

# International Mathematical Olympiad 2019

Manuel Eberl

February 6, 2026

## Abstract

This entry contains formalisations of the answers to three of the six problems 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

<b>1</b>	<b>Q1</b>	<b>2</b>
<b>2</b>	<b>Q4</b>	<b>3</b>
2.1	Auxiliary facts . . . . .	3
2.2	Main result . . . . .	5
<b>3</b>	<b>Q5</b>	<b>8</b>
3.1	Definition . . . . .	9
3.2	Correctness of the measure . . . . .	12
3.3	Average-case analysis . . . . .	15

# 1 Q1

```
theory IMO2019-Q1
  imports Main
begin
```

Consider a function  $f : \mathbb{Z} \rightarrow \mathbb{Z}$  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 ⇒ int and m :: int
  assumes f-eq: f (2 * a) + 2 * f b = f (f (a + b))
  defines m ≡ (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
proof -
  have rec: f (b + 1) = f b + m for b
    using f-eq[of 0 b] f-eq[of -1 b + 1] by (simp add: m-def)
  moreover have f (b - 1) = f b - m for b
    using rec[of b - 1] by simp
  ultimately show ?thesis
    by (induction x rule: int-induct[of - 0]) (auto simp: algebra-simps)
qed
```

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

```
lemma f-eq'-aux [simp]: NO-MATCH 0 x ⇒ f x = m * x + f 0
  by (rule f-eq')
```

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

```
lemma f-classification: (∀ x. f x = 0) ∨ (∀ x. f x = 2 * x + f 0)
  using f-eq[of 0 0] f-eq[of 0 1] by auto
```

end

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

```
theorem
  fixes f :: int ⇒ int
  shows (∀ a b. f (2 * a) + 2 * f b = f (f (a + b))) ↔
    (∀ x. f x = 0) ∨ (∀ x. f x = 2 * x + f 0) (is ?lhs ↔ ?rhs)
proof
  assume ?lhs
  thus ?rhs using f-classification[of f] by blast
```

```

next
  assume ?rhs
  thus ?lhs by (smt (verit, ccfv-threshold) mult-2)
qed

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 = (λy. (x, y)) ' f x ∪ Sigma A f
  by auto

```

```

lemma atLeastAtMost-nat-numeral:
  {(m::nat)..numeral k} =
    (if m ≤ numeral k then insert (numeral k) {m..pred-numeral k} else {})
  by (auto simp: numeral-eq-Suc)

```

```

lemma greaterThanAtMost-nat-numeral:
  {(m::nat)<..numeral k} =
    (if m < numeral k then insert (numeral k) {m<..pred-numeral k} else {})
  by (auto simp: numeral-eq-Suc)

```

```

lemma fact-ge-power:
  fixes c :: nat
  assumes fact n0 ≥ c ^ n0 c ≤ n0 + 1
  assumes n ≥ n0
  shows fact n ≥ c ^ n
  using assms(3,1,2)
proof (induction n rule: dec-induct)
  case (step n)
  have c * c ^ n ≤ Suc n * fact n
    using step by (intro mult-mono) auto
  thus ?case by simp
qed auto

```

```

lemma prime-multiplicity-prime:
  fixes p q :: 'a :: factorial-semiring
  assumes prime p prime q
  shows multiplicity p q = (if p = q then 1 else 0)
  using assms by (auto simp: prime-multiplicity-other)

```

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  $multiplicity\ p\ (fact\ n) = legendre\ aux\ n\ p$ 
proof (cases p ≤ n)
  case True
  have  $fact\ n = (\prod p \mid prime\ p \wedge p \leq n. p \wedge legendre\ aux\ n\ p)$ 
    using legendre-identity'[of real n] by simp
  also have  $multiplicity\ p \dots = (\sum q \mid prime\ q \wedge q \leq n. multiplicity\ p\ (q \wedge legendre\ aux\ n\ q))$ 
    using assms by (subst prime-elem-multiplicity-prod-distrib) auto
  also have  $\dots = (\sum q \in \{p\}. legendre\ aux\ n\ q)$ 
    using assms  $\langle p \leq n \rangle$  prime-multiplicity-other[of p]
    by (intro sum.mono-neutral-cong-right)
    (auto simp: prime-elem-multiplicity-power-distrib prime-multiplicity-prime split: if-splits)
  finally show ?thesis by simp
next
  case False
  hence  $multiplicity\ p\ (fact\ n) = 0$ 
    using assms by (intro not-dvd-imp-multiplicity-0) (auto simp: prime-dvd-fact-iff)
  moreover from False have  $legendre\ aux\ (real\ n)\ p = 0$ 
    by (intro legendre-aux-eq-0) auto
  ultimately show ?thesis by simp
qed

