Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Roßkopf

SS 24
17. 5. 2024

Functional Data Structures

Exercise Sheet 5

Solve this exercise sheet without *sledgehammer* proofs i.e., *smt*, *metis*, *meson*, or *moura* are forbidden! Furthermore, no *apply* allowed!

Additional note: As the lecture did not cover how to formulate induction proofs using Isar yet, you will not have to set up inductions yourself during this tutorial and homework. Instead the template will provide the necessary steps (except at indicated positions).

Exercise 5.1 Fist Isar Steps

Using Isar, show the following theorem over natural numbers:

theorem
  assumes *"x ≥ (1 :: nat)"*
 shows *"(x + x^2)^2 ≤ 4 * x^4"*

Hint: When phrasing intermediate goals, check your types. While you are not allowed to use *sledgehammer* proofs, it might still be helpful to search for relevant lemmas.

Exercise 5.2 Bounding power-of-two by factorial

Prove that, for all natural numbers $n > 3$, we have $2^n < n!$.

lemma *exp_fact_estimate*: *"n>3 ⟹ (2::nat)^n < fact n"*

Warning! Make sure that your numerals have the right type, otherwise proofs will not work! To check the type of a numeral, hover the mouse over it with pressed CTRL (Mac: CMD) key. Example:

lemma *"2^n ≤ 2^Suc n"*
 apply *auto* **oops**

Leaves the subgoal *2 ^ n ≤ 2 * 2 ^ n*

You will find out that the numeral *2* has type *'a*, for which you do not have any ordering laws. So you have to manually restrict the numeral's type to, e.g., *nat*.

lemma *"(2::nat)^n ≤ 2^Suc n"* **by** *simp*

Exercise 5.3 Sum Squared is Sum of Cubes

- Define a recursive function *sumto* $f\ n = \sum_{i=0\ldots n} f(i)$.
- Show that $(\sum_{i=0\ldots n} i)^2 = \sum_{i=0\ldots n} i^3$.

fun *sumto* :: *"(nat ⇒ nat) ⇒ nat ⇒ nat"*

You may need the following lemma (which requires a very simple induction proof):

lemma *sum_of_naturals*: *"2 * sumto (λx. x) n = n * (n + 1)"*

in order to prove the intended goal:

lemma *"sumto (λx. x) n ^ 2 = sumto (λx. x^3) n"*

Exercise 5.4 Pretty Printing of Binary Trees

(Only if time left, this whole exercise does require induction)

Binary trees can be uniquely pretty-printed by emitting a symbol L for a leaf, and a symbol N for a node. Each N is followed by the pretty-prints of the left and right tree. No additional brackets are required!

datatype *'a tchar = L | N 'a*

fun *pretty* :: *"'a tree ⇒ 'a tchar list"*
value *"pretty (Node (Node Leaf 0 Leaf) (1::nat) (Node Leaf 2 Leaf)) = [N 1, N 0, L, L, N 2, L, L]"*

Show that pretty-printing is actually unique, i.e. no two different trees are pretty-printed the same way. Hint: Auxiliary lemma.

lemma *pretty_unique*: *"pretty t = pretty t' ⟹ t=t'"*

Define a function that checks whether two binary trees have the same structure. The values at the nodes may differ.

fun *bin_tree2* :: *"'a tree ⇒ 'b tree ⇒ bool"*

While this function itself is not very useful, the induction principle generated by the function package is! It allows simultaneous induction over two trees:

print_statement *bin_tree2.induct*

Try to prove the above lemma with that new induction principle.

Note: The following homeworks impose restrictions that are not checked by the submission system, however the tutor will check them manually. If you are not sure how to interpret them, please contact the tutor.

Homework 5.1 More steps with Isar

Submission until Thursday, 23.05.24, 23:59pm.

Prove the following inequality which the tutor fondly remembers from some introductory mathematics course. Use a direct, structured Isar proof.

Hint: Try to prove the statement using pen and paper first and then formalize your proof.

Hint: Even if you are not allowed to use other proof tools in your submission, they can still be useful for exploring. *find_theorems* can also be useful to find missing facts

lemma *nth_root_of_plus_1_bound*:
  fixes x :: *real* **and** n :: *nat*
  assumes *"x≥0"* **and** *"n>0"*
  shows *"root n (1+x) ≤ 1 + x/n"*

In addition to the global restrictions on this sheet, the only proof methods allowed are *simp* and *blast* (So for example *auto*, *fastforce*, *algebra*, ... are also forbidden). Of course all of the Isar syntax (have, show, from, using, ...), forward proofs, instantiation, etc. are still allowed. Each proof step should therefore be of form *by* (*simp <optional modifiers>*) or *by blast*.

In your proof you will probably need the *Bernoulli_inequality*:
$-\ 1 \leq x \Longrightarrow 1 + real\ n * x \leq (1 + x)^n$
which is already proven for you in the standard library.

Homework 5.2 Binet

Submission until Thursday, 23.05.24, 23:59pm.

The Fibonacci numbers can be computed by the following function:

fun *fib* :: *"nat ⇒ nat"* **where**
  "fib 0 = 0"
| *"fib (Suc 0) = 1"*
| *"fib (Suc (Suc n)) = fib (Suc n) + fib n"*

Prove the following closed-form expression for them, well-known as Binet's formula:

lemma *binet*: *"fib n = (Φ^n − Ψ^n) / sqrt 5"*

Here, Φ, Ψ are defined as $\Phi \equiv (1 + sqrt\ 5)\ /\ 2$ and $\Psi \equiv (1 - sqrt\ 5)\ /\ 2$.

Once again, it might be helpful, to think about how to do the proof using pen and paper first.

For readability, each step in that chain must be very simple: it must use *simp* with at most one deleted fact and one additional fact (i.e. global lemma, assumption, IH) or *field_simps*.

The lemmas Φ*_square*: $\Phi^2 = 1 + \Phi$ and Ψ*_square*: $\Psi^2 = 1 + \Psi$ may be helpful.

Homework 5.3 Trisecting lists

Submission until Thursday, 23.05.24, 23:59pm.

For this exercise the restrictions from the previous exercises no longer apply. You are still forbidden *sledgehammer* proofs and *apply*, as defined at the top of the sheet.

Show that each list can be split into three parts, with the first two of them having exactly a third of the length of the original list, and the third being the leftover elements.

lemma *trisecting*:
"∃ xs ys zs . length xs = length as div 3 ∧ length ys = length as div 3 ∧ as = xs @ ys @ zs"