Exercise 1: Boogie  
Implement the following programs in Boogie.

(a) Implement a procedure with signature \( \text{gcd}(x: \text{int}, y: \text{int}) \text{ returns (div: int)} \)
that takes two (mathematical) integers \( x, y \) and, if they are both not equal to 0,
computes their greatest common divisor \( z \). The algorithm may only make use of
addition and subtraction, but not use multiplication, division or modulo.

(b) Implement a procedure with signature \( \text{prime}(x: \text{int}) \text{ returns (isprime: bool)} \)
that takes an integer \( x \) and, if \( x > 0 \), returns \text{true} if and only if \( x \) is a prime
number.

(c) Implement a procedure with signature \( \text{pow}(x: \text{int}, y: \text{int}) \text{ returns (exp: int)} \)
that takes two integers \( x, y \), and, if \( y \) is greater than 0, returns \( x^y \).

You can use the Boogie interpreter Boogaloo to test your program. A user manual is
available. The Boogie standard does not define division and modulo. In this lecture we
will consider an extension of Boogie where these two operations are defined via the SMT-
LIB semantics for division and modulo (Euclidean division). In the Boogaloo interpreter
the syntax is \text{div} and \text{mod}. In Ultimate the syntax is \text{/} and \text{\%}. In this exercise you may
use the syntax that you like most.

Please submit your Boogie programs electronically (via Email)!

Exercise 2: Satisfiability of FOL Formulas  
Are the following formulas \( \varphi_i \) satisfiable with respect to the theory of integers \( T_\mathbb{Z} \)? If the
formula is satisfiable, give a satisfying assignment.

You may use an SMT solver (e.g. Z3) to solve this task.

\[ \varphi_1 := \forall x, y. \ a \neq 21 \cdot x + 112 \cdot y \]
\[ \varphi_2 := \exists x. (x = 10 \cdot a + b \land a + b = 9 \land \neg \exists y. \ x = 3 \cdot y) \]

5. https://rise4fun.com/Z3
Exercise 3: Boo Grammar

In this exercise you should propose a syntax for the Boo programming language. State a context-free grammar \( G_{\text{Boo}} = (\Sigma_{\text{Boo}}, N_{\text{Boo}}, P_{\text{Boo}}, S_{\text{Boo}}) \) such that a word of the generated language is a program of (your version of) the Boo language.

In the lecture slides we propose the grammar \( G_I = (\Sigma_I, N_I, P_I, S_I) \) for integer expressions, where \( \Sigma_I = \{-, +, *, /, \%, (,), 0, \ldots, 9, a, \ldots, z, A, \ldots Z\} \), \( N_I = \{X_{\text{iexpr}}, X_{\text{num}}, X_{\text{num}'}, X_{\text{var}}, X_{\text{var}'}\} \), \( S_I = X_{\text{iexpr}} \) and the following derivation rules.

\[
P_I = \{ \begin{align*}
X_{\text{iexpr}} & \rightarrow (X_{\text{iexpr}}) \\
X_{\text{iexpr}} & \rightarrow -X_{\text{iexpr}} \\
X_{\text{iexpr}} & \rightarrow X_{\text{iexpr}}+X_{\text{iexpr}}|X_{\text{iexpr}}-X_{\text{iexpr}}|X_{\text{iexpr}}*X_{\text{iexpr}}|X_{\text{iexpr}}/X_{\text{iexpr}}|X_{\text{iexpr}}\%X_{\text{iexpr}} \\
X_{\text{iexpr}} & \rightarrow X_{\text{var}} \\
X_{\text{iexpr}} & \rightarrow X_{\text{num}} \\
X_{\text{num}} & \rightarrow 0X_{\text{num}'}|\ldots|9X_{\text{num}'} \\
X_{\text{num}'} & \rightarrow 0X_{\text{num}'}|\ldots|9X_{\text{num}'}|\varepsilon \\
X_{\text{var}} & \rightarrow aX_{\text{var}'}|zX_{\text{var}'}|AX_{\text{var}'}|\ldots|ZX_{\text{var}'} \\
X_{\text{var}'} & \rightarrow aX_{\text{var}'}|zX_{\text{var}'}|AX_{\text{var}'}|\ldots|ZX_{\text{var}'}|0X_{\text{var}'}|\ldots|9X_{\text{var}'}|\varepsilon
\end{align*} \]

Next, we proposed the grammar \( G_B = (\Sigma_B, N_B, P_B, S_B) \) for Boolean expressions, where \( \Sigma_B = \Sigma_I \cup \{!, &&, ||, ==>, ==, <=, >, <, <=, >=\} \), \( N_B = N_I \cup \{X_{\text{bexpr}}\} \), \( S_B = X_{\text{bexpr}} \) and the following derivation rules.

\[
P_B = \{ \begin{align*}
X_{\text{bexpr}} & \rightarrow (X_{\text{bexpr}}) \\
X_{\text{bexpr}} & \rightarrow !X_{\text{bexpr}} \\
X_{\text{bexpr}} & \rightarrow X_{\text{bexpr}}&&X_{\text{bexpr}}||X_{\text{bexpr}}|X_{\text{bexpr}}==X_{\text{bexpr}} \\
X_{\text{bexpr}} & \rightarrow X_{\text{iexpr}}==X_{\text{iexpr}}|X_{\text{iexpr}}<X_{\text{iexpr}}|X_{\text{iexpr}}>X_{\text{iexpr}}|X_{\text{iexpr}}<=X_{\text{iexpr}}|X_{\text{iexpr}}>=X_{\text{iexpr}} \\
X_{\text{bexpr}} & \rightarrow X_{\text{var}} \\
X_{\text{bexpr}} & \rightarrow \text{true}|\text{false} \} \cup P_I
\end{align*} \]

We propose that you use \( \Sigma_{\text{Boo}} = G_B \cup \{\text{while, if, else,},\ldots,\ldots,:\} \) and your language should have the following properties.

- There should be a while statement, an if-then-else statement and an assignment statement.
- The concatenation of statements should be a statement.
- A program should be a statement and we do not need statements for declaring variables.

Exercise 4: Derivation Tree

Give a derivation tree for the grammar \( G_I \) and the word \( 15 + a + 4 \).