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

1 Q1 2

2 Q4 3
  2.1 Auxiliary facts .............................................. 3
  2.2 Main result ................................................... 5

3 Q5 8
  3.1 Definition .................................................... 9
  3.2 Correctness of the measure ................................. 12
  3.3 Average-case analysis ...................................... 15
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 \( c \in \mathbb{Z} \).

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.

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

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

\[
\forall a, b, f(2a) + 2f(b) = f(f(a+b)) \iff \exists c \in \mathbb{Z} \forall x, f(x) = 2x + c.
\]

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

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

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

\[
\forall a, b, f(2a) + 2f(b) = f(f(a+b)) \iff \exists c \in \mathbb{Z} \forall x, f(x) = 2x + c.
\]

This version is better for the simplifier because it prevents it from looping.
next
 assume ?rhs
 thus ?lhs by smt
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: \(\Sigma (\text{insert } x A) f = (\lambda y. (x, y)) \cdot f x \cup \Sigma A f\)
by auto

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

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

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

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

\textit{legendre-aux} (related to Legendre’s identity) is the multiplicity of a given prime in the prime factorisation of \( n! \).

\textbf{lemma} \textit{multiplicity-prime-fact}:  
\begin{verbatim}
fixes \( p : \mathbb{N} \)
assumes \( \text{prime } p \)
shows \( \text{multiplicity } p \ (\text{fact } n) = \text{legendre-aux } n \ p \)
proof (cases \( p \leq n \))
  case True
  have \( \text{fact } n = (\prod p \mid \text{prime } p \land p \leq n. \ p ^ \text{legendre-aux } n \ p) \)
    using \( \text{legendre-identity}\ [\text{of real } n] \) by simp
  also have \( \ldots = (\sum q \mid \text{prime } q \land q \leq n. \text{multiplicity } p \ (q ^ \text{legendre-aux } n \ q)) \)
    using \( \text{assms} \) by \( \text{auto} \)
  also have \( \ldots = (\sum q \in \{p\}. \text{legendre-aux } n \ q) \)
    using \( \text{assms} \) \( \text{prime-multiplicity-other}[\text{of } p] \)
    by \( \text{(intro sum.mono-neutral-cong-right)} \)
      \( \text{(auto simp: \text{prime-elem-multiplicity-power-distrib \text{prime-multiplicity-prime}})} \)
  finally show \( \text{thesis} \) by simp
next
  case False
  hence \( \text{multiplicity } p \ (\text{fact } n) = 0 \)
    using \( \text{assms} \) \( \text{prime-dvd-fact-iff} \) \( \text{auto} \)
  moreover from False have \( \text{legendre-aux} \ (\text{real } n) \ p = 0 \)
    by \( \text{(intro \text{legendre-aux-eq-0}) auto} \)
  ultimately show \( \text{thesis} \) by simp
qed
\end{verbatim}

The following are simple and trivial lower and upper bounds for \textit{legendre-aux}:

\textbf{lemma} \textit{legendre-aux-ge}:
\begin{verbatim}
assumes \( \text{prime } p \ k \geq 1 \)
shows \( \text{legendre-aux } k \ p \geq \mathbb{N} \lfloor k / p \rfloor \)
proof (cases \( k \geq p \))
  case True
  have \( (\sum m \in \{1\}. \mathbb{N} \lfloor k / \text{real } p ^ m \rfloor) \leq (\sum m \mid 0 < m \land \text{real } p ^ m \leq k. \mathbb{N} \lfloor k / \text{real } p ^ m \rfloor) \)
    using \( \text{True finite-sum-legendre-aux[of } p \text{]} \) \( \text{assms} \) \( \text{auto} \)
    \( \text{introsim}{\text{add: legendre-aux-def}} \)
  with \( \text{assms} \) \( \text{True} \) show \( \text{thesis} \) by \( \text{simp} \)
next
  case False
  with \( \text{assms} \) have \( k / p < 1 \) by \( \text{simp} \)
  hence \( \mathbb{N} \lfloor k / p \rfloor = 0 \) by simp
  with False show \( \text{thesis} \)
    by \( \text{(simp add: \text{legendre-aux-eq-0})} \)
qed
\end{verbatim}
lemma legendre-aux-less:
assumes prime p k ≥ 1
shows legendre-aux k p < k / (p − 1)
proof
have (λm. (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: (λm. k / p * Suc m) sums (k / (p − 1))

