Homework 2 - Zuzanna Pilat (360670)

1. I will present a circuit solving this problem using divide and conquer approach. Important observation: this is a variation of subset sum problem. As numbers are non-negative we can only consider numbers and sums of numbers $\leqslant n$. My solution has two steps:

   - For each number I create n outputs saying "what sums from 1 to n can be created with this number only". For a single number this demands that for each of n sums we check if the number is exactly this sum. This can be achieved with n parallel AC^0 circuits.
   - On the top of the first step I build a binary tree of ciruits that compute available sums for bigger and bigger sets. Such a circuit has $2n$ inputs - answers for 2 subsets - and produces n outputs - answer for the sum of subsets. Sum i can be created if: (i can be created in 1st set) OR ($i-1$ can be created in 1st set AND 1 can be created in 2nd set) OR .. OR (i can be created in 2nd set). To compute all n outputs we can run in parallel n AC^0 circuits. So the whole step is a $\log n$ deep tree of AC^0 circuits, so it is AC^1.

   At the top of second step the n-th output gives us the answer - can sum n be created with all numbers. The whole circuit is AC^1, so it is NC. It is uniform - description of the circuit can be easily created with a constant number of counters of logarithmic size.

2. First I will show the problem is in NP. We know that the answer to the problem is NO if there is a non-terminal that cannot transitively produce only terminals. We can check that in the following way: if there is a non-terminal that can produce only terminals (or an empty word) in one production, we remove it from right sides of all productions and we remove all productions with this terminal on the left. We repeat

this until there is no rule left (OK) or there is no terminal we can remove (we answer NO). After that we know that once we have seen all non-terminals we can finish producing the word regardless of the mid-product we have. This means we can just guess a derivation up to the point where we have seen all the non-terminals. I will show that this derivation (if it exists) is reasonably short. Let's represent a derivation with a tree. The root is the starting symbol. The children of a symbol are a subset of non-terminals produced from it in one production. Each node that is not a leaf corresponds to a production used. Let's count all the nodes (this will be more than the number of productions we had to guess). Let's denote the number of non-terminals as n. Let's call a node with at least two children a multi-parent. Following facts are true:

- Each leaf corresponds to a different non-terminal (or the tree can be smaller because we don't need duplicates, we can remove them), so there are at most n leaves
- There are at most $n - 1$ multi-parents as each multi-parent increases the number of leaves, so the number of multi-parents and leaves is bounded by $2n - 1$
- The path from a leaf or multi-parent to it's lowest ancestor that is a multi-parent (or root) is of rank at most $n\times$(number of non-terminals on this path that appear exactly once in the tree+1). If there is a path for which this does not hold there is a part of this path of length at least n with no unique symbols on it. This path is so long it must contain two identical non-terminals and we can remove the part from one of them (exclusive) to the other (inclusive) and all non-terminals will still occur in our tree
- As the number of symbols that appear only once in the tree is bounded by n, the number of edges (and nodes) is at most of rank

$n(2n-1+n)$ - each multi-parent and leaf can have n ancestors before the next multi-parent or root for free, unique symbols allow us to add another nn nodes.

This altogether shows that the shortest derivation we have to guess is of polynomial size, so the problem is in NP.

Now I will reduce from SAT problem. For a given instance of SAT:

$x_1, x_2, ..., x_n$ - variables

$c_1, c_2, ..., c_m$ - clauses

let's take the following grammar:

- $P \to X_1 X_2 ... X_n$
- $\forall 1 \leqslant i \leqslant n$ and $j_1, ..., j_{k_i}$ - numbers of clauses containig x_i
  $X_i \to C_{j_1} C_{j_2} ... C_{j_{k_i}}$
- $\forall 1 \leqslant i \leqslant n$ and $j_1, ..., j_{l_i}$ - numbers of clauses containig $\neg x_i$
  $X_i \to C_{j_1} C_{j_2} ... C_{j_{l_i}}$
- $\forall 1 \leqslant i \leqslant m \quad C_i \to c$

This grammar has $1 + 2n + m$ rules, their right sides altogether have length of n+(number of variables in all clauses)+m. It's easy to construct it once we have an instance of SAT. It can be understood as follows: we take all the variables and we choose for each one of them whether it is true or false. Based on the chosen value the variable non-terminal produces non-terminals that represent the clauses this value satisfies. Each clause non-terminal then produces a terminal c. Each derivation in this grammar uses P, all X_i and only those C_i for which the chosen valuation satisfies clause c_i. Therefore all non-terminals are used if and only if the valuation satisfies the whole SAT formula. This proves that the problem is NP-complete.