theory *Problem-5*
imports
Main
HOL−Library.Multiset
HOL−Number-Theory.Number-Theory
Common.Future-Library
begin

The Bank of Bath issues coins with an H on one side and a T on the other. Harry has n of these coins arranged in a line from left to right. He repeatedly performs the following operation: if there are exactly $k > 0$ coins showing H, then he turns over the kth coin from the left; otherwise, all coins show T and he stops.

a) Show that, for each initial configuration, Harry stops after a finite number of operations.

b) Find the average number of steps Harry will take over all 2^n possible initial configurations.

1 Definitions

datatype *coin* = *H* | *T*

fun *flip* :: *coin* ⇒ *coin* **where**
flip H = *T* |
flip T = *H*

definition *headcount* :: *coin list* ⇒ *nat* **where**
headcount cs = *count* (*mset cs*) *H*

definition *step* :: *coin list* ⇒ *coin list* **where**
step x = (*case headcount x of*
0 ⇒ *x* |
Suc n ⇒ *x*[*n* := *flip* (*x* ! *n*)])

2 A closed formula for the number of steps

definition *heads* :: *coin list* ⇒ *nat set* **where**
heads cs = {*n*. *n* < *length cs* ∧ *cs* ! *n* = *H*}

lemma *finite-heads*[*simp*]: *finite* (*heads cs*)
unfolding *heads-def* **by** *simp*

lemma *headcount-heads*: *headcount cs* = *card* (*heads cs*)
unfolding *headcount-def heads-def*
by (*smt Collect-cong count-mset length-filter-conv-card*)

lemma *headcount-le*: *headcount cs* ≤ *length cs*
by (*metis count-mset headcount-def length-filter-le*)

definition *steps* :: *coin list* ⇒ *nat* **where**
steps cs = *headcount cs* + *2*∗∑ (*heads cs*) − *2*∗∑ {*0*..<*headcount cs*}

3 Lifting *steps* from *nat* to *int*

lemma *steps-sub*:
2∗∑ {*0*..<*headcount cs*} ≤ *2*∗∑ (*heads cs*)
using *sum-min* **unfolding** *headcount-heads* **by** *simp*

lemma *steps-alt*:
int (*steps cs*) = *int* (*headcount cs*) + *int* (*2*∗∑ (*heads cs*)) − *int* (*2*∗∑ {*0*..<*headcount cs*})
proof −

have *int* (*headcount cs*) + *int* (2 ∗ $\sum$ (*heads cs*))
≥ *int* (2 ∗ $\sum$ {0..<*headcount cs*})
using *steps-sub*[*of cs*] **by** *linarith*
then show *?thesis*
unfolding *steps-def int-ops*(6) *int-plus*
by *auto*
qed

4 *steps* describes the behavior of *step*

lemma *step-stopped*:
H ∉ *set x* ⟹ *steps x* = 0
proof −
assume *H* ∉ *set x*
then have *headcount x* = 0 **unfolding** *headcount-def*
by *simp*
with *finite-heads* **and** *headcount-heads* **have** *heads x* = {} **by** *simp*
show *steps x* = 0
using ‹*heads x* = {}› **and** ‹*headcount x* = 0› **by** (*simp add*: *steps-def*)
qed

lemma *step-running*:
assumes *H* ∈ *set x*
shows *steps x* = *Suc* (*steps* (*step x*))
proof −
from *assms* **have** *headcount x* ≠ 0
unfolding *headcount-def* **by** *simp*
then obtain *n* **where** *headcount*: *headcount x* = *Suc n*
using *not0-implies-Suc* **by** *blast*
with *assms* **have** *x'*: *step x* = *x*[*n* := *flip* (*x* ! *n*)] **unfolding** *step-def* **by** *simp*

from *headcount* **and** *headcount-le* **have** *n-lt*: *n* < *length x*
by (*metis Suc-le-lessD*)

