Unrestricted Grammars

Another model of computation.

Unrestricted Grammars

We saw that Regular and Context-Free Languages are accepted by DTMs. In both cases the DTMs are restricted in their behavior (read-only for RLs and with the tape used to simulate a stack plus simulation of non-determinism for CFLs).

Def.: An unrestricted grammar (or phrase-structured grammar) is a quadruple $(V, \Sigma, R, S)$, where $V$ is a finite set of non-terminal symbols; $\Sigma$ is a finite set of terminal symbols, with $\Sigma \cap V = \emptyset$; $S \in V$ is the starting symbol; $R$ is a finite set of production rules of the form $x \rightarrow y$, for some $x \in (V \cup \Sigma)^*V(V \cup \Sigma)^*$ and $y \in (V \cup \Sigma)^*$. Derivations are as before, except that at each step we may be replacing a substring rather than a single non-terminal.

Theorem: every CFL can be generated by an unrestricted grammar.

Pf. Trivial, since it is clear that CFGs are a subset of unrestricted grammars.

We will look at the equivalence between unrestricted grammars and TMs later; for the time being we will just concentrate on showing that unrestricted grammars are at least as powerful as TMs in the sense that acceptability by TM implies the existence of a generating grammar.

Ex. 4.15: Find a grammar $G$ s.t. $L(G) = \{a^n b^n c^n | n \geq 0\}$.

Soln. Basically, you have to "remember" more than in the case of CFGs (ex.: $\{a^n b^n | n \geq 0\}$). The question is how?

$S \rightarrow aSBC | \varepsilon$
$CB \rightarrow BC$
$AB \rightarrow ab$
$bb \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$

Notice that $S \rightarrow aSBC \varepsilon$ will generate $a^n b^n c^n$, which is not quite what we want, since the $bs$ and $cs$ are not contiguous. The second rule allows us to move all the $bs$ to the left of the $cs$. Once that is done, we replace, starting from the left, all the $bs$ with $ls$, and when that is done, we replace the $Cs$ with $cs$. The rules are designed so that we cannot generate $bs$ and $cs$ out of order.

Ex. 4.16. Find $G$ so that $L(G) = \{a^{2^n} | n \geq 0\}$.

Soln. Let $V = \{S, L, La, R, [, ]\}$ with the rules

$S \rightarrow [Ra] | a$
$Ra \rightarrow aaR$
$R \rightarrow L | L_o$
$La \rightarrow L a$
$L \rightarrow [R$
$L_o \rightarrow [L_a$
$La_o \rightarrow L_a o$

The idea is to construct a grammar that will allow only the construction of strings where the number of $as$ is a power of 2.

Check that this does the job. For example:

$S \rightarrow [Ra] \rightarrow [aaR] \rightarrow [aaL_o] \rightarrow [aaL_o a] \rightarrow [L_o a a] \rightarrow [L_o a] a = aa$

The text has some other examples. The main place we are going to is:
We have just shown that, for any TM, we can construct a grammar instructions of the TM (\(q, x, y\)) and (\(p, x, y'\)), where \(p, q \in Q, a, b \in \Gamma\), and \(x, y, x', y' \in \Gamma\).  

\((q, x, y) \leftrightarrow \{p, x, y'\} \leftrightarrow [xqay] \leftrightarrow [x'pb'y']\)

**Proof.** We keep tracks of words \([xqay]\) \(\in (\Sigma \cup \Gamma')^*\) and configurations \((q, x, y)\) of \(M\), with the rules of the grammar \(G_{xqay}\) simulating the instructions of the TM \(M\). We still have to define the grammar...

1. For \(b(q, a) = (p, b, L), b \in \Gamma\), we define the rule
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
2. For \(b(q, a) = (p, b, R), a \neq \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
3. For \(b(q, a) = (p, b, R), a = \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma, q\alpha \rightarrow b\beta\).

We have just shown that, for any TM, we can construct a grammar that simulates it. Once this is done, the rest follows.

---

**Unrestricted Grammars**

**Theorem 4.19.** For any one-tape DTM \(M = (Q, \Sigma, \Gamma, \delta, \epsilon)\), there exists a grammar \(G_{xqay}(V, \Sigma, R, S)\), with \(V = Q \cup (\Gamma - \Sigma) \cup \{[, ]\}\), such that, for any configurations \((q, x, y)\) and \((p, x, y')\), where \(p, q \in Q, a, b \in \Gamma\), and \(x, y, x', y' \in \Gamma\).

\((q, x, y) \leftrightarrow \{p, x, y'\} \leftrightarrow [xqay] \leftrightarrow [x'pb'y']\)

**Proof.** We keep tracks of words \([xqay]\) \(\in (\Sigma \cup \Gamma')^*\) and configurations \((q, x, y)\) of \(M\), with the rules of the grammar \(G_{xqay}\) simulating the instructions of the TM \(M\). We still have to define the grammar...

