# Turing machines and linear bounded automata

Informatics 2A: Lecture 30

John Longley

School of Informatics
University of Edinburgh
jrl@inf.ed.ac.uk

24 November 2017

## The Chomsky hierarchy: summary

| Level | Language type | Grammars | Accepting machines |
|---|---|---|---|
| 3 | Regular | $X \to \epsilon$, $X \to Y$, $X \to aY$ (regular) | NFAs (or DFAs) |
| 2 | Context-free | $X \to \beta$ (context-free) | NPDAs |
| 1 | Context-sensitive | $\alpha \to \beta$ with $\lvert\alpha\rvert \leq \lvert\beta\rvert$ (noncontracting) | Nondeterministic linear bounded automata |
| 0 | Recursively enumerable | $\alpha \to \beta$ (unrestricted) | Turing machines |

The material in red will be introduced today.

## The length restriction in noncontracting grammars

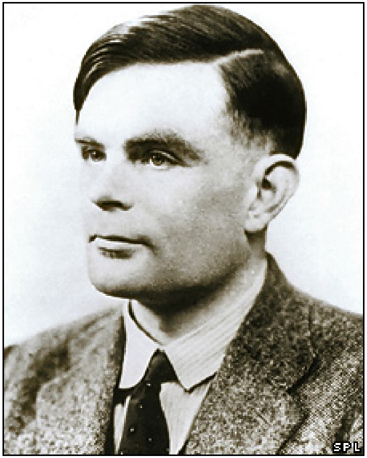
What's the effect of the restriction $|\alpha| \leq |\beta|$ in noncontracting grammar rules?

Idea: in a noncontracting derivation $S \Rightarrow \cdots \Rightarrow \cdots \Rightarrow s$ of a nonempty string $s$, all the sentential forms are of length at most $|s|$.

This means that if $L$ is context-sensitive, and we're trying to decide whether $s \in L$, we only need to consider possible sentential forms of length $\leq |s|$ ... and there are just finitely many of these. So in principle, we have the problem under control.

By contrast, without the length restriction, there's no upper limit on the length of intermediate forms that might appear in a derivation of $s$. So if we're searching for a derivation for $s$, how do we know when to stop looking? Intuitively, the problem here is wild and out of control. (This will be made more precise next lecture.)

## Alan Turing (1912–1954)

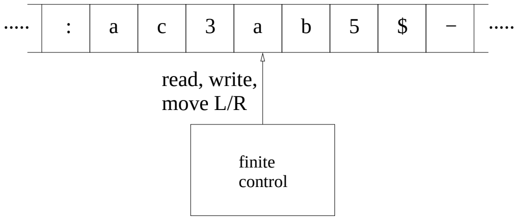


## Turing machines

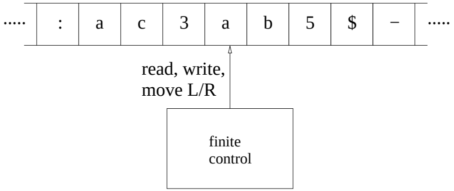
Recall that NFAs are 'essentially memoryless', whilst NPDAs are equipped with memory in the form of a stack.

To find the right kinds of machines for the top two Chomsky levels, we need to allow more general manipulation of memory.

A Turing machine consists of a finite-state control unit, equipped with a memory tape, infinite in both directions. Each cell on the tape contains a symbol drawn from a finite alphabet $\Gamma$.



## Turing machines, continued



At each step, the behaviour of the machine can depend on

- the current state of the control unit,
- the tape symbol at the current read position.

Depending on these things, the machine may then

- overwrite the current tape symbol with a new symbol,
- shift the tape left or right by one cell,
- jump to a new control state.

This happens repeatedly until (if ever) the control unit enters some identified final state.