```

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

```

lemma legendre-aux-ge:
  assumes prime p k ≥ 1
  shows  $legendre\ aux\ k\ p \geq nat\ \lfloor k / p \rfloor$ 
proof (cases k ≥ p)
  case True
  have  $(\sum m \in \{1\}. nat\ \lfloor k / real\ p \wedge m \rfloor) \leq (\sum m \mid 0 < m \wedge real\ p \wedge m \leq k. nat\ \lfloor k / real\ p \wedge m \rfloor)$ 
    using True finite-sum-legendre-aux[of p] assms by (intro sum-mono2) auto
  with assms True show ?thesis by (simp add: legendre-aux-def)
next
  case False
  with assms have  $k / p < 1$  by (simp add: field-simps)
  hence  $nat\ \lfloor k / p \rfloor = 0$  by simp
  with False show ?thesis
    by (simp add: legendre-aux-eq-0)

```

qed

**lemma** *legendre-aux-less*:

**assumes** *prime*  $p$   $k \geq 1$

**shows** *legendre-aux*  $k$   $p < k / (p - 1)$

**proof** –

**have**  $(\lambda m. (k / p) * (1 / p) ^ m)$  *sums*  $((k / p) * (1 / (1 - 1 / p)))$

**using** *assms prime-gt-1-nat*[of  $p$ ] **by** (*intro sums-mult geometric-sums*) (*auto simp: field-simps*)

**hence** *sums*:  $(\lambda m. k / p ^ Suc\ m)$  *sums*  $(k / (p - 1))$

**using** *assms prime-gt-1-nat*[of  $p$ ] **by** (*simp add: field-simps of-nat-diff*)

**have** *real* (*legendre-aux*  $k$   $p$ ) =  $(\sum m \in \{0 <.. nat \lfloor \log (\text{real } p) k \rfloor\}. \text{of-int } \lfloor k / \text{real } p ^ m \rfloor)$

**using** *assms* **by** (*simp add: legendre-aux-altdef1*)

**also have**  $\dots = (\sum m < nat \lfloor \log (\text{real } p) k \rfloor. \text{of-int } \lfloor k / \text{real } p ^ Suc\ m \rfloor)$

**by** (*intro sum.reindex-bij-witness*[of - *Suc*  $\lambda i. i - 1$ ]) (*auto simp flip: power-Suc*)

**also have**  $\dots \leq (\sum m < nat \lfloor \log (\text{real } p) k \rfloor. k / \text{real } p ^ Suc\ m)$

**by** (*intro sum-mono*) *auto*

**also have**  $\dots < (\sum m. k / \text{real } p ^ Suc\ m)$

**using** *sums assms prime-gt-1-nat*[of  $p$ ]

**by** (*intro sum-less-suminf*) (*auto simp: sums-iff intro!: divide-pos-pos*)

**also have**  $\dots = k / (p - 1)$

**using** *sums* **by** (*simp add: sums-iff*)

**finally show** *?thesis*

**using** *assms prime-gt-1-nat*[of  $p$ ] **by** (*simp add: of-nat-diff*)

qed

## 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*  $\equiv (\prod i < n. 2 ^ n - 2 ^ i)$

**defines**  $T \equiv (n * (n - 1)) \text{div } 2$

**assumes** *pos*:  $k > 0$   $n > 0$

**assumes** *k-n: fact*  $k = \text{rhs}$

**begin**

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

**lemma** *rhs-altdef*:  $\text{rhs} = 2 ^ T * (\prod i = 1..n. 2 ^ i - 1)$

**proof** –

**have**  $\text{rhs} = (\prod i < n. 2 ^ i * (2 ^ (n - i) - 1))$

**by** (*simp add: rhs-def algebra-simps flip: power-add*)

**also have**  $\dots = 2 ^ (\sum i < n. i) * (\prod i < n. 2 ^ (n - i) - 1)$

```

    by (simp add: prod.distrib power-sum)
  also have  $(\sum_{i < n}. i) = T$ 
    unfolding T-def using Sum-Ico-nat[of 0 n] by (simp add: atLeast0LessThan)
  also have  $(\prod_{i < n}. 2^{(n-i)} - 1) = (\prod_{i=1..n}. 2^i - 1)$ 
    by (rule prod.reindex-bij-witness[of -  $\lambda i. n - i$   $\lambda i. n - i$ ]) auto
  finally show ?thesis .
qed

```

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 -
  have nz:  $2^i - 1 \neq 0 \text{ :: nat}$  if  $i \geq 1$  for  $i$ 
  proof -
    from  $\langle i \geq 1 \rangle$  have  $2^0 < (2^i \text{ :: nat})$ 
    by (intro power-strict-increasing) auto
  thus ?thesis by simp
qed

```

