $\Box$ ("blank" symbol), $\triangleright$ ("start" symbol), 0 and 1 are four distinct elements of Γ.
- Q is the **state set**, a finite set of states that M's control register can be in. Assume q_{start} and q_{halt} are two distinct states in Q.
- δ is the **transition function**, a finite table that describes the rules (or program) by which M operates:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times (L, S, R)^k.$$

$\delta(q, (\sigma_1, ..., \sigma_k)) = (q', (\sigma'_2, ..., \sigma'_k), (z_1, ..., z_k))$ means that, if M is in state q, and the read (or read/write) tape heads are pointing at the cells containing $\sigma_1, ..., \sigma_k$, then the following "step" of the computation is performed:

- the read/write tape symbols $\sigma_2, ..., \sigma_k$ are replaced by $\sigma'_2, ..., \sigma'_k$;
- tape head i moves left, stays in place or moves right, depending on whether z_i is in L,S or R;
- the control-register state is changed to q'.

When M starts its execution on input $x = \sigma_1, ..., \sigma_n$, we have

- $q = q_{\text{start}}$
- input tape

$\triangleright$	σ_1	σ_2	$\cdots$	σ_n	$\square$	$\square$	$\cdots$	$\cdots$

- all other tapes

$\triangleright$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\cdots$	$\cdots$

Meaning of q_{halt}:

$$\delta(q_{\text{halt}}, (\sigma_1, ..., \sigma_k)) = (q_{\text{halt}}, (\sigma_2, ..., \sigma_k), S^k) \qquad \forall (\sigma_1, ..., \sigma_k).$$

Designate one of the read/write tapes as "the output tape".

Turing machine M "computes the function f", if for all $x \in \Gamma^*$ the execution of M on input x eventually reaches the state q_{halt}, and when it does, the contents of M's output tape is $f(x)$.

M "runs in time T" if for all n and all $x \in \Gamma^n$ M halts after at most $T(n)$ steps.