## Turing machines, formally

A Turing machine $T$ consists of:

- A set $Q$ of control states
- An initial state $i \in Q$
- A final (accepting) state $f \in Q$
- A finite tape alphabet $\Gamma$
- An input alphabet $\Sigma \subseteq \Gamma$
- A blank symbol $- \in \Gamma - \Sigma$
- A transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

A nondeterministic Turing machine replaces the transition function $\delta$ with a transition relation $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$.

(Numerous variant definitions of Turing machine are possible. All lead to notions of TM of equivalent power.)

## Turing machines as acceptors

To use a Turing machine $T$ as an acceptor for a language over $\Sigma$, assume $\Sigma \subseteq \Gamma$, and set up the tape with the test string $s \in \Sigma^*$ written left-to-right starting at the read position, and with blank symbols everywhere else.

Then let the machine run (maybe overwriting $s$), and if it enters the final state, declare that the original string $s$ is accepted.

The language accepted by $T$ (written $\mathcal{L}(T)$) consists of all strings $s$ that are accepted in this way.

Theorem: A set $L \subseteq \Sigma^*$ is generated by some unrestricted (Type 0) grammar if and only if $L = \mathcal{L}(T)$ for some Turing machine $T$.

So both Type 0 grammars and Turing machines lead to the same class of recursively enumerable languages.

## Questions

Q1. Which is the most powerful class of language acceptors (i.e. provides acceptors for the widest class of languages)?

1. DFAs
2. NPDAs
3. Turing machines
4. Your laptop (with suitable programming)

## Questions

Q1. Which is the most powerful class of language acceptors (i.e. provides acceptors for the widest class of languages)?

Q2. Which is the least powerful (i.e. provides acceptors for the narrowest class of languages)?

1. DFAs
2. NPDAs
3. Turing machines
4. Your laptop (with suitable programming)

## Linear bounded automata

Suppose we modify our model to allow just a finite tape, initially containing just the test string $s$ with endmarkers on either side:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\vdash$ | h | e | y | | m | a | n | $\dashv$ |

The machine therefore has just a finite amount of memory, determined by the length of the input string. We call this a linear bounded automaton.

(LBAs are sometimes defined as having tape length bounded by a constant multiple of length of input. No essential difference.)

## Example: An LBA for $\{a^n b^n c^n \mid n \geq 1\}$

The tape alphabet for an LBA, though finite, might be considerably bigger than the input alphabet. So we can store more information on a tape than just an input string or related sentential form.

E.g. let $\Gamma = \{a, b, c, {\color{green}a}, {\color{green}b}, {\color{green}c}\}$, and $\Sigma = \{a, b, c\}$. Occurrences of green letters can serve as markers for positions of interest.

To test whether an input string has the form $a^n b^n c^n$:

1. Scan string to ensure it has form $a^k b^m c^n$ for $k, m, n \geq 1$.
   Along the way, mark leftmost $a, b, c$. E.g. ${\color{green}a}aa{\color{green}b}bb{\color{green}c}cc$.
2. Scan string again to see if it's the rightmost $a, b, c$ that are marked.
   If yes for all three, ACCEPT.
   If yes for some but not all of $a, b, c$, REJECT.
3. Scan string again moving the 'markers' one position to the right.
   E.g. ${\color{green}a}aa{\color{green}b}bb{\color{green}c}cc$ becomes $a{\color{green}a}ab{\color{green}b}bc{\color{green}c}c$. Then Go to 2.

All this can be done by a (deterministic) LBA.

## Recommended exercise . . .

In a similar spirit, outline how the language

$$\{ss \mid s \in \{a, b\}^*\}$$

could be recognized by a deterministic LBA.

## LBAs and context-sensitive languages

Theorem: A language $L \subseteq \Sigma^*$ is context-sensitive if and only if $L = \mathcal{L}(T)$ for some non-deterministic linear bounded automaton $T$.

Rough idea: we can guess at a derivation for $s$. We can check each step since each sentential form fits onto the tape.