```

  have multiplicity 2 rhs = T + multiplicity 2  $(\prod_{i=1..n}. 2^i - 1 \text{ :: nat})$ 
    using nz by (simp add: rhs-altdef prime-elem-multiplicity-mult-distrib)
  also have multiplicity 2  $(\prod_{i=1..n}. 2^i - 1 \text{ :: nat}) = 0$ 
    by (intro not-dvd-imp-multiplicity-0) (auto simp: prime-dvd-prod-iff)
  finally show ?thesis by simp
qed

```

From Legendre's identities and the associated bounds, it can easily be seen that  $\lfloor k/2 \rfloor \leq T < k$ :

```

lemma k-gt-T:  $k > T$ 
proof -
  have  $T = \text{multiplicity } 2 \text{ rhs}$ 
    by simp
  also have  $\text{rhs} = \text{fact } k$ 
    by (simp add: k-n)
  also have  $\text{multiplicity } 2 (\text{fact } k \text{ :: nat}) = \text{legendre-aux } k \ 2$ 
    by (simp add: multiplicity-prime-fact)
  also have  $\dots < k$ 
    using legendre-aux-less[of 2 k] pos by simp
  finally show ?thesis .
qed

```

```

lemma T-ge-half-k:  $T \geq k \text{ div } 2$ 
proof -
  have  $k \text{ div } 2 \leq \text{legendre-aux } k \ 2$ 
    using legendre-aux-ge[of 2 k] pos by simp linarith?
  also have  $\dots = \text{multiplicity } 2 (\text{fact } k \text{ :: nat})$ 
    by (simp add: multiplicity-prime-fact)
  also have  $\dots = T$  by (simp add: k-n)
  finally show  $T \geq k \text{ div } 2$  .

```

qed

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** –

**have**  $rhs = 2^T * (\prod_{i=1..n} 2^{i-1})$   
**by** (*simp add: rhs-altdef*)  
**also have**  $(\prod_{i=1..n} 2^{i-1} :: nat) < (\prod_{i=1..n} 2^i)$   
**using** *pos* **by** (*intro prod-mono-strict[of 1]*) *auto*  
**also have**  $\dots = (\prod_{i=0..<n} 2 * 2^i)$   
**by** (*intro prod.reindex-bij-witness[of - Suc λi. i - 1]*) (*auto simp flip: power-Suc*)  
**also have**  $\dots = 2^n * 2^{(\sum_{i=0..<n} i)}$   
**by** (*simp add: power-sum prod.distrib*)  
**also have**  $(\sum_{i=0..<n} i) = T$   
**unfolding** *T-def* **by** (*simp add: Sum-Ico-nat*)  
**also have**  $2^T * (2^n * 2^T :: nat) = 2^{(2 * T + n)}$   
**by** (*simp flip: power-add power-Suc add: algebra-simps*)  
**also have**  $2 * T + n = n^2$   
**by** (*cases even n*) (*auto simp: T-def algebra-simps power2-eq-square*)  
**finally show**  $rhs < 2^{n^2}$   
**by** *simp*

qed

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  $T \geq 20$  and thereby  $n \geq 7$ . We can therefore conclude that  $n$  must be less than 7.

**lemma** *n-less-7*:  $n < 7$

**proof** (*rule ccontr*)

**assume**  $\neg n < 7$

**hence**  $n \geq 7$  **by** *simp*

**have**  $T \geq (7 * 6) \text{ div } 2$

**unfolding** *T-def* **using**  $\langle n \geq 7 \rangle$  **by** (*intro div-le-mono mult-mono*) *auto*

**hence**  $T \geq 21$  **by** *simp*

**from**  $\langle n \geq 7 \rangle$  **have**  $(n * 2) \text{ div } 2 \leq T$

**unfolding** *T-def* **by** (*intro div-le-mono*) *auto*

**hence**  $T \geq n$  **by** *simp*

**from**  $\langle T \geq 21 \rangle$  **have**  $\text{sqrt}(2 * \text{pi} * T) * (T / \text{exp } 1)^T \leq \text{fact } T$

**using** *fact-bounds[of T]* **by** *simp*

**have**  $\text{fact } T \leq (\text{fact } k :: nat)$

**using** *k-gt-T* **by** (*intro fact-mono*) (*auto simp: T-def*)

**also have**  $\dots = rhs$  **by** *fact*

**also have**  $rhs < 2^{n^2}$  **by** (*rule rhs-less*)

**also have**  $n^2 = 2 * T + n$

**by** (*cases even n*) (*auto simp: T-def algebra-simps power2-eq-square*)

**also have**  $\dots \leq 3 * T$

**using**  $\langle T \geq n \rangle$  **by** (*simp add: T-def*)

