Linear bounded automata and context-sensitive grammars, Intro to computability theory

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2024

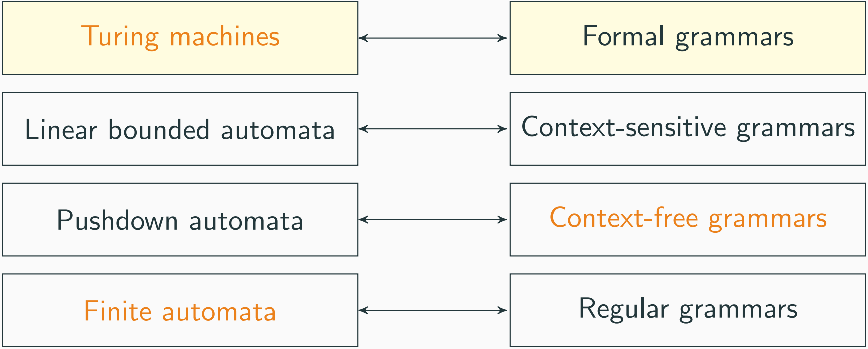
** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 10

- Turing machine: two-way infinite tape, read, write, move head
- Accept iff in a final state; configurations
- TMs with output, computing a function
- Recursively enumerable vs. recursive languages (always halt).
- Construction tricks:
  - storage in state
  - multiple tracks (on a single tape)
- Variants of TMs:
  - multi-tape (independent heads),
  - nondeterministic (accept iff some choices lead to final state)

3.3 Turing Machines and grammars

Chomsky hierarchy: Type 0



Theorem

A language is recursively enumerable, if and only if it is generated by a Type 0 grammar.

Turing machine to grammar

- First generate the relevant portion of the tape and a copy of the input word (nonterminal $\underline{X}$ for each $x \in \Gamma$, in reverse)
- Why? TM can rewrite w, G must generate it, cannot modify
- We have $wB^n\underline{W}^RQ_0B^m$, where B^n, B^m is sufficient free space
- Then simulate moves (essentially reverse configs+free space)
- In a final state erase the simulated tape, keep only w

$G = (\{S, C, D, E\} \cup \{\underline{X}\}_{x \in \Gamma} \cup \{Q_i\}_{q_i \in Q}, \Sigma, \mathcal{P}, S)$ where $\mathcal{P}$ is:

(1)	$S \to DQ_0E$	simulation starts in initial state
	$D \to xD\underline{X} \mid E$	generate input word, reverse copy for simulation
	$E \to BE \mid \epsilon$	generate sufficient free space for simulation
(2)	$\underline{X}P \to Q\underline{X'}$	for all $\delta(p, x) = (q, x', R)$ [direction reversed!]
	$\underline{X}P\underline{Y} \to \underline{X'}\,\underline{Y}Q$	for all $\delta(p, x) = (q, x', L)$
(3)	$P \to C$	for all $p \in F$
	$C\underline{X} \to C$,$\underline{X}C \to C$	clean the tape
	$C \to \epsilon$	finish, generated w

Example: $L = \{a^{2n} | n \geq 0\}$

$M = (\{q_0, q_1, q_2, q_F\}, \{a\}, \{a\}, \delta, q_0, B, \{q_F\})$ where

$$\delta(q_0, a) = (q_1, a, R),$$
$$\delta(q_1, a) = (q_0, a, R),$$
$$\delta(q_0, B) = (q_F, B, L)$$

$G = (\{S, C, D, E, Q_0, Q_1, Q_F, \underline{a}\}, \{a\}, S, \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3)$

Initialize: $\mathcal{P}_1$

$S \to DQ_0E$

$D \to aD\underline{a} \mid E$

$E \to BE \mid \epsilon$

Simulate: $\mathcal{P}_2$

$\underline{a}Q_0 \to Q_1\underline{a}$

$\underline{a}Q_1 \to Q_0\underline{a}$

$BQ_0\underline{a} \to B\underline{a}Q_F$

Cleanup: $\mathcal{P}_3$

$Q_F \to C$

$C\underline{a} \to C$

$\underline{a}C \to C$

$BC \to C$

$C \to \epsilon$

For $w = aa$: initialize $aaB\underline{aa}Q_0$, simulate $aaB\underline{a}Q_F\underline{a}$, cleanup: aa

Proof

$L(M) \subseteq L(G)$

- For $w \in L(M)$ there is a finite accepting sequence of moves
- The grammar generates sufficient space
- Then we simulate the moves
- Finally clean non-input symbols

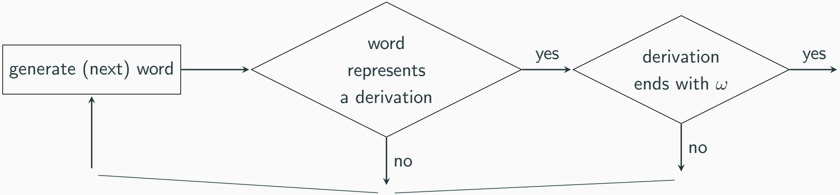
$L(G) \subseteq L(M)$

- Steps in a derivation for $w \in L(G)$ may be in different order
- But we can reorder them into the phases (1), (2), (3)
- Since we eliminated the underlined symbols, we must have generated the cleaning variable C
- In order to generate C we must have generated a final state
- A final state can only be generated from the initial state by a sequence of simulated moves $\square$

