International Mathematical Olympiad 2019

Manuel Eberl

March 17, 2025

Abstract

This entry contains formalisations of the answers to three of the six problem of the International Mathematical Olympiad 2019, namely Q1, Q4, and Q5. The reason why these problems were chosen is that they are particularly amenable to formalisation: they can be solved with minimal use of libraries. The remaining three concern geometry and graph theory, which, in the author's opinion, are more difficult to formalise resp. require a more complex library.

Contents

1 Q1

theory *IMO2019-Q1*
  imports *Main*
begin

Consider a function $f : \mathbb{Z} \to \mathbb{Z}$ that fulfils the functional equation $f(2a) + 2f(b) = f(f(a+b))$ for all $a, b \in \mathbb{Z}$.

Then f is either identically 0 or of the form $f(x) = 2x + c$ for some constant $c \in \mathbb{Z}$.

context
  fixes $f :: int \Rightarrow int$ **and** $m :: int$
  assumes *f-eq*: $f\ (2 * a) + 2 * f\ b = f\ (f\ (a + b))$
  defines $m \equiv (f\ 0 - f\ (-2))\ div\ 2$
begin

We first show that f is affine with slope $(f(0) - f(-2)) / 2$. This follows from plugging in $(0, b)$ and $(-1, b + 1)$ into the functional equation.

lemma *f-eq'*: $f\ x = m * x + f\ 0$
$\langle proof \rangle$

This version is better for the simplifier because it prevents it from looping.

lemma *f-eq'-aux* [*simp*]: $NO\text{-}MATCH\ 0\ x \Longrightarrow f\ x = m * x + f\ 0$
  $\langle proof \rangle$

Plugging in $(0, 0)$ and $(0, 1)$.

lemma *f-classification*: $(\forall x.\ f\ x = 0) \vee (\forall x.\ f\ x = 2 * x + f\ 0)$
  $\langle proof \rangle$

end

It is now easy to derive the full characterisation of the functions we considered:

theorem
  fixes $f :: int \Rightarrow int$
  shows $(\forall a\ b.\ f\ (2 * a) + 2 * f\ b = f\ (f\ (a + b))) \longleftrightarrow$
      $(\forall x.\ f\ x = 0) \vee (\forall x.\ f\ x = 2 * x + f\ 0)$ (**is** $?lhs \longleftrightarrow ?rhs$)
$\langle proof \rangle$

end

2 Q4

theory *IMO2019-Q4*
  imports *Prime-Distribution-Elementary.More-Dirichlet-Misc*
begin

Find all pairs (k, n) of positive integers such that $k! = \prod_{i=0}^{n-1}(2^n - 2^i)$.

2.1 Auxiliary facts

lemma *Sigma-insert*: $Sigma\ (insert\ x\ A)\ f = (\lambda y.\ (x,\ y))\ `\ f\ x \cup Sigma\ A\ f$
⟨*proof*⟩

lemma *atLeastAtMost-nat-numeral*:
$\{(m::nat)..numeral\ k\} =$
$(if\ m \leq numeral\ k\ then\ insert\ (numeral\ k)\ \{m..pred\text{-}numeral\ k\}\ else\ \{\})$
⟨*proof*⟩

lemma *greaterThanAtMost-nat-numeral*:
$\{(m::nat)<..numeral\ k\} =$
$(if\ m < numeral\ k\ then\ insert\ (numeral\ k)\ \{m<..pred\text{-}numeral\ k\}\ else\ \{\})$
⟨*proof*⟩

lemma *fact-ge-power*:
fixes $c :: nat$
assumes $fact\ n0 \geq c\ \hat{}\ n0\ c \leq n0 + 1$
assumes $n \geq n0$
shows $fact\ n \geq c\ \hat{}\ n$
⟨*proof*⟩

lemma *prime-multiplicity-prime*:
fixes $p\ q :: 'a :: factorial\text{-}semiring$
assumes $prime\ p\ prime\ q$
shows $multiplicity\ p\ q = (if\ p = q\ then\ 1\ else\ 0)$
⟨*proof*⟩