```

also have  $2 \wedge (3 * T) = (8 \wedge T :: \text{nat})$ 
  by (simp add: power-mult)
finally have fact  $T < (8 \wedge T :: \text{nat})$ 
  by simp
moreover have fact  $T \geq (8 \wedge T :: \text{nat})$ 
  by (rule fact-ge-power[of - 20]) (use  $\langle T \geq 21 \rangle$  in  $\langle \text{auto simp: fact-numeral} \rangle$ )
ultimately show False by simp
qed

```

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** –

```

define T' where  $T' = (\lambda n :: \text{nat}. n * (n - 1) \text{ div } 2)$ 
define A ::  $(\text{nat} \times \text{nat}) \text{ set}$  where  $A = (\text{SIGMA } n:\{1..6\}. \{T' n <.. 2 * T' n + 1\})$ 
define P where  $P = (\lambda(n, k). \text{fact } k = (\prod i < n. 2 \wedge n - 2 \wedge i :: \text{nat}))$ 
have [simp]:  $\{0 <.. \text{Suc } 0\} = \{1\}$  by auto
have  $(n, k) \in \text{Set.filter } P \ A$ 
  using k-n pos T-ge-half-k k-gt-T n-less-7
  by (auto simp: A-def T'-def T-def P-def rhs-def)
also have  $\text{Set.filter } P \ A = \{(1, 1), (2, 3)\}$ 
  by (simp add: P-def Set-filter-insert A-def atMost-nat-numeral atMost-Suc T'-def Sigma-insert
    greaterThanAtMost-nat-numeral atLeastAtMost-nat-numeral lessThan-nat-numeral
    fact-numeral split-def
    del: Set.filter-eq cong: if-weak-cong)
finally show ?thesis .
qed

```

**end**

Using this, deriving the final result is now trivial:

```

theorem  $\{(n, k). n > 0 \wedge k > 0 \wedge \text{fact } k = (\prod i < n. 2 \wedge n - 2 \wedge i :: \text{nat})\} = \{(1, 1), (2, 3)\}$ 
  (is ?lhs = ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs using n-k-in-set by blast
  show ?rhs  $\subseteq$  ?lhs by (auto simp: fact-numeral lessThan-nat-numeral)
qed

```

**end**

### 3 Q5

**theory** *IMO2019-Q5*

**imports** *Complex-Main*  
**begin**

Given a sequence  $(c_1, \dots, c_n)$  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*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool list* **where**  
*flip* *xs* *i* = *xs*[*i* :=  $\neg$ *xs* ! *i*]

**lemma** *flip-Cons-pos* [*simp*]:  $n > 0 \implies \text{flip } (x \# xs) \ n = x \# \text{flip } xs \ (n - 1)$   
**by** (*cases* *n*) (*auto simp: flip-def*)

**lemma** *flip-Cons-0* [*simp*]:  $\text{flip } (x \# xs) \ 0 = (\neg x) \# xs$   
**by** (*simp add: flip-def*)

**lemma** *flip-append1* [*simp*]:  $n < \text{length } xs \implies \text{flip } (xs \ @ \ ys) \ n = \text{flip } xs \ n \ @ \ ys$   
**and** *flip-append2* [*simp*]:  $n \geq \text{length } xs \implies n < \text{length } xs + \text{length } ys \implies$   
 $\text{flip } (xs \ @ \ ys) \ n = xs \ @ \ \text{flip } ys \ (n - \text{length } xs)$   
**by** (*auto simp: flip-def list-update-append nth-append*)

**lemma** *length-flip* [*simp*]:  $\text{length } (\text{flip } xs \ i) = \text{length } xs$   
**by** (*simp add: flip-def*)

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

**definition** *heads* :: *bool list*  $\Rightarrow$  *nat* **where** *heads* *xs* =  $\text{length } (\text{filter } id \ xs)$

**lemma** *heads-True* [*simp*]:  $\text{heads } (True \# xs) = 1 + \text{heads } xs$   
**and** *heads-False* [*simp*]:  $\text{heads } (False \# xs) = \text{heads } xs$   
**and** *heads-append* [*simp*]:  $\text{heads } (xs \ @ \ ys) = \text{heads } xs + \text{heads } ys$   
**and** *heads-Nil* [*simp*]:  $\text{heads } [] = 0$   
**by** (*auto simp: heads-def*)

**lemma** *heads-Cons*:  $\text{heads } (x \# xs) = (\text{if } x \text{ then } \text{heads } xs + 1 \text{ else } \text{heads } xs)$   
**by** (*auto simp: heads-def*)

**lemma** *heads-pos*:  $True \in \text{set } xs \implies \text{heads } xs > 0$   
**by** (*induction* *xs*) (*auto simp: heads-Cons*)

**lemma** *heads-eq-0* [simp]:  $True \notin set\ xs \implies heads\ xs = 0$   
**by** (induction *xs*) (auto simp: heads-Cons)

**lemma** *heads-eq-0-iff* [simp]:  $heads\ xs = 0 \longleftrightarrow True \notin set\ xs$   
**by** (induction *xs*) (auto simp: heads-Cons)

**lemma** *heads-pos-iff* [simp]:  $heads\ xs > 0 \longleftrightarrow True \in set\ xs$   
**by** (induction *xs*) (auto simp: heads-Cons)

**lemma** *heads-le-length*:  $heads\ xs \leq length\ xs$   
**by** (auto simp: heads-def)

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

**definition** *harry-step* ::  $bool\ list \Rightarrow bool\ list$  **where**  
*harry-step xs = flip xs (heads xs - 1)*

**lemma** *length-harry-step* [simp]:  $length\ (harry-step\ xs) = length\ xs$   
**by** (simp add: harry-step-def)

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)*  
**by** *auto*

**termination** **by** (relation *Wellfounded.measure length*) (auto simp: min-def)

**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**  $xs = [] \implies P$   
**assumes**  $\bigwedge ys. xs = True \# ys \implies P \ \bigwedge ys. xs = ys @ [False] \implies P$   
 $\bigwedge ys. xs = False \# ys @ [True] \implies P$   
**shows**  $P$   
**proof** –  
**consider**  $length\ xs = 0 \mid length\ xs = 1 \mid length\ xs \geq 2$  **by** *linarith*  
**thus** *?thesis*  
**proof** *cases*