have real (legendre-aux k p) = (∑m∈{0<..nat ⌊log (real p) k⌋}. of-int ⌊k / real p ^ m⌋)
using assms by (simp add: legendre-aux-altdef1)

also have ... ≤ (∑m∈{0<..nat ⌊log (real p) k⌋}. k / real p ^ Suc m)
using assms prime-gt-1-nat[of p] (auto simp flip: power-Suc)
also have ... = k / (p − 1)
using sums assms by (simp add: of-nat-diff)

finally show ?thesis
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 ≡ (∏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 * (∏i=1..n. 2 ^ i − 1)
proof
have rhs = (∏i<n. 2 ^ i * (2 ^ (n − i) − 1))
by (simp add: rhs-def algebra-simps flip: power-add)
also have ... = 2 ^ (∑i<n. i) * (∏i<n. 2 ^ (n − i) − 1)
by (simp add: prod.distrib power-sum)
also have \( \sum_{i<n} i = T \)
unfolding \( T \)-def using \( \text{Sum-Ico-nat[of 0 n]} \) by (simp add: atLeast0LessThan)
also have \( \prod_{i<n} 2^i \cdot (n-i) - 1 = (\prod_{i=1..n} 2^i - 1) \)
by (rule prod.reindex-bij-witness[of - \( \lambda i. n-i \cdot n-i \)]) 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 \( i \geq 1 \) have \( 2^0 < (2^i :: \text{nat}) \)
    by (intro power-strict-increasing) auto
    thus \(?thesis\) by simp
  qed

  have multiplicity 2 rhs = \( T + \text{multiplicity 2} (\prod_{i=1..n} 2^i - 1 :: \text{nat}) \)
  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 \( T = \text{multiplicity 2 rhs} \)
  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 \( \ldots < k \)
  using \( \text{legendre-aux-less[of 2 k]} \) pos by simp
  finally show \(?thesis\).
  qed

lemma T-ge-half-k; \( T \geq k \div 2 \)
proof
  have \( k \div 2 \leq \text{legendre-aux} k 2 \)
  using \( \text{legendre-aux-ge[of 2 k]} \) pos by simp linarith?
  also have \( \ldots = \text{multiplicity 2} (\text{fact} k :: \text{nat}) \)
  by (simp add: multiplicity-prime-fact)
  also have \( \ldots = T \) by (simp add: k-n)
  finally show \( T \geq k \div 2 \).
  qed
It can also be seen fairly easily that the right-hand side is strictly smaller than $2^n^2$:

**Lemma** rhs-less: $rhs < 2^\cdot n^2$

**Proof**

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

- assume $\neg n < 7$
- hence $n \geq 7$ by simp
- have $T \geq (7 \cdot 6) \div 2$
  - unfolding $T$-def using $(n \geq 7)$ by (intro div-le-mono mult-mono) auto
  - hence $T \geq 21$ by simp

from $(n \geq 7)$ have $(n \cdot 2) \div 2 \leq T$

  - unfolding $T$-def by (intro div-le-mono) auto
  - hence $T \geq n$ by simp

from $(T \geq 21)$ have $\sqrt{(2 \cdot pi \cdot T) \cdot (T / exp 1)} \cdot T \leq fact T$

