Sheet 5 ocaml verification

Funktionale Programmierung (Technische Universität München)

Big Step Proofs
We can use Big Step proof trees to prove that an expression e evaluates to a value v: $e \Rightarrow v$. To construct a Big Step tree, we need to follow the Big Step rules that are defined in the lecture slides (p.340ff).
This proof technique also allows us to prove that the evaluation of an expression terminates: If we can find a value v such that $e \Rightarrow v$, we know that the evaluation of e terminates.

Rewrite Proofs
Another proof technique we can use to prove equality between expressions, are rewrite proofs. We can use a set of rewrite rules given in the slides on pages 357ff to rewrite expressions and prove equalities. These proofs are generally a lot more concise than Big Step proofs.
However, here we need to assume that all expressions of the proof terminate, which is usually given in the exercise.

(Structural) Induction
To prove properties for all possible input values using any of the two techniques, we often use induction on the input parameter. In the case of natural numbers, we can use normal induction. For inputs of recursive types, we can use structural induction on the respective type. In the table below, the proof obligations for different versions of induction are shown. In every case we want to prove a property $P(x)$ for every value x of a type.

Types of Induction

Parameter Type	Base Case	Inductive Case
$\mathbb{N}$	$P(0)$	$\forall n.\ (P(n) \implies P(n+1))$
`'a list`	$P(\texttt{[]})$	$\forall \texttt{x xs}.\ (P(\texttt{xs}) \implies P(\texttt{x::xs}))$
`'a tree`	$P(\texttt{Leaf})$	$\forall \texttt{x l r}.\ ((P(\texttt{l}) \wedge P(\texttt{r})) \implies P(\texttt{Node(l,x,r)}))$

5.1 First Big Steps (*2016, T12.1*)

Let the following MiniOCaml values be defined:

```
let f = fun x -> x + 1 * 2
let f = fun x -> x + 42
let h = 3
```

Now prove the following statements:

$$\texttt{match [1;2;3] with [] -> 0|x::xs -> f x} \Rightarrow 3 \quad (1)$$

$$\texttt{let x=5 in g x+h} \Rightarrow 50 \quad (2)$$

5.2 Termination (*2016, T12.1*)

We define a recursive square function in MiniOCaml:

```
let rec doit a x =
  if x = 0
    then a
  else
    doit (a + 2*x - 1) (x - 1)
let square x = doit 0 x
```

Show that `square x` terminates for all positive values of `x`.

5.3 Verifying the optimization (*2016, T13.2*)

We now want to verify the correctness of our tail recursive version of the factorial function. Recall the definitions of `fac` and `fac_tl`:

```
let rec fac = fun n ->
  match n with
    0 -> 1
  | n -> n * fac (n-1)

let rec fac_aux = fun acc n ->
  match n with
    0 -> acc
  | n -> fac_aux (n * acc) (n-1)

let fac_tl = fac_aux 1
```

Show the correctness of our tail recursive optimization, that is, show that `fac a` = `fac_tl a` for all $a \in \mathbb{N}_0$. You can assume that all expressions terminate.
Hint: generalize the statement before starting the proof.

5.4 Composed Map Functions (*2016, T13.2*)

We can use comp to compose two functions. This enables us to only use a single call of the map function when applying multiple functions to elements of a list.

```
let comp = fun f g x -> f (g x)

let rec map = fun op l ->
  match l with
    [] -> []
  | x::xs -> op x :: map op xs
```

Show the correctness of this optimization: `map (comp f g)` = `comp (map f) (map g)`.
Hint: use extensional equality to obtain a statement that is provable by induction.