```

assume  $length\ xs = 1$ 
hence  $xs = [hd\ xs]$  by (cases xs) auto
thus  $P$  using assms(2)[of []] assms(3)[of []] by (cases hd xs) auto
next
assume  $len: length\ xs \geq 2$ 
from  $len$  obtain  $x\ xs'$  where  $*$ :  $xs = x \# xs'$ 
  by (cases xs) auto
have  $**$ :  $xs' = butlast\ xs' @ [last\ xs']$ 
  using  $len$  by (subst append-butlast-last-id) (auto simp: *)
have [simp]:  $xs = x \# butlast\ xs' @ [last\ xs']$ 
  by (subst *, subst **) auto
show  $P$ 
  using assms(2)[of  $xs'$ ] assms(3)[of  $x \# butlast\ xs'$ ] assms(4)[of  $butlast\ xs'$ ]  $**$ 
  by (cases x; cases last xs') auto
qed (use assms in auto)
qed

```

```

lemma harry-meas-Nil [simp]:  $harry-meas\ [] = 0$ 
by (simp add: harry-meas.simps)

```

```

lemma harry-meas-True-start [simp]:  $harry-meas\ (True \# xs) = 1 + harry-meas\ xs$ 
by (subst harry-meas.simps) auto

```

```

lemma harry-meas-False-end [simp]:  $harry-meas\ (xs @ [False]) = harry-meas\ xs$ 
proof (induction xs)
  case (Cons x xs)
  thus  $?case$  by (cases x) (auto simp: harry-meas.simps)
qed (auto simp: harry-meas.simps)

```

```

lemma harry-meas-False-True:  $harry-meas\ (False \# xs @ [True]) = harry-meas\ xs + 2 * length\ xs + 3$ 
by (subst harry-meas.simps) auto

```

```

lemma harry-meas-eq-0 [simp]:
  assumes  $True \notin set\ xs$ 
  shows  $harry-meas\ xs = 0$ 
  using assms by (induction xs rule: rev-induct) auto

```

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]:
   $harry-step\ (True \# xs) = (if\ True \in set\ xs\ then\ True \# harry-step\ xs\ else\ False \# xs)$ 
by (auto simp: harry-step-def)

```

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**  $True \in set\ xs$   
**shows**  $harry-step\ (xs\ @\ [False]) = harry-step\ xs\ @\ [False]$   
**proof** –  
**have**  $harry-step\ (xs\ @\ [False]) = flip\ (xs\ @\ [False])\ (heads\ xs - 1)$   
**using**  $heads-le-length[of\ xs]$  **by**  $(auto\ simp:\ harry-step-def)$   
**also have**  $\dots = harry-step\ xs\ @\ [False]$   
**using**  $Suc-less-eq\ assms\ heads-le-length[of\ xs]$   
**by**  $(subst\ flip-append1;\ fastforce\ simp:\ harry-step-def)$   
**finally show**  $?thesis$  .  
**qed**