show *?thesis*
proof (*cases x* ! *n*)
case *H*
then have *heads* (*step x*) = *heads x* − {*n*}
using *x' n-lt* **unfolding** *heads-def*
by (*auto simp add*: *nth-list-update*)
moreover have *n* ∈ *heads x*
using ‹*x* ! *n* = *H*› *n-lt* **unfolding** *heads-def*
by *auto*
ultimately have $\sum$ (*heads* (*step x*)) = $\sum$ (*heads x*) − *n*
by (*simp add*: *sum-diff1-nat*)
moreover have *n* ≤ $\sum$ (*heads x*)
using ‹*n* ∈ *heads x*› *member-le-sum* **by** *fastforce*
ultimately have ∗: $\sum$ (*heads x*) = $\sum$ (*heads* (*step x*)) + *n*
by *simp*
from ‹*n* ∈ *heads x*›
have *headcount x* = *Suc* (*headcount* (*step x*))
unfolding *headcount-heads* ‹*heads* (*step x*) = *heads x* − {*n*}›
by (*intro card.remove*) *simp*
with *headcount* **have** *headcount* (*step x*) = *n* **by** *simp*
with ∗ **have** *int* (*steps x*) = *int* (*steps* (*step x*)) + 1
unfolding *steps-alt*
by (*simp add*: *headcount*)
thus *?thesis* **by** *simp*
next
case *T*
then have *heads* (*step x*) = *insert n* (*heads x*)
using *x' n-lt* **unfolding** *heads-def*

```
    by (auto simp add: nth-list-update)
  moreover have n ∉ heads x
    using ‹x ! n = T› n-lt unfolding heads-def
    by auto
  ultimately have ∑ (heads (step x)) = ∑ (heads x) + n
    by simp
  moreover have headcount (step x) = Suc (headcount x)
    unfolding headcount-heads ‹heads (step x) = insert n (heads x)›
    using ‹n ∉ heads x›
    by (intro card-insert-disjoint; auto)
  ultimately have int (steps x) = int (steps (step x)) + 1
    unfolding steps-alt
    by (simp add: headcount)
  thus ?thesis by simp
 qed
qed

corollary steps-stopped-iff:
 H ∉ set x ⟷ steps x = 0
 using step-stopped step-running by auto
```

The above two lemmas can be combined using the saturating property of *nat* subtraction.

```
lemma stopped-idempotent:
 H ∉ set x ⟹ step x = x
proof −
 assume H ∉ set x
 then have headcount x = 0 unfolding headcount-def by simp
 thus step x = x unfolding step-def by simp
qed

lemma steps-decreases:
 steps (step x) = steps x − 1
 apply (cases H ∈ set x)
 using step-running step-stopped stopped-idempotent by auto

lemma step-funpow[simp]:
 steps ((step^^k) x) = steps x − k
 apply (induction k)
 by (auto simp add: steps-decreases)

corollary terminates:
 H ∉ set ((step^^steps x) x)
 by (simp add: steps-stopped-iff)

proposition steps-is-least:
 (LEAST k. H ∉ set ((step^^k) x)) = steps x
proof (intro Least-equality)
 show H ∉ set ((step^^steps x) x) by (fact terminates)
 fix y
 assume y-end: H ∉ set ((step^^y) x)
 {
  assume y < steps x
  then have steps ((step^^y) x) ≠ 0
   by simp
  with y-end have False by (simp add: steps-stopped-iff)
 }
 then show steps x ≤ y
  by fastforce
qed
```

5 Average step count

definition *positions* :: *nat* ⇒ *coin list set* **where**
positions n = {*cs*. *length cs* = *n*}

definition *step-avg* :: *nat* ⇒ *real* **where**
step-avg n = (∑ *p* ∈ *positions n*. *steps p*) / *card* (*positions n*)

lemma *coin-finite*: *finite* (*UNIV*::*coin set*)
and *coin-card*: *card* (*UNIV*::*coin set*) = *2*
proof −
have *x* = *H* ∨ *x* = *T* **for** *x* :: *coin*
by (*cases x*) *auto*
hence (*UNIV*::*coin set*) = {*H*, *T*}
by *auto*
moreover have *finite* {*H*, *T*}
by *simp*
ultimately show *finite* (*UNIV*::*coin set*)
by *simp*
have *card* {*H*, *T*} = *2*
by *simp*
with ‹*UNIV* = {*H*, *T*}› **show** *card* (*UNIV*::*coin set*) = *2*
by *simp*
qed

lemma *numpos*: *card* (*positions n*) = *2*^*n*
unfolding *positions-def*
using *card-lists-length-eq*[*OF coin-finite*]
by (*auto simp add*: *coin-card*)

corollary *finite-positions*[*intro*]: *finite* (*positions n*)
using *card.infinite numpos* **by** *fastforce*

lemma *real-sum*: *real* (*sum f S*) = *sum* (*real* ∘ *f*) *S*
using *int-sum of-int-sum* **by** *simp*

