

Recap: From Turing Machines to Boolean Circuits

Computational Complexity 2026/2027 — Massimo Lauria

Recap of Turing machines, robustness of the definition, oblivious machines — then the simulation of a $T(n)$ -time machine by a Boolean circuit of size $O(T(n) \log T(n))$.

READING. [AB, Chapter 1] (esp. §1.2, §1.3, §1.7), and §6.1.1 for the proof of Theorem 5.

Everything before the last section is recap of things you saw in previous courses: we fix notation and collect the normal-form statements we need. The point of this lecture note is to collect the preliminaries for Theorem 5.

The Turing machine: the general model of computation

The Turing machine is the model that defines what is computable and what is not. The Church-Turing thesis asserts that this is indeed the appropriate definition, in the sense that captures the epistemic notion of computation.

DEFINITION 1 (k -tape Turing machine). A TM is a triple $M = (\Gamma, Q, \delta)$ where Γ is a finite alphabet containing the blank $\square$ and the start marker $\triangleright$, Q is a finite set of states containing q_{start} and q_{halt} , and

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$$

is the transition function. The machine has k tapes, each infinite to the right, each with one head:

- **tape 0**: the *input tape*, read-only, initialised to $\triangleright x \square \square \dots$;
- **tapes 1, $\dots$, $k-1$** : *work tapes*, initialised to $\triangleright \square \square \dots$;
- **tape $k-1$** : also serves as the *output tape*.

A *configuration* records the state, the contents of all tapes and all head positions. Computation starts in q_{start} with all heads on cell 0 of each tape. The proceeds by applying δ ; given the current state and the symbol under each head, δ specifies the next state, what to write under the head on the work tapes, and whether to move each head left, right, or leave it alone. The computation stops when q_{halt} is reached. We denote as $M(x)$ the output of the computation, namely the content of the output tape at that moment. The output is undefined if M does not halt on input x .

Since we care about complexity, and not just computability, we need a measure of efficiency. The first and most direct one is the running time.

DEFINITION 2 (Running time). M runs in time $T : \mathbb{N} \rightarrow \mathbb{N}$ if for every input x it halts within $T(|x|)$ steps.

A boring technicality: many of the theorem that we will mention about Turing machines hold for *time constructible* functions. In practice, some of the constructions and manipulation require to compute

[AB, Definition 1.3]. Note the convention: tape 0 is *read-only*, tapes $1, \dots, k-1$ are read/write, and the last one doubles as output tape.

Note that δ writes only on tapes $1, \dots, k-1$: the read-only status of tape 0 is built into the type of δ .

[AB, Definition 1.3], and Definition 1.12 for the class **DTIME**. This is a *worst-case* measure: a single bad input of length n determines $T(n)$.

or estimate the running time $T(n)$ of the machine itself on an input of length n , assuming T is known. If T is too crazy, this may not be possible, and some of the following theorem do not hold. A function T is time constructible if $T(n) \geq n$, and there is a Turing machine that given n , in binary, outputs $T(n)$ in binary, in time at most $T(n)$. Observe that $n, n^2, n \log n, n^c, 2^{\sqrt{n}}, 2^n, n!$ are all time constructible functions. In the future we will ignore this issue and assume that all time bounds we consider are time constructible.

Robustness of the definition

In definition above essentially every parameter can be changed without altering *computability* at all, and altering *complexity* only by a controlled amount. For example:

[AB, §1.3.1, Claims 1.5, 1.6, 1.8 and Remark 1.7]. See also Exercise 1.9 on random access memory.

- **single tape:** k tapes can be simulated with 1 read/write tape at cost $T \mapsto O(kT^2)$ [AB, Claim 1.6]; we may see later in the course that this simulation is optimal;
- **fixed number of tapes:** k tapes can be simulated with 1 readonly input tape and 2 read/write work tapes at cost $T \mapsto O(T \log T)$ [Hennie–Stearns 1966]; a more general version of this claim is in [AB]Theorem 1.9.
- **alphabet size:** any Γ can be translated into binary alphabet at cost $T \mapsto O(\log |\Gamma| \cdot T)$ [AB, Claim 1.5];
- **random access memory:** RAM machines and TMs simulate each other with polynomial overhead;
- **obliviousness:** see Definition 3 below.

Computability is insensitive to these choices; complexity is insensitive up to polynomial (often up to polylogarithmic) factors. This is exactly why classes such as **P** and **PSPACE** are meaningful objects, and also why statements at a finer granularity than polynomial, e.g. “linear” vs “quadratic” time, must be more precise about the model.

Functions vs. decision problems

A decision problem is a boolean function $f : \{0, 1\}^* \rightarrow \{0, 1\}$, hence with a yes/no answer. Any such a function can equivalently be represented as the language (i.e., a subset of $\{0, 1\}^*$). In particular f corresponds to language $L = \{x \mid f(x) = 1\}$. A TM that computes f just writes 0 or 1 on the output tape. An equivalent definition of TM that **decides a language** $L \subseteq \{0, 1\}^*$ is one where we replace q_{halt} by two states $q_{\text{accept}}, q_{\text{reject}}$, and require the machine to enter the first iff $x \in L$.

