Homework

An exercise that we will do for each of HOL Light, Mizar, and Coq, is to prove the statement:

$$\forall n.(\sum_{i=0}^{n} i)^2 = \sum_{i=0}^{n} i^3 \tag{1}$$

For instance, for $n = 3$ we have $(0+1+2+3)^2 = 6^2 = 36 = 0+1+8+27 = 0^3+1^3+2^3+3^3$. The goal is to make you via this exercise acquainted with the basics of each of HOL Light, Mizar, and Coq, so you can make your own decision later during the course which one you prefer to do the final assignment with.

1. Read the HOL Light tutorial up to and including Section 8. In particular, work through the example in Section 8.2 step by step (this result is used in our proof of (1)).

2. Replay the proof `ADD_0` of the fact that $\forall m.m + 0 = m$ in `arith.ml`, step by step.

```
# g `!m. m + 0 = m`
  ;;
val it : goalstack = 1 subgoal (1 total)

`!m. m + 0 = m`

# e (INDUCT_TAC THEN ASM_REWRITE_TAC[ADD]);;
val it : goalstack = No subgoals

# let rechtsnullinks = top_thm();;
val rechtsnullinks : thm = |- !m. m + 0 = m
```

   Then try to prove the fact that $\forall m.m = m + 0$ in the same way.

```
# g `!m. m = m + 0`;;
val it : goalstack = 1 subgoal (1 total)

`!m. m = m + 0`

# e (INDUCT_TAC THEN ASM_REWRITE_TAC[ADD]);;
val it : goalstack = 1 subgoal (1 total)

  0 [`m = m + 0`]

`SUC m = SUC (m + 0)`
```

   What is the problem?

   The assumption is used to rewrite only if this is does not result in non-termination (if the left-hand side is not embedded in the right-hand side). Here it would, since any m can be rewritten to $m + 0$.

   What solutions can you think of to work around the problem?

```
# b();;
val it : goalstack = 1 subgoal (1 total)

`!m. m = m + 0`

# e (REWRITE_TAC[rechtsnullinks]);;
val it : goalstack = No subgoals
```

That's cheating in the sense that we have used the earlier theorem. However, we can also do without it, by explicitly constructing the symmetric version of the assumption.

```
  # b();;
val it : goalstack = 1 subgoal (1 total)

`!m. m = m + 0`

# e (INDUCT_TAC);;
val it : goalstack = 2 subgoals (2 total)

  0 [`m = m + 0`]

`SUC m = SUC m + 0`

`0 = 0 + 0`

# e (REWRITE_TAC[ADD]);;
val it : goalstack = 1 subgoal (1 total)

  0 [`m = m + 0`]

`SUC m = SUC m + 0`

# e (ASSUME_TAC (SYM (ASSUME (`m = m + 0`))));;
val it : goalstack = 1 subgoal (1 total)

  0 [`m = m + 0`]
  1 [`m + 0 = m`]

`SUC m = SUC m + 0`

# e (ASM_REWRITE_TAC[ADD]);;
val it : goalstack = No subgoals

#  let rechtsnulrechts = top_thm();;
val rechtsnulrechts : thm = |- !m. m = m + 0
```

And there are several other ways, as discussed in the PS.

3. During the PS last week we have given a step-by-step paper-proof of (1), splitting it into several lemmas (most of them simple algebraic laws on addition and multiplication like the one of the previous exercise). Write down these lemmas, their dependencies, and formulate them in HOL Light (try to come up with systematic names for them).

$$\forall nm.(n+m)^2 = n^2 + 2*n*m + m^2 \tag{2}$$

$$\forall n.2*(\sum_{i=0}^{n} i) = n*(n+1) \tag{3}$$

$$\forall n.(\sum_{i=0}^{n} i)^2 = \sum_{i=0}^{n} i^3 \tag{4}$$

with (4) depending on (2) and (3).