We use Legendre's identity from the library. One could easily prove the property in question without the library, but it probably still saves a few lines.

legendre-aux (related to Legendre's identity) is the multiplicity of a given prime in the prime factorisation of $n!$.

lemma *multiplicity-prime-fact*:
fixes $p :: nat$
assumes $prime\ p$
shows $multiplicity\ p\ (fact\ n) = legendre\text{-}aux\ n\ p$
⟨*proof*⟩

The following are simple and trivial lower and upper bounds for *legendre-aux*:

lemma *legendre-aux-ge*:
assumes $prime\ p\ k \geq 1$
shows $legendre\text{-}aux\ k\ p \geq nat\ \lfloor k\ /\ p \rfloor$
⟨*proof*⟩

lemma *legendre-aux-less*:
assumes $prime\ p\ k \geq 1$
shows $legendre\text{-}aux\ k\ p < k\ /\ (p - 1)$

⟨*proof*⟩

2.2 Main result

Now we move on to the main result: We fix two numbers n and k with the property in question and derive facts from that.

The triangle number $T = n(n+1)/2$ is of particular importance here, so we introduce an abbreviation for it.

context
 fixes *k n* :: *nat* **and** *rhs T* :: *nat*
 defines *rhs* ≡ ($\prod$ *i*<*n*. *2* ^ *n* − *2* ^ *i*)
 defines *T* ≡ (*n* ∗ (*n* − *1*)) *div 2*
 assumes *pos*: *k* > *0 n* > *0*
 assumes *k-n*: *fact k* = *rhs*
begin

We can rewrite the right-hand side into a more convenient form:

lemma *rhs-altdef*: *rhs* = *2* ^ *T* ∗ ($\prod$ *i*=*1*..*n*. *2* ^ *i* − *1*)
⟨*proof*⟩

The multiplicity of 2 in the prime factorisation of the right-hand side is precisely *T*.

lemma *multiplicity-2-rhs* [*simp*]: *multiplicity 2 rhs* = *T*
⟨*proof*⟩

From Legendre's identities and the associated bounds, it can easily be seen that ⌊*k*/*2*⌋ ≤ *T* < *k*:

lemma *k-gt-T*: *k* > *T*
⟨*proof*⟩

lemma *T-ge-half-k*: *T* ≥ *k div 2*
⟨*proof*⟩

It can also be seen fairly easily that the right-hand side is strictly smaller than 2^{n^2}:

lemma *rhs-less*: *rhs* < *2* ^ n^2
⟨*proof*⟩

It is clear that $2^{n^2} \leq 8^T$ and that $8^T < T!$ if T is sufficiently big. In this case, 'sufficiently big' means *T* ≥ *20* and thereby *n* ≥ *7*. We can therefore conclude that n must be less than 7.

lemma *n-less-7*: *n* < *7*
⟨*proof*⟩

We now only have 6 values for n to check. Together with the bounds that we obtained on k, this only leaves a few combinations of n and k to check, and we do precisely that and find that $n = k = 1$ and $n = 2$, $k = 3$ are the only possible combinations.

lemma *n-k-in-set*: $(n, k) \in \{(1, 1), (2, 3)\}$
⟨*proof*⟩

end

Using this, deriving the final result is now trivial:

theorem $\{(n, k).\ n > 0 \wedge k > 0 \wedge \mathit{fact}\ k = (\prod i{<}n.\ 2 \mathbin{\hat{}} n - 2 \mathbin{\hat{}} i :: \mathit{nat})\} = \{(1, 1), (2, 3)\}$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

end

3 Q5

theory *IMO2019-Q5*
imports *Complex-Main*
begin

Given a sequence $(c_1, \ldots, c_n)$ of coins, each of which can be heads (H) or tails (T), Harry performs the following process: Let k be the number of coins that show H. If $k > 0$, flip the k-th coin and repeat the process. Otherwise, stop.

What is the average number of steps that this process takes, averaged over all 2^n coin sequences of length n?

3.1 Definition

We represent coins as Booleans, where *True* indicates H and *False* indicates T. Coin sequences are then simply lists of Booleans.