lemma *heads-inj*: *length a* = *length b* ⟹ *heads a* = *heads b* ⟹ *a* = *b*
proof −
assume *len*: *length a* = *length b* **and** *heads-eq*: *heads a* = *heads b*
{
fix *n*
assume *n* < *length a*
from *heads-eq* **have** ∗: *n* < *length a* ∧ *a* ! *n* = *H* ⟷ *n* < *length b* ∧ *b* ! *n* = *H*
unfolding *heads-def* **by** *auto*
hence *a* ! *n* = *H* ⟷ *b* ! *n* = *H*
using *len* ‹*n* < *length a*› **by** *simp*
hence *a* ! *n* = *b* ! *n*
by (*metis flip.cases*)
}
with *list-eq-iff-nth-eq len* **show** *?thesis*
by *blast*
qed

corollary *heads-inj-on-positions*: *inj-on heads* (*positions n*)
using *heads-inj* **by** (*auto simp add*: *positions-def intro*: *inj-onI*)

lemma *heads-positions-subset*:
h ∈ *heads* ‘ *positions n* ⟷ *h* ⊆ {*0*..<*n*}
proof
assume *h*: *h* ⊆ {*0*..<*n*}
let *?L* = *map* (λ*n*. *if n* ∈ *h then H else T*) [*0*..<*n*]
have *?L* ∈ *positions n*
unfolding *positions-def* **by** *simp*

```
  moreover have heads ?L = h
    unfolding heads-def
    using h apply auto
    by (meson coin.simps(2))
  ultimately show h ∈ heads ‘ positions n by auto
qed (auto simp add: positions-def heads-def)

lemma card-headcount[simp]: card {p ∈ positions n. headcount p = k} = n choose k
  (is card ?S = -)
proof −
  have ?S = {p ∈ positions n. card (heads p) = k}
    unfolding headcount-heads..
  also have card ... = card {h ∈ heads ‘ positions n. card h = k}
    (is card ?A = card ?B)
  proof (intro bij-betw-same-card bij-betw-imageI)
    show inj-on heads ?A
      by (metis (mono-tags, lifting) inj-onD inj-onI mem-Collect-eq heads-inj-on-positions)
  qed auto
  also have ... = card {h. h ⊆ {0..<n} ∧ card h = k}
    using heads-positions-subset by auto
  also have ... = n choose k
    by (simp add: n-subsets)
  finally show ?thesis.
qed


lemma headcount-image[iff]: headcount ‘ positions n ⊆ {0..n}
  unfolding positions-def
  by (auto simp: headcount-le)

lemma part1: (∑ p ∈ positions n. headcount p) = n * 2^(n−1)
proof −
  have (∑ p ∈ positions n. headcount p) = (∑ k∈{0..n}. (n choose k) * k)
    apply (intro sum-count[where f=λx. x and g=headcount, OF finite-positions, simplified])
    by (simp, fact headcount-image)
  also have ... = (∑ k≤n. (n choose k) * k)
    using atMost-atLeast0 by simp
  finally show ?thesis
    by (simp add: choose-linear-sum ac-simps)
qed

lemma part3: (∑ p ∈ positions n. 2*∑ {0..<headcount p}) = n * (n−1) * 2^(n−2)
proof −
  have ∗: ∑ {0..<n} = ∑ {0..n−1} for n::nat
    by (cases n; auto simp: atLeastLessThanSuc-atLeastAtMost)
  have (∑ p ∈ positions n. 2*∑ {0..<headcount p})
      = (∑ k∈{0..n}. of-nat (card {p ∈ positions n. headcount p = k}) * (2 * ∑ {0..<k}))
    apply (intro sum-count)
    by auto
  also have ... = (∑ k∈{0..n}. (n choose k) * ((k − 1) * Suc (k − 1)))
    by (simp add: ∗ gauss-sum-nat)
  also have ... = (∑ k∈{0..n}. (n choose k) * (k − 1) * k)
  proof −
    have (k − 1) * Suc (k − 1) = (k − 1) * k for k::nat
      by (cases k; auto)
    thus ?thesis
      by (simp add: ac-simps)
  qed
  also have ... = (∑ k∈{2..n}. (n choose k) * (k − 1) * k)
  proof (cases n < 2)
    case False
    then have {0..n} = {0, 1} ∪ {2..n}
      by auto
```

```
  moreover have (∑k∈{0, 1}. (n choose k) * (k - 1) * k) = 0
    by auto
  ultimately show ?thesis by simp
qed auto
also have ... = (∑k∈{2..n}. ((n-2) choose (k-2)) * n * (n - 1))
