The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual
Chapter 7, Additional Problem 21

Membership in NP is non-trivial, because we cannot verify a complete derivation (as the number of steps could well exceed a polynomial function of $\log_2 K$). The fact that the language must in fact be regular (any language generated by a context-free grammar on an alphabet of one symbol is easily seen as regular: the nesting property is irrelevant, since all symbols can be arbitrarily permuted) does not directly help either. However, note that the only point of interest is whether a string of the correct length can be generated, using productions chosen from a collection of constant size. Thus, if a derivation is very long, it can only reach that length by using the same productions over and over again; thus we can ask that the certificate be a runlength-encoded description of the sequence of productions used—grouping subsequences and adding to each group a repeating count. Such a certificate is now concise (because of elementary properties of context-free languages) and easily verified with some simple arithmetic.

Now we prove that this problem is NP-hard by transforming *Subset Sum* to it. Given an instance of subset sum with set $S = \{e_1, \ldots, e_n\}$ and size function $s\colon S \to \Re$, we construct an instance of our problem as follows. The idea is to set up a grammar where the start symbol will derive all "subsets" of S; to that effect, there will be one non-terminal, E_i, per element, e_i, of S and the start symbol will derive all strings of length $1, 2, \ldots, n$, composed of these non-terminals, where said non-terminals appear in increasing order of subscript. Each non-terminal, E_i, will then derive a string of as of length $s(e_i)$; however, we cannot write down such productions directly, as they would involve lengths of order $s(e_i)$, which is an exponential function of the input size of $\log_2 s(e_i)$. Thus, we turn to binary representations and write productions that generate 2^j as, for $0 \leq j \leq \lceil \log_2 smax \rceil$ (i.e., binary "digits"), where $smax$ is the maximum size of any element of S; then each E_i will generate a string of nonterminals corresponding to the non-zero digits in its binary representation.

The complete transformation thus produces $2n + 1 + \lceil \log_2 smax \rceil$ non-terminals and the following $3n + 1 + \lceil \log_2 smax \rceil$ productions:

$S \rightarrow F_1 F_2 \ldots F_n$
$F_i \rightarrow E_i | \epsilon$, for all i, $1 \leq i \leq n$
$E_i \rightarrow$corresponding non-zero digits[1]
$< 0 > \rightarrow a$
$< 1 > \rightarrow < 0 > < 0 >$
$< 2 > \rightarrow < 1 > < 1 >$
$\ldots$
$< \lceil \log_2 smax \rceil > \rightarrow < \lceil \log_2 smax \rceil - 1 > < \lceil \log_2 smax \rceil - 1 >$

Now each E_i derives exactly one string, composed of $s(e_i)$ as. The start symbol S derives all 2^n strings corresponding to the various subsets of the set S; thus the start symbol derives a terminal string of length K if and only if there is a subset of the set S, the sizes of whose elements sum up to K.

[1]For instance, if $s(e_i)$ is $21 = 2^4 + 2^2 + 2^0$, then $E_i \rightarrow < 4 > < 2 > < 0 >$.