The following function flips the i-th coin in the sequence (in Isabelle, the convention is that the first list element is indexed with 0).

definition *flip* :: *bool list* ⇒ *nat* ⇒ *bool list* **where**
flip xs i = *xs*[*i* := ¬*xs* ! *i*]

lemma *flip-Cons-pos* [*simp*]: $n > 0 \Longrightarrow \mathit{flip}\ (x \# \mathit{xs})\ n = x \# \mathit{flip}\ \mathit{xs}\ (n - 1)$
⟨*proof*⟩

lemma *flip-Cons-0* [*simp*]: $\mathit{flip}\ (x \# \mathit{xs})\ 0 = (\neg x) \# \mathit{xs}$
⟨*proof*⟩

lemma *flip-append1* [*simp*]: $n < \mathit{length}\ \mathit{xs} \Longrightarrow \mathit{flip}\ (\mathit{xs}\ @\ \mathit{ys})\ n = \mathit{flip}\ \mathit{xs}\ n\ @\ \mathit{ys}$
and *flip-append2* [*simp*]: $n \geq \mathit{length}\ \mathit{xs} \Longrightarrow n < \mathit{length}\ \mathit{xs} + \mathit{length}\ \mathit{ys} \Longrightarrow$
$\mathit{flip}\ (\mathit{xs}\ @\ \mathit{ys})\ n = \mathit{xs}\ @\ \mathit{flip}\ \mathit{ys}\ (n - \mathit{length}\ \mathit{xs})$

⟨*proof*⟩

lemma *length-flip* [*simp*]: *length* (*flip xs i*) = *length xs*
⟨*proof*⟩

The following function computes the number of H in a coin sequence.

definition *heads* :: *bool list* ⇒ *nat* **where** *heads xs* = *length* (*filter id xs*)

lemma *heads-True* [*simp*]: *heads* (*True* # *xs*) = *1* + *heads xs*
and *heads-False* [*simp*]: *heads* (*False* # *xs*) = *heads xs*
and *heads-append* [*simp*]: *heads* (*xs* @ *ys*) = *heads xs* + *heads ys*
and *heads-Nil* [*simp*]: *heads* [] = *0*
⟨*proof*⟩

lemma *heads-Cons*: *heads* (*x* # *xs*) = (*if x then heads xs* + *1 else heads xs*)
⟨*proof*⟩

lemma *heads-pos*: *True* ∈ *set xs* ⟹ *heads xs* > *0*
⟨*proof*⟩

lemma *heads-eq-0* [*simp*]: *True* ∉ *set xs* ⟹ *heads xs* = *0*
⟨*proof*⟩

lemma *heads-eq-0-iff* [*simp*]: *heads xs* = *0* ⟷ *True* ∉ *set xs*
⟨*proof*⟩

lemma *heads-pos-iff* [*simp*]: *heads xs* > *0* ⟷ *True* ∈ *set xs*
⟨*proof*⟩

lemma *heads-le-length*: *heads xs* ≤ *length xs*
⟨*proof*⟩

The following function performs a single step of Harry's process.

definition *harry-step* :: *bool list* ⇒ *bool list* **where**
harry-step xs = *flip xs* (*heads xs* − *1*)

lemma *length-harry-step* [*simp*]: *length* (*harry-step xs*) = *length xs*
⟨*proof*⟩

The following is the measure function for Harry's process, i.e. how many steps the process takes to terminate starting from the given sequence. We define it like this now and prove the correctness later.

function *harry-meas* **where**
harry-meas xs =
(*if xs* = [] *then 0*
else if hd xs then 1 + *harry-meas* (*tl xs*)
else if ¬*last xs then harry-meas* (*butlast xs*)
else let n = *length xs in harry-meas* (*take* (*n* − *2*) (*tl xs*)) + *2* ∗ *n* − *1*)

⟨*proof*⟩
termination ⟨*proof*⟩

lemmas [*simp del*] = *harry-meas.simps*

We now prove some simple properties of *harry-meas* and *harry-step*.