1. For \(b(q, a) = (p, b, L), b \in \Gamma\), we define the rule
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
2. For \(b(q, a) = (p, b, R), a \neq \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
3. For \(b(q, a) = (p, b, R), a = \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma, q\alpha \rightarrow b\beta\).

We have just shown that, for any TM, we can construct a grammar that simulates it. Once this is done, the rest follows.

---

**Unrestricted Grammars**

**Theorem 4.20.** If a language \(L \subseteq \Sigma\) is Turing-acceptable, then \(L = L(G)\) for some phrase-structured (= unrestricted) grammar \(G\).

**Note:** acceptance simply means that the TM halts in a configuration \((b, x, y)\). What we want in this case is an indication that the input has been consumed (corresponding to the string being generated from the start symbol). A way to guarantee that would be to require that the output be empty - the configuration \((b, \epsilon)\) - finding some way to match "generation" with "consumption".

**Proof.** Assume \(M\) is as above: whenever it halts, it halts with empty output. Define \(G\) to be the grammar that contains all reversals of rules of \(G_{xqay}\), the grammar of the previous theorem: \(\gamma \rightarrow \gamma_0 \Rightarrow a \rightarrow a\gamma_0 \gamma, \gamma, v\), and the rules
\[ S \rightarrow [B\beta], \quad xB \rightarrow y \] for all \(a \in \Sigma\), where \(L(\beta, \epsilon)\). We keep tracks of words \([xqay]\) \(\in (\Sigma \cup \Gamma')^*\) and configurations \((q, x, y)\) of \(M\), with the rules of the grammar \(G_{xqay}\) simulating the instructions of the TM \(M\). We still have to define the grammar...

1. For \(b(q, a) = (p, b, L), b \in \Gamma\), we define the rule
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
2. For \(b(q, a) = (p, b, R), a \neq \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma\).
3. For \(b(q, a) = (p, b, R), a = \epsilon\), we define the rules
   \(\epsilon q\alpha \rightarrow pb\alpha, \forall \alpha \in \Gamma, q\alpha \rightarrow b\beta\).

We have just shown that, for any TM, we can construct a grammar that simulates it. Once this is done, the rest follows.
Unrestricted Grammars

That it does what needed, follows from the observations:

1. If $M$ accepts $x \in \Sigma^*$, then $[BxB] \Rightarrow^* [BhB]$. We can see this by observing that $[BxB] \Rightarrow (s, BxB)$ and that $[BxB] \Rightarrow^* [BhB] \Rightarrow (s, BhB) \Rightarrow^* (h, BxB)$.

   When we reverse the rules, we have $S \Rightarrow [BhB] \Rightarrow [BxB] \Rightarrow [BxL] \Rightarrow [BxL] \Rightarrow [Lx] \Rightarrow [x]$.

2. Conversely, if $M$ does not accept $x$, there is no derivation $[BhB] \Rightarrow^* [BxB]$, and so no rule reversal can provide a derivation $[BxB] \Rightarrow^* [BhB]$.

Unrestricted Grammars

Theorem. If $L = L(G)$ for an unrestricted grammar $G = (V, \Sigma, R, S)$, then there exists a Turing Machine $M$ that accepts $L$.

Proof (Hopcroft & Ullman, 1979, Thm. 9.3). Let $M$ be a non-deterministic 2-tape machine, whose first tape is the input tape holding the string $w$ we want to check for acceptance. The second tape will hold a sentential form $\alpha$ of $G$. Initialize this sentential form to $S$, the start symbol of $G$.

1. Nondeterministically, select a position $i$ in $\alpha$, $1 \leq i \leq |\alpha|$. You can do this by starting at the left and choosing to select the current position or moving one place to the right.

2. Nondeterministically, select a rule $\beta \rightarrow \gamma$ of $G$.

3. If $\beta$ appears starting at position $i$ of $\alpha$, replace it by $\gamma$. If $i \not\in \beta$, you will need to move characters of $\alpha$ to the left or right.

4. Compare the resulting sentential form to $w$ on tape 1. If they match, accept; if not go to 1.

It should be clear that all possible sentential forms of $G$ (and only sentential forms of $G$) will be generated this way on tape 2. If $w$ can be generated by $G$, it will be obtained through at least one execution path of the nondeterministic TM.

QED

We will look at this equivalence again, on the next set of slides, making some of these ideas more precise.

We now have a second computational formalism - unrestricted grammars - and we have just proven that its computational power is the same as that of the Turing Machine. We will not prove that the Lambda Calculus has the same power, but that can be also proven relatively easily. Three formalisms - all the same. More evidence that the Church-Turing thesis is at least defensible...