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**  $True \in set\ xs$   
**shows**  $harry-step\ (False\ \#\ xs\ @\ [True]) = False\ \#\ harry-step\ xs\ @\ [True]$   
**proof** –  
**have**  $harry-step\ (False\ \#\ xs\ @\ [True]) = False\ \#\ flip\ (xs\ @\ [True])\ (heads\ xs - 1)$   
**using**  $assms\ heads-le-length[of\ xs]$  **by**  $(auto\ simp:\ harry-step-def\ heads-le-length)$   
**also have**  $\dots = False\ \#\ harry-step\ xs\ @\ [True]$   
**using**  $assms$  **by**  $(subst\ flip-append1)$   
 $(auto\ simp:\ harry-step-def\ Suc-less-SucD\ heads-le-length\ less-Suc-eq-le)$   
**finally show**  $?thesis$  .  
**qed**

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**  $(cases\ n = 0)$   
**case**  $False$   
**hence**  $replicate\ n\ False\ @\ [True] = False\ \#\ replicate\ (n - 1)\ False\ @\ [True]$   
**by**  $(cases\ n)\ auto$   
**also have**  $harry-meas\ \dots = 2 * n + 1$   
**using**  $False$  **by**  $(simp\ add:\ harry-meas-False-True\ algebra-simps)$   
**finally show**  $?thesis$  .  
**qed**  $auto$

**lemma**  $harry-step-Falses-True$   $[simp]:$   
 $n > 0 \implies harry-step\ (replicate\ n\ False\ @\ [True]) = True\ \#\ replicate\ (n - 1)\ False\ @\ [True]$   
**by**  $(cases\ n)\ (simp-all\ add:\ harry-step-def)$

### 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**  $True \in set\ xs$

**shows**  $harry-meas\ xs = Suc\ (harry-meas\ (harry-step\ xs))$

**using** *assms*

**proof** (*induction xs rule: length-induct*)

**case** ( $1\ xs$ )

**hence** *IH*:  $harry-meas\ ys = Suc\ (harry-meas\ (harry-step\ ys))$

**if**  $length\ ys < length\ xs$   $True \in set\ ys$  **for**  $ys$

**using** *that by blast*

**show** *?case*

**proof** (*cases xs rule: head-last-cases*)

**case** ( $True\ ys$ )

**thus** *?thesis by (auto simp: IH)*

**next**

**case** ( $False\ ys$ )

**thus** *?thesis using 1.prem by (auto simp: IH)*

**next**

**case** ( $False-True\ ys$ )

**thus** *?thesis*

**proof** (*cases True ∈ set ys*)

**case** *False*

**define**  $n$  **where**  $n = length\ ys + 1$

**have**  $n > 0$  **by** (*simp add: n-def*)

**from** *False* **have**  $ys = replicate\ (n - 1)\ False$

**unfolding** *n-def by (induction ys) auto*

**with**  $False-True\ \langle n > 0 \rangle$  **have** [*simp*]:  $xs = replicate\ n\ False\ @\ [True]$

**by** (*cases n auto*)

**show** *?thesis using <n > 0> by auto*

**qed** (*auto simp: IH False-True harry-step-False-True harry-meas-False-True*)

**qed** (*use 1 in auto*)

**qed**

**lemma** *harry-meas-step*:  $True \in set\ xs \implies harry-meas\ (harry-step\ xs) = harry-meas\ xs - 1$

**using** *harry-meas-step-aux[of xs] by simp*

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*]:  $harry-meas\ xs = 0 \iff True \notin set\ xs$

**proof** (*induction xs rule: length-induct*)

**case** ( $1\ xs$ )

**show** *?case*

**by** (*cases xs rule: head-last-cases (auto simp: 1 harry-meas-False-True 1)*)

**qed**

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**  $n < \text{harry-meas } xs$