Decision problems are the natural setting for the whole complexity zoo (**P**, **NP**, **PSPACE**, ...), because the yes/no answer is a single bit and reductions compose cleanly.

The translation is trivial in both directions, try it as an exercise.

The normal form

In order to transform a TM into a circuit we do make our life as easy as possible enforcing the TM to be “oblivious”. This lets us *hard-wire* the circuit, since at “compile time” we already know which cell each head touches at each step.

DEFINITION 3 (Oblivious TM). M is *oblivious* if for every input x and every step i , the position of each head at step i depends only on $|x|$ and i — not on the contents of x . In particular the running time depends only on $|x|$.

The naive way to become oblivious is the single-tape simulation of Claim 1.6, whose sweeps are visibly input-independent; it costs a square. One can do much better.

THEOREM 4. For every time-constructible T and every TM M running in time $T(n)$ there is a machine $\tilde{M}$ computing the same function such that

1. $\tilde{M}$ has exactly one read-only input tape and two work/output tapes;
2. $\tilde{M}$ has binary alphabet;
3. $\tilde{M}$ is oblivious;
4. $\tilde{M}$ runs in time $O(T(n) \log T(n))$.

From machines to circuits

This is the actual goal of the lecture. Boolean circuits simulate efficiently TM.

THEOREM 5. Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be time-constructible and let M be a TM computing $f : \{0, 1\}^* \rightarrow \{0, 1\}$ in time $T(n)$. Then there is a family of Boolean circuits $\{C_n\}_{n \in \mathbb{N}}$ with

$$C_n(x) = M(x) \text{ for all } x \in \{0, 1\}^n, \quad \text{size}(C_n) = O(T(n) \log T(n)).$$

Full proof in class on in the textbook. By Theorem 4 we may assume M is oblivious, binary, with one read-only input tape and a constant number of work/output tapes, running in time $T'(n) = O(T(n) \log T(n))$. Fix n the construction depends on n only. $\square$

Remarks

- **Uniformity.** The construction is entirely explicit: given n , the circuit C_n can be written down in time $\text{poly}(T(n))$, indeed in space $O(\log T(n))$.
- **Corollary: $\mathbf{P} \subseteq \mathbf{P/poly}$.** With $T(n) = n^{O(1)}$ we get circuits of size $n^{O(1)}$ [AB, Theorem 6.6]. The inclusion is strict, since unary undecidable languages have linear-size circuits [AB, Claim 6.8].
- **Corollary: Cook–Levin.** From this construction the Cook–Levin theorem will fall easily. Applying the construction to the verifier

[AB, Remark 1.7 and Exercise 1.5]. Obliviousness is the TM analogue of a *data-independent* program: the control flow is fixed, only the values flowing through it depend on x . This idea is useful for cryptographic systems as well. The position of program counter or RAM access would not disclose the content of x .

[AB, Remark 1.7 together with Exercise 1.6]; the $O(T \log T)$ two-tape simulation is by [Hennie and Stearns, 1966], and is the engine of the efficient universal TM [AB, Theorem 1.9, §1.7]. The oblivious version is usually credited to [Pippenger and Fischer, 1979]. The separate input tape is only necessary if we want to keep it read-only.

A weaker and much easier version — one r/o input tapes + one work tape, oblivious, time $O(T(n)^2)$ — is [AB, Exercise 1.5], and it is all that is needed for the Cook–Levin theorem [AB, Theorem 2.10]. The stronger version buys better parameters below.

[AB, Remark 6.7]. The point is that inputpos , workpos and prev are themselves computable in logarithmic space for the standard oblivious simulations, so the map $1^n \mapsto C_n$ is logspace computable. This is what “uniform circuit family” means, and it is the difference between $\mathbf{P}$ and $\mathbf{P/poly}$.

of an NP language turns membership into satisfiability of a circuit, giving NP-hardness of Circuit-SAT and thence of 3SAT [AB, Lemma 6.10, §6.1.2].

- **Where the log lives.** Every loss in the chain is a constant factor except the multitape $\rightarrow$ oblivious. Whether $O(T \log T)$ is optimal there is a genuine open question; so is the corresponding circuit question (does every T -time language have $O(T)$ -size circuits?).

Homework

- [AB, Sections 1.1, 1.2, 1.3, 1.7, 6.1].
- Think about these exercises, no need to do them in full details. [AB, Exercises 1.2, 1.3, 1.5, 1.6, 1.9].
- Check that the proof of Theorem 5 really needs obliviousness: what goes wrong, and how large does the circuit become, if head positions may depend on x ?

References

- Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009. ISBN 978-0-521-42426-4. DOI: 10.1017/CBO9780511804090.
- F. C. Hennie and Richard Edwin Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4), 1966. URL <https://doi.org/10.1145/321356.321362>.
- Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2), 1979. URL <https://doi.org/10.1145/322123.322138>.