To implement this by an LBA, expand the tape alphabet so that the tape can simultaneously store:

- the input string
- two successive sentential forms in a derivation
- a few 'markers' used to check the validity of the derivation step.

The context-sensitive grammar rules themselves will be hardwired into the design of the LBA.

## A non-context-sensitive language

Sentences of first-order predicate logic can be regarded as strings over a certain alphabet, with symbols like $\forall, \wedge, x, =, (, )$ etc.

Let $P$ be the language consisting of all such sentences that are provable using the rules of FOPL. E.g. the sentence

$$(\forall x.\, A(x)) \ \wedge\ (\forall x.\, B(x)) \ \Rightarrow\ (\forall x.\, A(x) \wedge B(x))$$

is in $P$. But the sentence

$$(\exists x.\, A(x)) \ \wedge\ (\exists x.\, B(x)) \ \Rightarrow\ (\exists x.\, A(x) \wedge B(x))$$

is not in $P$, even though it's syntactically well-formed.

Theorem: $P$ is recursively enumerable, but not context-sensitive.

Intuition: to show $s \in P$, we'd in effect have to construct a proof of $s$. But in general, a proof of $s$ might involve formulae much, much longer than $s$ itself, which wouldn't fit on the LBA tape.

Put another way, mathematics itself is wild and out of control!

## Determinism vs. non-determinism: a curiosity

- At the bottom level of the Chomsky hierarchy, it makes no difference: every NFA can be simulated by a DFA.
- At the top level, the same happens. Any nondeterministic Turing machine can be simulated by a deterministic one.
- At the context-free level, there is a difference: we need NPDAs to account for all context-free languages.

  (Reason: Context-free languages aren't closed under complementation, see last lecture. However, if $L$ is accepted by a DPDA then so is its complement.)
- What about the context-sensitive level? Are NLBAs strictly more powerful than DLBAs? Asked in 1964, and still open!!

  Can't use the above argument here because CSLs are closed under complementation (shown in 1988).

## Detecting non-acceptance: LBAs versus TMs

Suppose $T$ is an LBA. How might we detect that $s$ is not in $\mathcal{L}(T)$?

Clearly, if there's an accepting computation for $s$, there's one that doesn't pass through exactly the same machine configuration twice (if it did, we could shorten it).

Since the tape is finite, the total number of machine configurations is finite (though ridiculously large). So in theory, if $T$ runs for long enough without reaching the final state, it will enter the same configuration twice, and we may as well abort.

Note that on this view, repeated configurations would be spotted not by $T$ itself, but by 'us looking on', or perhaps by some super-machine spying on $T$.

For Turing machines with unlimited tape space, this reasoning doesn't work. Is there some general way of spotting that a computation isn't going to terminate ?? See next lecture . . .

## Wider significance of Turing machines

Turing machines are important because (it's generally believed that) any symbolic computation that can be done by any mechanical procedure or algorithm can in principle be done by a Turing machine. This is called the Church-Turing Thesis.

E.g.:

- Any language $L \subseteq \Sigma^*$ that can be 'recognized' by some mechanical procedure can be recognized by a TM.
- Any mathematical function $f : \mathbb{N} \to \mathbb{N}$ that can be computed by a mechanical procedure can be computed by a TM (e.g. representing integers in binary, and requiring the TM to write the result onto the tape.)

## Status of Church-Turing Thesis

The CT Thesis is a somewhat informal statement insofar as the general notion of a mechanical procedure isn't formally defined (although we have a pretty good idea of what we mean by it).

Although a certain amount of philosophical hair-splitting is possible, the broad idea behind CTT is generally accepted.

At any rate, anything that can be done on any present-day computer (even disregarding time/memory limitations) can in principle be done on a TM.

So if we buy into CTT, theorems about what TMs can/can't do can be interpreted as fundamental statements about what can/can't be accomplished by mechanical computation in general.

We'll see some examples of such theorems next time.