  - using fact-bounds[of T] by simp
  - have $fact T \leq (fact k :: nat)$
    - using k-gt-T by (intro fact-mono) (auto simp: T-def)
  - also have $\ldots = rhs$ by fact
  - also have $rhs < 2^\cdot n^2$ by (rule rhs-less)
  - also have $n^2 = 2 \cdot T + n$
    - by (cases even n) (auto simp: T-def algebra-simps power2-eq-square)
  - also have $\ldots \leq 3 \cdot T$
    - using $(T \geq n)$ by (simp add: T-def)
  - also have $2^\cdot (3 \cdot T) = (8^\cdot T :: nat)$
    - by (simp add: power-mult)
finally have \( \text{fact } T < (8 \cdot T :: \text{nat}) \)
  by simp
moreover have \( \text{fact } T \geq (8 \cdot T :: \text{nat}) \)
  by (rule fact-ge-power[of - 20]) (use \( T \geq 21 \) in :auto simp: fact-numeral)
ultimately show \( \text{False} \) 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 \( T' = (\lambda n :: \text{nat}. n \cdot (n - 1) \div 2) \)
  define \( A :: (\text{nat} \times \text{nat}) \text{set} \) where \( A = (\Sigma n : \{1..6\}. \{ T' n < 2 \cdot T' n + 1 \}) \)
  define \( P \) where \( P = (\lambda (n, k). \text{fact } k = (\prod i < n. 2 \cdot n - 2 \cdot i :: \text{nat})) \)
  have [simp]: \( \{0 <.. Suc 0\} = \{1\} \) 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 Set.filter-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
      cong: if-weak-cong)
  finally show ?thesis .
qed
end

Using this, deriving the final result is now trivial:

theorem \{ (n, k). \( n > 0 \land k > 0 \land \text{fact } k = (\prod i < n. 2 \cdot n - 2 \cdot 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 \((c_1, \ldots, c_n)\) 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** \(\text{flip} :: \text{bool list} \Rightarrow \text{nat} \Rightarrow \text{bool list}\)

\[
\text{flip} \; xs \; i = xs[i := \neg xs \; ! \; i]
\]

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

by (cases \(n\)) (auto simp: \(\text{flip-def}\))

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

by (simp add: \(\text{flip-def}\))

**lemma** \(\text{flip-append1} [\text{simp}]: n < \text{length} \; xs \implies \text{flip} \; (xs @ \; ys) \; n = \text{flip} \; xs \; n @ \; ys\)

and \(\text{flip-append2} [\text{simp}]: n \geq \text{length} \; xs \implies n < \text{length} \; xs + \text{length} \; ys \implies \text{flip} \; (xs @ \; ys) \; n = xs @ \; \text{flip} \; ys \; (n - \text{length} \; xs)\)

by (auto simp: \(\text{flip-def}\) list-update-append nth-append)

**lemma** \(\text{length-flip} [\text{simp}]: \text{length} \; (\text{flip} \; xs \; i) = \text{length} \; xs\)

by (simp add: \(\text{flip-def}\))

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

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

**lemma** \(\text{heads-True} [\text{simp}]: \text{heads} \; (\text{True} \# \; xs) = 1 + \text{heads} \; xs\)

and \(\text{heads-False} [\text{simp}]: \text{heads} \; (\text{False} \# \; xs) = \text{heads} \; xs\)

and \(\text{heads-append} [\text{simp}]: \text{heads} \; (xs @ \; ys) = \text{heads} \; xs + \text{heads} \; ys\)

and \(\text{heads-Nil} [\text{simp}]: \text{heads} \; [] = 0\)

by (auto simp: \(\text{heads-def}\))

**lemma** \(\text{heads-Cons} [\text{simp}]: \text{heads} \; (x \# \; xs) = (\text{if} \; x \; \text{then} \; \text{heads} \; xs + 1 \; \text{else} \; \text{heads} \; xs)\)

by (auto simp: \(\text{heads-def}\))

**lemma** \(\text{heads-pos} [\text{simp}]: \text{True} \in \text{set} \; xs \implies \text{heads} \; xs > 0\)

by (induction \(xs\)) (auto simp: \(\text{heads-Cons}\))

**lemma** \(\text{heads-eq-0} [\text{simp}]: \text{True} \notin \text{set} \; xs \implies \text{heads} \; xs = 0\)

by (induction \(xs\)) (auto simp: \(\text{heads-Cons}\))
lemma heads-eq-0-iff [simp]: heads xs = 0 ←→ True ∉ set xs
by (induction xs) (auto simp: heads-Cons)

lemma heads-pos-iff [simp]: heads xs > 0 ←→ True ∈ set xs
by (induction xs) (auto simp: heads-Cons)

lemma heads-le-length: heads xs ≤ length xs
by (auto simp: heads-def)