We prove a more convenient case distinction rule for lists that allows us to distinguish between lists starting with *True*, ending with *False*, and starting with *False* and ending with *True*.

lemma *head-last-cases* [*case-names Nil True False False-True*]:
assumes $xs = [] \Longrightarrow P$
assumes $\bigwedge ys.\ xs = True \# ys \Longrightarrow P$ $\bigwedge ys.\ xs = ys @ [False] \Longrightarrow P$
$\bigwedge ys.\ xs = False \# ys @ [True] \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *harry-meas-Nil* [*simp*]: *harry-meas* [] = *0*
⟨*proof*⟩

lemma *harry-meas-True-start* [*simp*]: *harry-meas* (*True* # *xs*) = *1* + *harry-meas* *xs*
⟨*proof*⟩

lemma *harry-meas-False-end* [*simp*]: *harry-meas* (*xs* @ [*False*]) = *harry-meas xs*
⟨*proof*⟩

lemma *harry-meas-False-True*: *harry-meas* (*False* # *xs* @ [*True*]) = *harry-meas* *xs* + *2* ∗ *length xs* + *3*
⟨*proof*⟩

lemma *harry-meas-eq-0* [*simp*]:
assumes *True* ∉ *set xs*
shows *harry-meas xs* = *0*
⟨*proof*⟩

If the sequence starts with *H*, the process runs on the remaining sequence until it terminates and then flips this *H* in another single step.

lemma *harry-step-True-start* [*simp*]:
harry-step (*True* # *xs*) = (*if True* ∈ *set xs then True* # *harry-step xs else False* # *xs*)
⟨*proof*⟩

If the sequence ends in *T*, the process simply runs on the remaining sequence as if it were not present.

lemma *harry-step-False-end* [*simp*]:
assumes *True* ∈ *set xs*
shows *harry-step* (*xs* @ [*False*]) = *harry-step xs* @ [*False*]

⟨*proof*⟩

If the sequence starts with T and ends with H, the process runs on the remaining sequence inbetween as if these two were not present, eventually leaving a sequence that consists entirely if T except for a single final H.

lemma *harry-step-False-True*:
assumes *True* ∈ *set xs*
shows *harry-step* (*False* # *xs* @ [*True*]) = *False* # *harry-step xs* @ [*True*]
⟨*proof*⟩

That sequence consisting only of T except for a single final H is then turned into an all-T sequence in $2n+1$ steps.

lemma *harry-meas-Falses-True* [*simp*]: *harry-meas* (*replicate n False* @ [*True*]) = $2 * n + 1$
⟨*proof*⟩

lemma *harry-step-Falses-True* [*simp*]:
$n > 0 \Longrightarrow$ *harry-step* (*replicate n False* @ [*True*]) = *True* # *replicate* $(n - 1)$ *False* @ [*True*]
⟨*proof*⟩

3.2 Correctness of the measure

We will now show that *harry-meas* indeed counts the length of the process. As a first step, we will show that if there is a H in a sequence, applying a single step decreases the measure by one.

lemma *harry-meas-step-aux*:
assumes *True* ∈ *set xs*
shows *harry-meas xs* = *Suc* (*harry-meas* (*harry-step xs*))
⟨*proof*⟩

lemma *harry-meas-step*: *True* ∈ *set xs* ⟹ *harry-meas* (*harry-step xs*) = *harry-meas xs* − *1*
⟨*proof*⟩

Next, we show that the measure is zero if and only if there is no H left in the sequence.

lemma *harry-meas-eq-0-iff* [*simp*]: *harry-meas xs* = *0* ⟷ *True* ∉ *set xs*
⟨*proof*⟩

It follows by induction that if the measure of a sequence is n, then iterating the step less than n times yields a sequence with at least one H in it, but iterating it exactly n times yields a sequence that contains no more H.

lemma *True-in-funpow-harry-step*:
assumes $n <$ *harry-meas xs*
shows *True* ∈ *set* ((*harry-step* ^^ *n*) *xs*)
⟨*proof*⟩

lemma *True-notin-funpow-harry-step*: $True \notin set\ ((harry\text{-}step\ \widehat{}\widehat{}\ harry\text{-}meas\ xs)\ xs)$
⟨*proof*⟩

