Lecture 35

The generator side of Turing Machines...

Generalized notion of a grammar:

- left-hand-side of a grammar rule can have *terminals and non-terminals* (but at least one non-terminal)
- in a derivation, remove the entire left-hand-side and replace it with the right-hand-side of the rule

terminology:

"grammar" = "rewriting system" = "unrestricted grammar"

Def. A **grammar** is a quadruple $G = (V, \Sigma, R, S)$, where

V is an alphabet
$\Sigma \subseteq V$ is the set of terminal symbols
$(V-\Sigma)$ are non-terminals
$S \in (V-\Sigma)$ is the start symbol
R (the set of rules) is a finite subset of
$(V^* (V-\Sigma) V^*) \times V^*$

all other grammar-related terminology still applies here:

yields in one step $\Rightarrow$
reflexive transitive closure $\Rightarrow^*$

Example. $\{a^n b^n c^n : n \geq 0\}$

Let $G = (V, \Sigma, R, S)$, where

$V = \{S, B, B_1, a, b, c\}$
$\Sigma = \{a, b, c\}$

and R contains:

$S \rightarrow e$
$S \rightarrow aSB$
$B \rightarrow B_1c$
$cB_1 \rightarrow B_1c$
$aB_1 \rightarrow ab$
$bB_1 \rightarrow bb$

consider the derivation of $a^3b^3c^3$:

$S \Rightarrow aSB$
$\Rightarrow aaSBB$
$\Rightarrow aaaSBBB$
$\Rightarrow aaaBBB$
$\Rightarrow aaaB_1cB_1cB_1c$
$\Rightarrow aaabcB_1cB_1c$
$\Rightarrow aaabB1cB1cc$
$\Rightarrow aaabbB1ccc \Rightarrow aaabbbccc$

Thm. A language is generated by a grammar iff it is recursively enumerable.

(=>) Grammar -> TM is easy. Simulate derivations on the TM.

(<=) TM -> Grammar.

Every Turing Machine can be simulated by a grammar:

- start with any Turing Machine; represent configurations as strings
- construct a grammar that can derive one of these strings from another iff the corresponding configurations stand in the "yields" relation ($|\!\!-\!\!-^*$) of the TM

Lemma. Let $M = (K, \Sigma, \delta, s, H)$ be any Turing Machine. Then there is a grammar G such that for any two configurations (q, u, a, v) and (q', u', a', v') of M,

$$(q, u, a, v) \; |\!\!-\!\!-^* \; (q', u', a', v')$$

iff

$$\boxtimes u q a v \boxleftarrow \Rightarrow^* \boxtimes u' q' a' v' \boxleftarrow$$

Proof (Idea behind proof, that is).

Note

position of state serves as an indicator of where the tape head is.

in general, only state, scanned symbol, and head position need to be changed when passing from one configuration to the next.

$\boxleftarrow$ helps to deal with blank tape

Let's define $G = (V, \Sigma, R, S)$ as follows:

$V = K \cup \Sigma \cup \{\boxleftarrow, S\}$

R is defined as follows:

(I.) For any $q \in K$, $a \in \Sigma$

if $\delta(q, a) = (p, b)$, $p \in K$ and $b \in \Sigma$
then R contains
$qa \rightarrow pb$

(II.) For any $q \in K$, $a \in \Sigma$

if $\delta(q, a) = (p, \rightarrow)$
then R contains
$qab \rightarrow apb$ for each $b \in \Sigma$
and $qa\boxleftarrow \rightarrow ap\#\boxleftarrow$

etc. Essentially, steps in a derivation will mimic steps in the computation.

Important: Note that the grammar we have constructed simulates the corresponding TM exactly. So it is effectively taking a string that would be accepted by a TM and transforming it to mimic acceptance. This is not really what we want. We want our grammar to **create** the string.

Here's what we need to do it right:

(1) Make sure the TM we're copying accepts in the special configuration (h, ⊠⊳#); that is, before it actually accepts, it erases the tape.
(2) Our grammar will effectively behave like the grammar we just discussed, but will do it backward.
(3) All we need to add are some special beginning and ending rules:

$S \rightarrow \boxtimes\rhd h\#\lhd\boxtimes$
$\boxtimes\rhd s\# \rightarrow e$
$\lhd\boxtimes \rightarrow e$