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
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 = [] ⇒ P
assumes \( \forall ys. \, \text{xs} = \text{True} \# \, \text{ys} \Rightarrow P \) \( \forall ys. \, \text{xs} = \text{ys} \# \, [\text{False}] \Rightarrow P \)
\( \forall ys. \, \text{xs} = \text{False} \# \, \text{ys} \# \, [\text{True}] \Rightarrow P \)
shows \( P \)
proof –
consider length xs = 0 | length xs = 1 | length xs ≥ 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
assumption length xs ≥ 2
from length obtain x xs’ where *: xs = x # xs’
  by cases xs auto
have **: xs’ = butlast xs’ @ [last xs]
  using length 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-measNil [simp]: harry-meas [] = 0
  by (simp add: harry-meas.simps)

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

lemma harry-measFalse-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-measFalse-True: harry-meas (False # xs @ [True]) = harry-meas xs  
  by (subst harry-meas.simps) auto

lemma harry-meas-eq-0 [simp]:
  assumes True ∉ 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-stepTrue-start [simp]:
  harry-step (True # xs) = (if True ∈ set xs then True # harry-step xs else False  
  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-stepFalse-end [simp]:
  assumes True ∈ set xs
  shows harry-step (xs @ [False]) = harry-step xs @ [False]
proof

have harry-step \( (xs @ [False]) = \text{flip} \ (xs @ [False]) \ (\text{heads} \ xs - 1) \)
  using heads-le-length[of \( xs \)] by (auto simp: harry-step-def)
also have \ldots = 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 \( \text{True} \in \text{set} \ (xs) \)
  shows harry-step \( (\text{False} \# \ (xs @ [\text{True}])) = \text{False} \# \text{harry-step} \ (xs @ [\text{True}] \)\)
proof
  have harry-step \( (\text{False} \# \ (xs @ [\text{True}])) = \text{False} \# \text{flip} \ (xs @ [\text{True}] \ (\text{heads} \ xs - 1) \)
    using assms heads-le-length[of \( xs \)] by (auto simp: harry-step-def heads-le-length)
  also have \ldots = False \# harry-step \( xs @ [\text{True}] \)
    using assms by (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 \( \text{replicate} \ (n) \text{False} @ [\text{True}] \) = \( 2 \# n + 1 \)
proof (cases \( n = 0 \))
  case False
  hence \( \text{replicate} \ n \text{False} @ [\text{True}] = \text{False} \# \text{replicate} \ (n - 1) \text{False} @ [\text{True}] \)
    by (cases \( n \)) auto
  also have \( \text{harry-meas} \ldots = 2 \# n + 1 \)
    using False by (simp add: harry-meas-False-True algebra-simps)
  finally show ?thesis.
qed auto

lemma harry-step-Falses-True [simp]:
  \( n > 0 \implies \text{harry-step} \ (\text{replicate} \ n \text{False} @ [\text{True}]) = \text{True} \# \text{replicate} \ (n - 1) \text{False} @ [\text{True}] \)
  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 ∈ 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 ∈ 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.prems 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 (n > 0) 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 ∈ set xs ⇒ 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 ⇐⇒ True ∉ 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 < harry-meas xs
  shows True ∈ set ((harry-step ^^ 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 True ∈ set xs by (rule ccontr) (use Suc in auto)
  have (harry-step ^^ Suc n) xs = (harry-step ^^ n) (harry-step xs)
    by (simp only: funpow-Suc-right o-def)
  also have True ∈ set ...
  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 ^^ harry-meas xs) xs)
proof (induction harry-meas xs arbitrary: xs)
case (Suc n)
  have True ∈ set xs by (rule ccontr) (use Suc in auto)
  have (harry-step ^^ harry-meas xs) xs = (harry-step ^^ Suc n) xs
    by (simp only: Suc)
  also have ... = (harry-step ^^ n) (harry-step xs)
    by (simp only: funpow-Suc-right o-def)
  also have ... = (harry-step ^^ (harry-meas xs - 1)) (harry-step xs)
    by (simp flip: Suc(2))
  also have harry-meas xs - 1 = harry-meas (harry-step xs)
    using (True ∈ set xs) by (subst harry-meas-step) auto
  also have True /∈ set ((harry-step ^^ ...) (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 harry-meas xs = (LEAST n. True /∈ set ((harry-step ^^ n) xs))
proof (rule sym, rule Least-equality, goal-cases)
  show True /∈ set ((harry-step ^^ 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 ∈ seqs n ⇒ length xs = n
  by (simp add: seqs-def)

