theory *Problem-4*
  imports *Main*
begin

Let a and b be positive integers, and let A and B be finite sets of integers satisfying

- A and B are disjoint.
- if an integer i belongs to either A or B then either $i + a$ belongs to A or $i - b$ belongs to B.

  Prove that $a|A| = b|B|$.

```
context
  fixes a b :: int
  fixes A B :: int set
  assumes pos: a > 0  b > 0
  assumes finite: finite A  finite B
  assumes disjoint: A ∩ B = {}
  assumes property: ∀ i ∈ A ∪ B. i + a ∈ A ∨ i − b ∈ B
begin

definition allows (infix allows 50) where
  x allows i ⟷ (x ∈ A ∧ i + a = x) ∨ (x ∈ B ∧ i − b = x)

lemma allows-right-unique:
  assumes x allows i and x allows j
  shows i = j
proof (rule ccontr)
  assume i ≠ j
  with assms have x ∈ A ∧ x ∈ B unfolding allows-def by auto
  with disjoint show False by auto
qed

definition allowers where
  allowers i = {x. x allows i}

lemma has-allowers: i ∈ A ∪ B ⟹ allowers i ≠ {}
  unfolding allowers-def allows-def using property by auto

lemma allowers-in-set: allowers i ⊆ A ∪ B
  unfolding allowers-def allows-def by auto

lemma allowers-finite: finite (allowers i)
  using allowers-in-set apply (rule finite-subset)
  using finite by auto

lemma card-allowers: i ∈ A ∪ B ⟹ card (allowers i) > 0
  using has-allowers allowers-finite by fastforce

lemma allowers-inj: inj-on allowers (A ∪ B)
proof (rule inj-onI)
  fix i j
  assume i ∈ A ∪ B  j ∈ A ∪ B
  with has-allowers obtain x where x ∈ allowers i by auto
  moreover assume allowers i = allowers j
  ultimately have x allows i and x allows j
    by (auto simp: allowers-def)
  thus i = j by (rule allows-right-unique)
qed

lemma one-allower:
  assumes i ∈ A ∪ B
```

```
  shows card (allowers i) = 1
proof –
  let ?C = allowers ‘ (A ∪ B)
  have pairwise disjnt ?C
  proof
    fix P Q
    assume P ∈ ?C and Q ∈ ?C and P ≠ Q
    then obtain i and j
      where ∗: allowers i = P  allowers j = Q
        and i ∈ A ∪ B  j ∈ A ∪ B
      by auto
    with ‹P ≠ Q› have i ≠ j by auto
    {
      fix x
      assume x ∈ P and x ∈ Q
      with ∗ have x allows i and x allows j
        unfolding allowers-def by auto
      with allows-right-unique have i = j by auto
      with ‹i ≠ j› have False..
    }
    thus disjnt P Q unfolding disjnt-def by auto
  qed
  moreover have P ∈ ?C ⟹ finite P for P
    using allowers-finite by auto
  ultimately have sum-card: card (⋃ ?C) = sum card ?C
    by (intro card-Union-disjoint; auto)

  have ⋃ ?C ⊆ A ∪ B
    apply (intro Union-least)
    using allowers-in-set by auto
  hence card (⋃ ?C) ≤ card (A ∪ B)
    using finite by (intro card-mono; auto)
  moreover have card ?C = card (A ∪ B)
    using allowers-inj by (intro card-image)
  ultimately have sum-card-le: sum card ?C ≤ card ?C
    using sum-card by simp

  show card (allowers i) = 1 when ¬ card (allowers i) > 1
    using card-allowers assms that by force
  show ¬ card (allowers i) > 1
  proof
    assume card (allowers i) > 1
    moreover have sum card ?C = card (allowers i) + sum card (?C − {allowers i})
      apply (intro sum.remove)
      using finite assms by auto
    moreover have sum card (?C − {allowers i}) ≥ card (?C − {allowers i})
      apply (intro sum-bounded-below[where K=(1::nat), simplified])
      using card-allowers by fastforce
    moreover have Suc (card (?C − {allowers i})) = card ?C
