Exercise 1 (Definite Clause Grammars): (Not relevant for V3M! 5 points)

Consider the following context free grammar \( G = (N, T, S, P) \) with
\[ N = \{ \text{Program, Rule, Body, Atom, Identifier, Variable} \}, \]
\[ T = \{ \ldots, -, \:, , (, ), a, b, c, X, Y, Z \}, \]
\[ S = \text{Program} \quad \text{and} \quad P \quad \text{as follows (\( \epsilon \) represents the empty word).} \]

\[
\begin{align*}
\text{Program} & \rightarrow \epsilon \\
\text{Program} & \rightarrow \text{Rule . Program} \\
\text{Rule} & \rightarrow \text{Atom} \\
\text{Rule} & \rightarrow \text{Atom} :- \text{Body} \\
\text{Body} & \rightarrow \text{Atom} \\
\text{Body} & \rightarrow \text{Variable} \\
\text{Body} & \rightarrow \text{Atom} , \text{Body} \\
\text{Body} & \rightarrow \text{Variable} , \text{Body}
\end{align*}
\]

\[
\begin{align*}
\text{Atom} & \rightarrow \text{Identifier} \\
\text{Identifier} & \rightarrow a \\
\text{Identifier} & \rightarrow b \\
\text{Identifier} & \rightarrow c \\
\text{Variable} & \rightarrow X \\
\text{Variable} & \rightarrow Y \\
\text{Variable} & \rightarrow Z
\end{align*}
\]

Please write a predicate \( \text{program/1} \) such that the query \( \text{?- program(W).} \) is true iff \( W \) is in \( L(G) \). For example, \( a :- b(X). \) is in \( L(G) \). In your program, it would be represented by the list \(["a", ":-", "b", "(", ",", "]\)\. Your Prolog program must not contain the symbol \( \rightarrow \) and must not use predicates for list concatenation. Make use of difference lists as much as possible.

Exercise 2 (Advanced Definite Clause Grammars): (Not relevant for V3M! 3 points)

In definite clause grammars, you can also use arguments for the non-terminal symbols and normal predicates on the right-hand side of a production. This can be useful to check complex restrictions on your language which are not expressable by context free grammars. Consider the following small Prolog program recognizing words built from just one letter \( a \):

\[
\begin{align*}
\text{oneletter} & \rightarrow [\]. \\
\text{oneletter} & \rightarrow [a], \text{oneletter}.
\end{align*}
\]
If we want to modify this program such that the query ?- squareword(W,[]). is true iff W represents a word built from just one letter a which has the length of a square number, we can do the following:

\[
\text{squareword} \rightarrow \text{oneletter}(\text{Length}), \{\text{isSquareNumber}(\text{Length})\}.
\]

\[
\text{oneletter}(0) \rightarrow [].
\]

\[
\text{oneletter}(L) \rightarrow [a], \text{oneletter}(L1), \{L \text{ is } L1 + 1\}.
\]

\[
\text{isSquareNumber}(N) :\text{isSquare}(N,0).
\]

\[
\text{isSquare}(N,C) :\text{if } N \text{ is } C \times C.
\]

\[
\text{isSquare}(N,C) :\text{if } S \text{ is } C \times C, S < N, \text{C1 is } C + 1, \text{isSquare}(N,C1).
\]

We added one argument (the current length of the word) to the non-terminal symbol oneletter. Thus, we are able to calculate this length during the recognition of the word and then check whether this length is a square number. Note that normal predicate calls must be written within curly brackets whenever they are used within a definite clause grammar rule.

This modified program can then be transformed into the following Prolog clauses without definite clause grammar rules:

\[
\text{squareword}(S,R) :\text{oneletter}(\text{Length},S,R), \text{isSquareNumber}(\text{Length}).
\]

\[
\text{oneletter}(0,R,R).
\]

\[
\text{oneletter}(L,[a|S],R) :\text{oneletter}(L1,S,R), L \text{ is } L1 + 1.
\]

\[
\text{isSquareNumber}(N) :\text{isSquare}(N,0).
\]

\[
\text{isSquare}(N,C) :\text{if } N \text{ is } C \times C.
\]

\[
\text{isSquare}(N,C) :\text{if } S \text{ is } C \times C, S < N, \text{C1 is } C + 1, \text{isSquare}(N,C1).
\]

Now consider the following Prolog program recognizing binary numbers:

\[
\text{number} \rightarrow [0].
\]

\[
\text{number} \rightarrow [1].
\]

\[
\text{number} \rightarrow [0], \text{number}.
\]

\[
\text{number} \rightarrow [1], \text{number}.
\]

Please modify this program as described above such that the query ?- number3(N,[]). is true iff N represents a binary number which is divisible by 3.

**Hint:** A binary number is divisible by 3 iff the alternating sum of all its digits is 0. For example, the alternating sum of digits of 101 (representing 5) is 1 − 0 + 1 = 2 ≠ 0 while the alternating sum of digits of 110 (representing 6) is 1 − 1 + 0 = 0. Note that you can of course also add two arguments to the non-terminal symbol number.

**Exercise 3 (Constraint Logic Programming): (Not relevant for V3M and V3B! 4 points)**

Consider the following constraint logic program \( \mathcal{P} \) for the computation of the Fibonacci numbers together with the constraint theory \( CT_{FD} \) from Example 6.1.4 of the lecture (i.e., \( CT_{FD} \) consists of all true formulas on integers):

\[
\text{fib}(0,0).
\]

\[
\text{fib}(1,1).
\]

\[
\text{fib}(N,F) :\text{if } N > 1, N1 \neq N - 1, N2 \neq N - 2, \text{fib}(N1,F1), \text{fib}(N2,F2), F \neq F1 + F2.
\]
Please write down a successful computation of the query $G = \{\neg \text{fib}(2, Z)\}$ w.r.t. $P$ and $CT_{FD}$, i.e., a sequence of configurations of the form $(\text{fib}(2, Z), \text{true}) \vdash_P \ldots \vdash_P (\square, CO)$. As in Example 6.1.15 from the lecture, you may leave out the negations in the queries and simplify your constraints after every computation step. Moreover, please give the computed answer $P[P, CT_{FD}, G]$. 