lemma seqs-0 [simp]: 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: seqs (Suc n) = (λxs. True # xs) ' seqs n ∪ (λxs. 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 ⊆ seqs n
    by (auto simp: seqs-def)
  hence rev ' rev ' seqs n ⊆ rev ' seqs n
    by blast
  thus seqs n ⊆ 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) = (λxs. xs @ [True]) ' seqs n ∪ (λxs. xs @ [False]) ' seqs n
proof
  have rev ' ((λxs. xs @ [True]) ' seqs n ∪ (λxs. xs @ [False]) ' seqs n) =
    rev ' ((λxs. True # xs) ' rev ' seqs n ∪ (λxs. False # xs) ' rev ' seqs n)
    unfolding image-Un image-image by simp
  also have (λxs. True # xs) ' rev ' seqs n ∪ (λxs. 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 ∪ (#) False ' seqs n)
    by (auto simp: seqs-Suc)
also from Suc.IH have ... = 2 * Suc n
  by (subst card-Un-disjoint) (auto simp: card-image)
finally show ?case.
qed auto

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-sum :: nat ⇒ 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 ^ n

lemma Suc-Suc-induct: P 0 ⇒ P ( Suc 0) ⇒ (∀ n. P n ⇒ P ( Suc n)) ⇒ P ( Suc ( Suc n)) ⇒ 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 (Suc 3 n)
  have seqs (Suc ( Suc n)) =
    (λxs. xs @ [False]) * seqs ( Suc n) ∪
    (λxs. True # xs @ [True]) * seqs n ∪
    (λxs. 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 (harry-sum (Suc n)) +
  int (∑ xs ∈ seqs n. 1 + harry-meas (xs @ [True])) +
  int (∑ xs ∈ seqs n. harry-meas (False # xs @ [True]))
  by (subst sum.union-disjoint sum.reindex, auto simp: inj-on-def)+
also have int (∑ xs ∈ seqs n. 1 + harry-meas (xs @ [True])) =
    2 ^ n + int (∑ xs ∈ seqs n. harry-meas (xs @ [True]))
  by (subst sum.distrib) auto
also have (∑ xs ∈ seqs n. harry-meas (False # xs @ [True])) =
    harry-sum n +
    (2 * n + 3) * 2 ^ n
  by (auto simp: 3 harry-meas-False-True sum.distrib algebra-simps length-seqs)
also have harry-sum (Suc n) = (∑ xs ∈ seqs n. 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 (∑ xs ∈ seqs n. harry-meas (xs @ [True])) = int (harry-sum (Suc n))
  - int (harry-sum n)
  by simp
finally have int (∑ x ∈ seqs (Suc (Suc n)). harry-meas x) =
  int (2 * harry-sum (Suc n) + (2 * n + 4) * 2 ^ n)
  unfolding of-nat-add by (simp add: algebra-simps)
\[ \sum_{x \in \text{seqs} (\text{Suc} (\text{Suc} n))} (\text{harry-meas } x) = (2 \times \text{harry-sum} (\text{Suc} n) + (2 \times n + 4) \times 2^n) \text{ by linarith} \]

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

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

Solving the recurrence gives us the following solution:

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

The average is now a simple consequence:

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

**corollary** harry-avg \( n \) = \( n \times (n + 1) \) / 4  
**proof** –  
**have** real \( (4 \times \text{harry-sum } n) = n \times (n + 1) \times 2^n \)  
**by** (subst harry-sum-closed-form-aux) auto  
**hence** real \( \text{harry-sum } n \) = \( n \times (n + 1) \times 2^n / 4 \)  
**by** (simp add: field-simps)  
**thus** ?thesis  
**by** (simp add: harry-avg-def field-simps)
**qed**

end

**References**

[1] 60th International Mathematical Olympiad.  