This shows that the measure is indeed the correct one: It is the smallest number such that iterating Harry's step that often yields a sequence with no heads in it.

theorem $harry\text{-}meas\ xs = (LEAST\ n.\ True \notin set\ ((harry\text{-}step\ \widehat{}\widehat{}\ n)\ xs))$
⟨*proof*⟩

3.3 Average-case analysis

The set of all coin sequences of a given length.

definition *seqs* **where** $seqs\ n = \{xs :: bool\ list\ .\ length\ xs = n\}$

lemma *length-seqs* [*dest*]: $xs \in seqs\ n \Longrightarrow length\ xs = n$
⟨*proof*⟩

lemma *seqs-0* [*simp*]: $seqs\ 0 = \{[]\}$
⟨*proof*⟩

The coin sequences of length $n + 1$ are simply what is obtained by appending either H or T to each coin sequence of length n.

lemma *seqs-Suc*: $seqs\ (Suc\ n) = (\lambda xs.\ True\ \#\ xs)\ \text{`}\ seqs\ n \cup (\lambda xs.\ False\ \#\ xs)\ \text{`}\ seqs\ n$
⟨*proof*⟩

The set of coin sequences of length n is invariant under reversal.

lemma *seqs-rev* [*simp*]: $rev\ \text{`}\ seqs\ n = seqs\ n$
⟨*proof*⟩

Hence we get a similar decomposition theorem that appends at the end.

lemma *seqs-Suc'*: $seqs\ (Suc\ n) = (\lambda xs.\ xs\ @\ [True])\ \text{`}\ seqs\ n \cup (\lambda xs.\ xs\ @\ [False])\ \text{`}\ seqs\ n$
⟨*proof*⟩

lemma *finite-seqs* [*intro*]: $finite\ (seqs\ n)$
⟨*proof*⟩

lemma *card-seqs* [*simp*]: $card\ (seqs\ n) = 2\ \widehat{}\ n$
⟨*proof*⟩

lemmas *seqs-code* [*code*] = *seqs-0 seqs-Suc*

The sum of the measures over all possible coin sequences of a given length (defined as a recurrence relation; correctness proven later).

fun $harry\text{-}sum :: nat \Rightarrow nat$ **where**
$harry\text{-}sum\ 0 = 0$
$|\ harry\text{-}sum\ (Suc\ 0) = 1$
$|\ harry\text{-}sum\ (Suc\ (Suc\ n)) = 2 * harry\text{-}sum\ (Suc\ n) + (2 * n + 4) * 2\ \hat{}\ n$

lemma *Suc-Suc-induct*: $P\ 0 \Longrightarrow P\ (Suc\ 0) \Longrightarrow (\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n) \Longrightarrow P\ (Suc\ (Suc\ n))) \Longrightarrow P\ n$
⟨*proof*⟩

The recurrence relation really does describe the sum over all measures:

lemma *harry-sum-correct*: $harry\text{-}sum\ n = sum\ harry\text{-}meas\ (seqs\ n)$
⟨*proof*⟩

lemma *harry-sum-closed-form-aux*: $4 * harry\text{-}sum\ n = n * (n + 1) * 2\ \hat{}\ n$
⟨*proof*⟩

Solving the recurrence gives us the following solution:

theorem *harry-sum-closed-form*: $harry\text{-}sum\ n = n * (n + 1) * 2\ \hat{}\ n\ div\ 4$
⟨*proof*⟩

The average is now a simple consequence:

definition *harry-avg* **where** $harry\text{-}avg\ n = harry\text{-}sum\ n\ /\ card\ (seqs\ n)$

corollary $harry\text{-}avg\ n = n * (n + 1)\ /\ 4$
⟨*proof*⟩

end

References

[1] 60th International Mathematical Olympiad. https://www.imo2019.uk/wp-content/uploads/2018/07/solutions-r856.pdf. 11th–22nd July 2019.