**shows**  $\text{True} \in \text{set } ((\text{harry-step } \sim^n) xs)$

**using** *assms*

**proof** (*induction n arbitrary: xs*)

**case**  $0$

**show** *?case* **by** (*rule ccontr*) (*use 0 in auto*)

**next**

**case** (*Suc n*)

**have**  $\text{True} \in \text{set } xs$  **by** (*rule ccontr*) (*use Suc in auto*)

**have**  $(\text{harry-step } \sim^{\text{Suc } n} xs) = (\text{harry-step } \sim^n) (\text{harry-step } xs)$

**by** (*simp only: funpow-Suc-right o-def*)

**also have**  $\text{True} \in \text{set } \dots$

**using** *Suc <True ∈ set xs>* **by** (*intro Suc*) (*auto simp: harry-meas-step*)

**finally show** *?case* .

**qed**

**lemma** *True-notin-funpow-harry-step: True ∉ set ((harry-step ∼<sup>n</sup> harry-meas xs) xs)*

**proof** (*induction harry-meas xs arbitrary: xs*)

**case** (*Suc n*)

**have**  $\text{True} \in \text{set } xs$  **by** (*rule ccontr*) (*use Suc in auto*)

**have**  $(\text{harry-step } \sim^{\text{harry-meas } xs} xs) = (\text{harry-step } \sim^{\text{Suc } n} xs)$

**by** (*simp only: Suc*)

**also have**  $\dots = (\text{harry-step } \sim^n) (\text{harry-step } xs)$

**by** (*simp only: funpow-Suc-right o-def*)

**also have**  $\dots = (\text{harry-step } \sim^{\text{harry-meas } xs - 1}) (\text{harry-step } xs)$

**by** (*simp flip: Suc(2)*)

**also have**  $\text{harry-meas } xs - 1 = \text{harry-meas } (\text{harry-step } xs)$

**using** *<True ∈ set xs>* **by** (*subst harry-meas-step*) *auto*

**also have**  $\text{True} \notin \text{set } ((\text{harry-step } \sim^{\dots}) (\text{harry-step } xs))$

**using** *Suc <True ∈ set xs>* **by** (*intro Suc*) (*auto simp: harry-meas-step*)

**finally show** *?case* .

**qed** *auto*

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**  $\text{harry-meas } xs = (\text{LEAST } n. \text{True} \notin \text{set } ((\text{harry-step } \sim^n) xs))$

**proof** (*rule sym, rule Least-equality, goal-cases*)

**show**  $\text{True} \notin \text{set } ((\text{harry-step } \sim^{\text{harry-meas } xs}) xs)$

**by** (*rule True-notin-funpow-harry-step*)

**next**

**case** ( $2 y$ )

**show** *?case*

**by** (*rule ccontr*) (*use 2 True-in-funpow-harry-step[of y] in auto*)

**qed**

### 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*]:  $xs \in seqs\ n \implies length\ xs = n$   
**by** (*simp add: seqs-def*)

**lemma** *seqs-0* [*simp, code*]:  $seqs\ 0 = \{\}\}$   
**by** (*auto simp: seqs-def*)

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* [*code*]:  $seqs\ (Suc\ n) = (\lambda xs.\ True\ \#\ xs)\ ' seqs\ n \cup (\lambda xs.\ False\ \#\ xs)\ ' seqs\ n$   
**by** (*auto simp: seqs-def length-Suc-conv*)

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

**lemma** *seqs-rev* [*simp*]:  $rev\ ' seqs\ n = seqs\ n$

**proof**

**show**  $rev\ ' seqs\ n \subseteq seqs\ n$

**by** (*auto simp: seqs-def*)

**hence**  $rev\ ' rev\ ' seqs\ n \subseteq rev\ ' seqs\ n$

**by** *blast*

**thus**  $seqs\ n \subseteq rev\ ' seqs\ n$  **by** (*simp add: image-image*)

**qed**

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

**lemma** *seqs-Suc'*:  $seqs\ (Suc\ n) = (\lambda xs.\ xs\ @\ [True])\ ' seqs\ n \cup (\lambda xs.\ xs\ @\ [False])\ ' seqs\ n$

**proof** –

**have**  $rev\ ' rev\ ' ((\lambda xs.\ xs\ @\ [True])\ ' seqs\ n \cup (\lambda xs.\ xs\ @\ [False])\ ' seqs\ n) =$   
 $rev\ ' ((\lambda xs.\ True\ \#\ xs)\ ' rev\ ' seqs\ n \cup (\lambda xs.\ False\ \#\ xs)\ ' rev\ ' seqs\ n)$

**unfolding** *image-Un image-image* **by** *simp*

**also have**  $(\lambda xs.\ True\ \#\ xs)\ ' rev\ ' seqs\ n \cup (\lambda xs.\ False\ \#\ xs)\ ' rev\ ' seqs\ n =$   
 $seqs\ (Suc\ n)$

**by** (*simp add: seqs-Suc*)

**finally show** *?thesis* **by** (*simp add: image-image*)

**qed**

**lemma** *finite-seqs* [*intro*]: *finite* (*seqs*  $n$ )

**by** (*induction n*) (*auto simp: seqs-Suc*)

**lemma** *card-seqs* [*simp*]:  $card\ (seqs\ n) = 2\ ^\ n$

**proof** (*induction n*)

**case** (*Suc*  $n$ )

**have**  $card\ (seqs\ (Suc\ n)) = card\ ((\#\ True\ ' seqs\ n \cup (\#\ False\ ' seqs\ n))$

**by** (*auto simp: seqs-Suc*)

**also from** *Suc.IH* **have**  $\dots = 2 \wedge \text{Suc } n$   
**by** (*subst card-Un-disjoint*) (*auto simp: card-image*)  
**finally show** *?case* .  
**qed** *auto*