      apply (intro card.remove[symmetric])
      using finite assms by auto
    ultimately have sum card ?C > card ?C
      by auto
    with sum-card-le show False by simp
  qed
qed

definition allower where
  allower i = the-elem (allowers i)

lemma the-elem-in-set:
  assumes is-singleton S
  shows the-elem S ∈ S
```

```
  using assms by (metis is-singleton-the-elem singletonI)

lemma allower-allows:
  assumes i ∈ A ∪ B
  shows allower i ∈ allowers i and (allower i) allows i
proof –
  from assms have is-singleton (allowers i)
    using one-allower is-singleton-altdef by blast
  thus allower i ∈ allowers i
    unfolding allower-def by (intro the-elem-in-set)
  thus (allower i) allows i
    unfolding allowers-def..
qed

lemma allower-in-set:
  assumes i ∈ A ∪ B
  shows allower i ∈ A ∪ B
  using allowers-in-set allower-allows assms by auto

lemma allower-bij: bij-betw allower (A ∪ B) (A ∪ B)
  unfolding bij-betw-def
proof
  show inj-on allower (A ∪ B)
    apply (intro inj-onI)
    by (metis allower-allows(2) allows-right-unique)
  hence card (allower ‘ (A ∪ B)) = card (A ∪ B)
    by (intro card-image)
  moreover have allower ‘ (A ∪ B) ⊆ A ∪ B
    using allower-in-set by auto
  ultimately show allower ‘ (A ∪ B) = A ∪ B
    using finite by (intro card-subset-eq; auto)
qed

theorem a ∗ int (card A) = b ∗ int (card B)
proof –
  let ?A = {i ∈ A ∪ B. allower i ∈ A}
  let ?B = {i ∈ A ∪ B. allower i ∈ B}
  have ∗: ?A ∪ ?B = A ∪ B
    using allower-in-set by auto
  have disjoint′: ?A ∩ ?B = {}
    using disjoint by auto

  have in-a: i ∈ ?A ⟹ allower i = i + a for i
    using allower-allows allows-def
    by (metis (mono-tags, lifting) IntI disjoint empty-iff mem-Collect-eq)
  have in-b: i ∈ ?B ⟹ allower i = i − b for i
    using allower-allows allows-def
    by (metis (mono-tags, lifting) IntI disjoint empty-iff mem-Collect-eq)

  have allower ‘ ?A ⊆ A and allower ‘ ?B ⊆ B
    by auto
  moreover have allower ‘ (?A ∪ ?B) = A ∪ B
    using allower-bij unfolding ∗ bij-betw-def by auto
  ultimately have allower ‘ ?A = A and allower ‘ ?B = B
    using disjoint by auto
  moreover have inj-on allower ?A and inj-on allower ?B
    using bij-betw-imp-inj-on allower-bij inj-on-subset ∗ by blast+
  ultimately have cards: card ?A = card A   card ?B = card B
    using card-image by fastforce+

  have ∑(A ∪ B) = sum allower (A ∪ B)
    using sum.reindex-bij-betw[OF allower-bij, of λx. x]..
  also have ... = sum allower (?A ∪ ?B)
```

```
    using ∗ by simp
  also have ... = sum allower ?A + sum allower ?B
    using finite disjoint′ by (intro sum.union-disjoint; auto)
  also have ... = (∑ i ∈ ?A. i + a) + (∑ i ∈ ?B. i − b)
    using in-a in-b sum.cong
    by (metis (no-types, lifting))
  also have ... = (∑ ?A + a ∗ int (card ?A)) + (∑ ?B − b ∗ int (card ?B))
    by (simp add: sum.distrib sum-subtractf)
  also have ... = (∑ ?A + ∑ ?B) + a ∗ int (card A) − b ∗ int (card B)
    unfolding cards
    by (simp add: ac-simps)
  also have ... = ∑ (?A ∪ ?B) + a ∗ int (card A) − b ∗ int (card B)
  proof −
    have ∑ (?A ∪ ?B) = ∑ ?A + ∑ ?B
      using finite disjoint′ by (intro sum.union-disjoint; auto)
    thus ?thesis by simp
  qed
  also have ... = ∑ (A ∪ B) + a ∗ int (card A) − b ∗ int (card B)
    unfolding ∗..
  finally show a ∗ int (card A) = b ∗ int (card B)
    by simp
qed

end
end
```