4. Prove (1) by successively proving your lemmas of the previous item in HOL Light.
   (Although powerful tactics can be used, try using only simple ones in the beginning, to learn how to manipulate statements/theorems.)

   first, the notable product[1]*, by computation*

```
g `!n m. (n+m) EXP 2 = n EXP 2 + (2 * n) * m + m EXP 2`;;
e (REPEAT GEN_TAC);;
e (ARITH_TAC);;
let merkwaardigprodukt = top_thm();;
```

   next, Gauss' summation, by induction and distributivity

```
g `!n. 2 * nsum (1..n) (\i. i) = n * (n+1)`;;
e INDUCT_TAC;;
e (SIMP_TAC[NSUM_CLAUSES_NUMSEG]);;
e (ARITH_TAC);;
e (SIMP_TAC[NSUM_CLAUSES_NUMSEG]);;
e (ASSUME_TAC (ARITH_RULE (`!n.1 <= SUC n`)));;
e (SIMP_TAC[ASSUME `!n. 1 <= SUC n`]);;
e (REWRITE_TAC[LEFT_ADD_DISTRIB]);;
e (ASM_REWRITE_TAC[]);;
e (ARITH_TAC);;
let gausssom = top_thm();;
```

   finally, the main result by induction and cancellation, using the previous results

```
g `!n. (nsum(1..n) (\i.i)) EXP 2 = nsum(1..n) (\i.i EXP 3)`;;
e INDUCT_TAC;;
e (SIMP_TAC[NSUM_CLAUSES_NUMSEG]);;
e (ARITH_TAC);;
e (SIMP_TAC[NSUM_CLAUSES_NUMSEG]);;
e (ASSUME_TAC (ARITH_RULE (`!n.1 <= SUC n`)));;
e (SIMP_TAC[ASSUME `!n. 1 <= SUC n`]);;
e (REWRITE_TAC[merkwaardigprodukt]);;
e (ASM_REWRITE_TAC[]);;
e (SIMP_TAC[EQ_ADD_LCANCEL]);;
e (REWRITE_TAC[gausssom]);;
e (ARITH_TAC);;
let squarescubes = top_thm();;
```

   Note that the tutorial does in fact already contain a short proof of this result, or rather where Gauss' summation has been expanded already.

5. In the second part of the PS a proof of strong normalisation (termination) of β-reduction for simply type λ-calculus (due to Tait) was given, see the note on the next page.[2]

   - Spell out the details of why the identity substitution is strongly computable.
     A substitution σ is strongly computable, if for each variable x, the λ-term $\sigma(x)$ is strongly computable. Hence to show that the identity substitution is strongly computable, it suffices to show that every variable is. By definition, a variable x is strongly computable is for every vector $\vec{N}$ of strongly computable terms $x\vec{N}$ is strongly normalising. It is easy to see (by inspection of the shape of the β-rule) that if $x\vec{N} \to_\beta M$, then $M = x\vec{M}$ for some vector $\vec{M}$ of the same length as $\vec{N}$ such that for each i, $N_i \to_\beta^= M_i$. Hence, by the Pigeon Hole Principle, if $x\vec{N}$ were not *strongly normalising, then one of the N_i would be not strongly normalising. But since by assumption each N_i is strongly computable, we have in particular that each N_i is strongly normalising (instantiate to the empty vector of arguments).*

[1] with parentheses such that it is easily used later on

[2] Beware that in the note $\vec{R}$ is used both as a vector of terms (when stand-alone $\vec{R} = R_1, \ldots, R_n$) and as a repeated application (when applied $M\vec{R} = (\ldots(MR_1)\ldots R_n)$, i.e. a sries of applications).

- Explain in what way the predicate SC is inductively defined.
  (How would its definition fail to be inductive for the untyped λ-calculus?)

  It is defined by induction on types, first defining strong computability for (all terms of) base types (where it coincides with strong normalisation), and then extending it to functional types, by parametrising over the (terms of the) strongly computable types of their arguments. Formally, one may define, for each type A, the set SC_A of strongly computable simply typed λ-terms of type A *by induction on A, as in the note. In the base case, A is a base type and since a term of base type cannot be applied to any term, the vector of arguments is empty, so then strong computabiliy just coincides with strong normalisation. In the step case, A is a function type and can be applied only to vectors of arguments that are of* smaller *type, hence for these strong computability is already defined.*

  *Note: Untyped λ-terms can viewed of being of a type $T \equiv T \to T$. That is, even if we have a term M of base type T, it still can be applied to other terms by using the equivalence to see that then M is also of (*not smaller*) type $T \to T$. Hence for untyped terms strong computability cannot be defined by induction on types as above.*