proof -
  {
    fix k :: nat
    assume k ≥ 2
    then obtain k' where k = Suc (Suc k')
      using add-2-eq-Suc le-Suc-ex by blast
    hence (k - 1) * (k * (n choose k)) = (n - 1) * n * ((n-2) choose (k-2))
     by (auto simp del: mult-Suc-right mult-Suc simp add: binomial-absorption numeral-eq-Suc)
  }
  thus ?thesis
    by (intro sum.cong; auto simp: ac-simps)
qed
also have ... = n * (n - 1) * (∑k∈{2..n}. (n - 2) choose (k - 2))
  by (simp flip: sum-distrib-right)
also have ... = n * (n - 1) * (∑k∈{0..n-2}. (n - 2) choose k)
proof (cases n < 2)
  case False
  then obtain n' where n': n = Suc (Suc n')
    by (metis less-iff-Suc-add not-less-eq numeral-2-eq-2)
  then have [simp]: (∑k∈{2..n}. f k) = (∑k∈{0..n-2}. f (k + 2)) for f :: nat ⇒ nat
    using sum.shift-bounds-cl-nat-ivl[where k=2 and m=0 and n=n' and g=f]
    by (simp add: numeral-eq-Suc)
  show ?thesis by simp
qed auto
also have ... = n * (n - 1) * 2^(n-2)
  using atMost-atLeast0 choose-row-sum by simp
finally show ?thesis.
qed

lemma part2: (∑p ∈ positions n. 2*∑ (heads p)) = n * (n-1) * 2^(n-1)
proof -
  have (∑p ∈ positions n. 2*∑ (heads p)) = (∑h ∈ heads ‘ positions n. 2*∑h)
    using sum.reindex[OF heads-inj-on-positions, symmetric, where n=n and g=λh. 2*∑h]
    by simp
  also have ... = (∑h ∈ Pow {0..<n}. 2*∑h)
  proof -
    have heads ‘ positions n = Pow {0..<n}
      using heads-positions-subset by auto
    thus ?thesis by simp
  qed
  also have ... = n * (n - 1) * 2^(n - 1)
  proof (induction n)
    case (Suc n)
    note IH = this
    show ?case
    proof (cases n)
      case 0
      then show ?thesis by simp eval
    next
      case (Suc n')
      have Pow {0..<Suc n} = Pow {0..<n} ∪ insert n ‘ Pow {0..<n}
        by (simp add: Pow-insert atLeast0-lessThan-Suc)
      moreover have Pow {0..<n} ∩ insert n ‘ Pow {0..<n} = {}
        by auto
      moreover have inj-on (insert n) (Pow {0..<n})
        by (force intro: inj-onI)
      moreover have x∈Pow {0..<n} ⟹ ∑ (insert n x) = n + ∑x for x
        apply (intro sum.insert)
```

```
      apply auto
      using infinite-super by blast
    ultimately show ?thesis
      apply (simp add: sum.union-disjoint sum.reindex sum.distrib Suc.IH card-Pow)
      by (simp add: ‹n = Suc n'› algebra-simps)
  qed
 qed simp
 finally show ?thesis.
qed

lemma stepsum:
 sum steps (positions n)
 = 2^(n-1) * (n^2 + n) div 2
 (is ?L = ?R)
proof -
 have int (∑ p ∈ positions n. steps p)
 = int (∑ p ∈ positions n. headcount p)
 + int (∑ p ∈ positions n. 2*∑ (heads p))
 - int (∑ p ∈ positions n. 2*∑ {0..<headcount p})
   unfolding int-sum steps-alt sum-subtractf sum.distrib..
 hence int ?L = int ?R
   unfolding part1 part2 part3
   apply (cases n rule: fib.cases)
   by (auto simp add: algebra-simps power2-eq-square)
 thus ?L = ?R by linarith
qed


theorem avg-steps:
 step-avg n = real (n^2 + n) / 4
proof -
 have even: even (n^2 + n) unfolding power2-eq-square by simp
 hence real (2 ^ (n - 1) * (n^2 + n) div 2) / real (2 ^ n)
    = real (2 ^ (n - 1) * ((n^2 + n) div 2)) / real (2 ^ n)
   by fastforce
 also have ... = 2^(n - 1) * real (n^2 + n) / 2^Suc n
   by (simp add: real-of-nat-div[OF even])
 also have ... = real (n^2 + n) / 4
   apply (cases n)
   by auto
 finally show ?thesis
   unfolding step-avg-def numpos stepsum.
qed

end
```