1 More Decision Algorithms for Context Free Languages

Next, we describe a more efficient algorithm Eff-$A_{Rec-CFG}$, for determining if a CNF grammar $G$ can generate a string $w$. It runs in time $O(mn^3)$, where $m$ is the number of rules in $G$ and $n = |w|$.

First, we introduce a little notation. $w = w_1w_2 \cdots w_n$, where each $w_i \in T$, the terminal alphabet, for $1 \leq i \leq n$. $w_i^l$ denotes the length $l$ substring of $w$ beginning at $w_i$: $w_i^l = w_i w_{i+1} \cdots w_{i+l-1}$.

Eff-$A_{Rec-CFG}$ uses dynamic programming. Specifically, in turn, for $l = 1, 2, \cdots, n$, it determines, for each variable $A$, whether $A$ can generate $w_i^l$, for each possible value of $i$, i.e. for $1 \leq i \leq n-l+1$. This information suffices, for $G$ can generate $w$ exactly if $S \Rightarrow^* w_1^n$, when $S$ is $G$’s start variable.

For $l = 1$, the test amounts to asking whether $A \rightarrow w_i$ is a rule.

For $l > 1$, the test amounts to the following question:

Is there a rule $A \rightarrow BC$, and a length $k$, with $1 \leq k < l$, such that $B$ generates the length $k$ substring of $w$ beginning at $w_i$, and such that $C$ generates the remainder of $w_i^l$ (i.e. $B \Rightarrow^* w_i^k$ and $C \Rightarrow^* w_i^l - w_i^{l-k}$). Note that the results of the tests involving $B$ and $C$ have have already been computed, so for a single rule and a single value of $k$, this test runs in $O(1)$ time.

Summing the running times over all possible values of $i, k, l$, and all $m$ rules yields the overall running time of $O(mn^3)$.

This shows:

**Lemma 1** The decision procedure for language Rec-CFG runs in time $O(mn^3)$ on input $(M,w)$, where $n = |w|$ and $m$ is the number of rules in $G$.

**Example 2** Inf-CFG = $\{G \mid G$ is a CNF grammar and $L(G)$ is infinite$\}$. 
Claim 3  *Inf-CFG is decidable.*

**Proof:** Note that $L(G)$ is infinite exactly if there is a path in a derivation tree with a repeated variable. The following algorithm, $A_{\text{inf-CFG}}$ identifies the variables that can be repeated in this way; $L(G)$ is infinite exactly if there is at least one such variable. $A_{\text{inf-CFG}}$ proceeds in several steps.

**Step 1.** This step identifies *useful* variables, variables that generate non-empty strings of terminals.

This can be done using a marking procedure. First, $A_{\text{inf-CFG}}$ marks the variables $A$ for which there is a rule of the form $A \rightarrow a$. Then, iteratively, for each rule $A \rightarrow BC$, where both $B$ and $C$ are marked, it also marks $A$, continuing until no additional variables can be marked. The marked variables are exactly the useful variables.

**Step 2.** $A_{\text{inf-CFG}}$ now identifies the *reachable useful* variables, i.e. those useful variables for which there is a derivation $S \Rightarrow^* \sigma A \tau$, where $\sigma, \tau \in V^*$, with $V$ being $G$'s variable set and $S$ its start variable. This is done via the following marking process.

**Step 2.1.** $A_{\text{inf-CFG}}$ marks $S$.

**Step 2.2.** For each unprocessed variable $A$, $A_{\text{inf-CFG}}$ marks all variables on the RHS of a rule with $A$ on the LHS.

When this process terminates, the marked variables are exactly the reachable useful variables.

**Step 3.** Finally, $A_{\text{inf-CFG}}$ identifies the repeating, reachable useful variables, namely the variables that can repeat on a derivation tree path.

To do this, $A_{\text{inf-CFG}}$ uses a procedure analogous to the one used in Step 2: For each reachable useful variable $A$, $A_{\text{inf-CFG}}$ determines the variables reachable from $A$; if this collection includes $A$, then $A$ is repeating. 

\[\]