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

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

**lemma** *Suc-Suc-induct*:  $P\ 0 \Longrightarrow P\ (\text{Suc } 0) \Longrightarrow (\bigwedge n. P\ n \Longrightarrow P\ (\text{Suc } n) \Longrightarrow P\ (\text{Suc } (\text{Suc } n))) \Longrightarrow P\ n$   
**by** *induction-schema* (*pat-completeness*, *rule wf-measure[of id]*, *auto*)

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

**lemma** *harry-sum-correct*: *harry-sum* n = *sum* *harry-meas* (*seqs* n)

**proof** (*induction n rule: Suc-Suc-induct*)

**case** (*3* n)

**have** *seqs* (*Suc* (*Suc* n)) =  
 $(\lambda xs. xs\ @\ [False])\ 'seqs\ (\text{Suc } n) \cup$   
 $(\lambda xs. True\ \#\ xs\ @\ [True])\ 'seqs\ n \cup$   
 $(\lambda xs. False\ \#\ xs\ @\ [True])\ 'seqs\ n$

**by** (*subst* 1) *seqs-Suc*, *subst* (1 2) *seqs-Suc'* (*simp add: image-Un image-image Un-ac seqs-Suc*)

**also have** *int* (*sum* *harry-meas* ...) =  
 $int\ (\text{harry-sum } (\text{Suc } n)) +$   
 $int\ (\sum_{xs \in seqs\ n} 1 + \text{harry-meas } (xs\ @\ [True])) +$   
 $int\ (\sum_{xs \in seqs\ n} \text{harry-meas } (False\ \#\ xs\ @\ [True]))$

**by** (*subst sum.union-disjoint sum.reindex*, *auto simp: inj-on-def 3*)

**also have**  $int\ (\sum_{xs \in seqs\ n} 1 + \text{harry-meas } (xs\ @\ [True])) =$   
 $2 \wedge n + int\ (\sum_{xs \in seqs\ n} \text{harry-meas } (xs\ @\ [True]))$

**by** (*subst sum.distrib*) *auto*

**also have**  $(\sum_{xs \in seqs\ n} \text{harry-meas } (False\ \#\ xs\ @\ [True])) = \text{harry-sum } n +$   
 $(2 * n + 3) * 2 \wedge n$

**by** (*auto simp: 3 harry-meas-False-True sum.distrib algebra-simps length-seqs*)

**also have** *harry-sum* (*Suc* n) =  $(\sum_{xs \in seqs\ n} \text{harry-meas } (xs\ @\ [True])) +$   
*harry-sum* n

**unfolding** *seqs-Suc'* 3 **by** (*subst sum.union-disjoint sum.reindex*, *auto simp: inj-on-def*)

**hence**  $int\ (\sum_{xs \in seqs\ n} \text{harry-meas } (xs\ @\ [True])) = int\ (\text{harry-sum } (\text{Suc } n))$   
 $- int\ (\text{harry-sum } n)$

**by** *simp*

**finally have**  $int\ (\sum_{x \in seqs\ (\text{Suc } (\text{Suc } n))} \text{harry-meas } x) =$   
 $int\ (2 * \text{harry-sum } (\text{Suc } n) + (2 * n + 4) * 2 \wedge n)$

**unfolding** *of-nat-add* **by** (*simp add: algebra-simps*)

**hence**  $(\sum_{x \in seqs\ (\text{Suc } (\text{Suc } n))} \text{harry-meas } x) =$   
 $(2 * \text{harry-sum } (\text{Suc } n) + (2 * n + 4) * 2 \wedge n)$  **by** *linarith*

**thus** *?case* **by** *simp*  
**qed** (*auto simp: seqs-Suc*)

**lemma** *harry-sum-closed-form-aux*:  $4 * \text{harry-sum } n = n * (n + 1) * 2^n$   
**by** (*induction n rule: harry-sum.induct*) (*auto simp: algebra-simps*)

Solving the recurrence gives us the following solution:

**theorem** *harry-sum-closed-form*:  $\text{harry-sum } n = n * (n + 1) * 2^n \text{ div } 4$   
**using** *harry-sum-closed-form-aux*[of *n*] **by** *simp*

The average is now a simple consequence:

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

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

**proof** –

**have**  $\text{real } (4 * \text{harry-sum } n) = n * (n + 1) * 2^n$

**by** (*subst harry-sum-closed-form-aux*) *auto*

**hence**  $\text{real } (\text{harry-sum } n) = n * (n + 1) * 2^n / 4$

**by** (*simp add: field-simps*)

**thus** *?thesis*

**by** (*simp add: harry-avg-def field-simps*)

**qed**

**end**

## References

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