Isabelle/HOL Exercises
Trees, Inductive Data Types

Representation of Propositional Formulae by Polynomials

Let the following data type for propositional formulae be given:

```
datatype form = T | Var nat | And form form | Xor form form
```

Here `T` denotes a formula that is always true, `Var n` denotes a propositional variable, its name given by a natural number, `And f1 f2` denotes the AND combination, and `Xor f1 f2` the XOR (exclusive or) combination of two formulae. A constructor `F` for a formula that is always false is not necessary, since `F` can be expressed by `Xor T T`.

Exercise 1: Define a function

```
consts evalf :: "(nat ⇒ bool) ⇒ form ⇒ bool"
```

that evaluates a formula under a given variable assignment.

Propositional formulae can be represented by so-called *polynomials*. A polynomial is a list of lists of propositional variables, i.e. an element of type `nat list list`. The inner lists (the so-called *monomials*) are interpreted as conjunctive combination of variables, whereas the outer list is interpreted as exclusive-or combination of the inner lists.

Exercise 2: Define two functions

```
consts
  evalm :: "(nat ⇒ bool) ⇒ nat list ⇒ bool"
  evalp :: "(nat ⇒ bool) ⇒ nat list list ⇒ bool"
```

for evaluation of monomials and polynomials under a given variable assignment. In particular think about how empty lists have to be evaluated.

Exercise 3: Define a function

```
consts poly :: "form ⇒ nat list list"
```

that turns a formula into a polynomial. You will need an auxiliary function

```
consts mulpp :: "nat list list ⇒ nat list list ⇒ nat list list"
```

to "multiply" two polynomials, i.e. to compute

$$((v_1^1 \odot \cdots \odot v_{m_1}^1) \oplus \cdots \oplus (v_1^k \odot \cdots \odot v_{m_k}^k)) \odot ((w_1^1 \odot \cdots \odot w_{n_1}^1) \oplus \cdots \oplus (w_1^l \odot \cdots \odot w_{n_l}^l))$$

where $\oplus$ denotes "exclusive or", and $\odot$ denotes "and". This is done using the usual calculation rules for addition and multiplication.

Exercise 4: Now show correctness of your function *poly*:

theorem *poly_correct: "evalf e f = evalp e (poly f)"*

It is useful to prove a similar correctness theorem for *mulpp* first.