Grammar to Turing machine

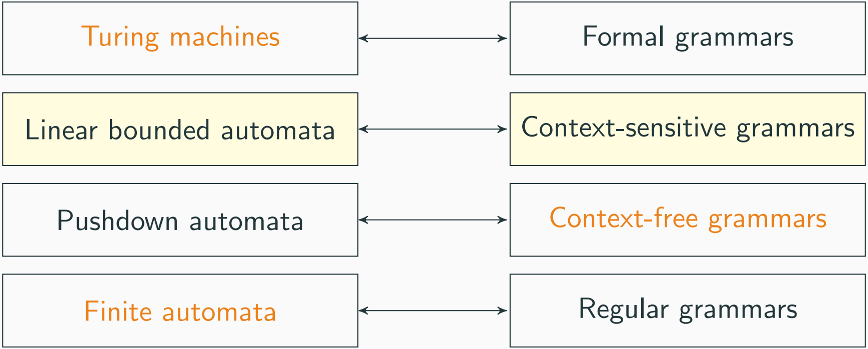
Idea: The TM sequentially generates all possible derivations.
(Note: here we do not care about efficiency.)

- code $S \Rightarrow \beta_1 \Rightarrow \ldots \Rightarrow \beta_n = \omega$ as a string $\#S\#\beta_1\#\ldots\#\omega\#$
- construct a TM accepting exactly $\#\alpha\#\beta\#$ where $\alpha \Rightarrow \beta$
- construct a TM accepting $\#\beta_1\#\ldots\#\beta_k\#$ where $\beta_1 \Rightarrow^* \beta_k$
- construct a TM generating sequentially all possible strings
- check if the string is a valid derivation ending with ω



3.4 Linear bounded automata and context-sensitive grammars

Chomsky hierarchy: Type 1



Context-sensitive languages

Theorem

The following are equivalent for a language L:

(i) *L is generated by a context-sensitive grammar.*

(ii) *L is generated by a monotone grammar.*

(iii) *L is recognized by a Linear Bounded Automaton (LBA).*

- context-sensitive grammar: $\alpha_1 A \alpha_2 \to \alpha_1 \gamma \alpha_2$ where $A \in V$, $\gamma \in (V \cup T)^+$, $\alpha_1, \alpha_2 \in (V \cup T)^*$ ($S \to \epsilon$ if S not in bodies)
- monotone grammar: $\alpha \to \beta$ where $|\alpha| \leq |\beta|$
- Linear Bounded Automaton (LBA): a nondeterministic TM only using the input portion of the tape [we formalize later]

Note: Context-sensitive grammars are monotone, $(i) \Rightarrow (ii)$ trivial. Monotone grammars do not shorten sentential forms in a derivation

Example: $L = \{a^n b^n c^n | n \geq 1\}$ is context-sensitive

(Recall that L is not context-free.)

A monotone grammar:

$S \to aSBC \mid abC$	*right amount of* a, B, C
$CB \to BC$	*reorder to* $a^n b B^{n-1} C^n$
$bB \to bb$	$B \to b$ *only if preceded by* b
$bC \to bc$	$C \to c$ *only if preceded by* b
$cC \to cc$	*. . . or by* c

The rule $CB \to BC$ is not context-sensitive. But we can convert it to a chain of context-sensitive rules:

$$CB \to XB,\ XB \to XY,\ XY \to BY,\ BY \to BC$$

(Same for any monotone rule, as long as there are no terminals.)

From monotone to context-sensitive grammars $(ii) \Rightarrow (i)$

Recall: separated grammar means productions of the form $\alpha \to \beta$ where either $\alpha, \beta \in V^+$ or $\alpha \in V, \beta \in T \cup \{\epsilon\}$

Lemma

Every monotone grammar can be converted to an equivalent context-sensitive grammar.

Proof: First, convert to separated grammar (as for ChNF). This preserves monotonicity, $V_a \to a$ is monotone, context-sensitive.

Then, convert every production $A_1 \ldots A_m \to B_1 \ldots B_n$ ($m \leq n$) to the following chain (using new auxiliary variables C_i):

$$\begin{array}{cc}
A_1 A_2 \ldots A_m \to C_1 A_2 \ldots A_m & C_1 C_2 \ldots C_m \to B_1 C_2 \ldots C_m \\
C_1 A_2 \ldots A_m \to C_1 C_2 \ldots A_m & B_1 C_2 \ldots C_m \to B_1 B_2 \ldots C_m \\
\vdots & \vdots \\
C_1 \ldots C_{m-1} A_m \to C_1 \ldots C_{m-1} C_m & B_1 \ldots B_{m-1} C_m \to B_1 \ldots B_{m-1} B_m \ldots B_n
\end{array}$$